HARDWARE IMPLEMENTATION OF SOFTWARE DEFINED RADIO BASED ON DYNAMIC PARTIAL RECONFIGURATION

By

Sherif Mohamed Hosny Afifi

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2018

HARDWARE IMPLEMENTATION OF SOFTWARE DEFINED RADIO BASED ON DYNAMIC PARTIAL RECONFIGURATION

By

Sherif Mohamed Hosny Afifi

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Under the Supervision of

Prof. Dr. Ahmed H. Khalil	**Dr. Hassan Mostafa Hassan**
Professor Electronics and Electrical Communications Engineering Department Faculty of Engineering, Cairo University	Assistant Professor Electronics and Electrical Communications Engineering Department Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2018

HARDWARE IMPLEMENTATION OF SOFTWARE DEFINED RADIO BASED ON DYNAMIC PARTIAL RECONFIGURATION

By

Sherif Mohamed Hosny Afifi

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Approved by the
Examining Committee

Prof. Dr. Ahmed H. Khalil	Thesis Main Advisor
Prof. Dr. Mohamed F. Abu-ElYazeed	Internal Examiner
Dr. Magdy A. El-Moursy (Electronics Research Institute)	External Examiner

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2018

Engineer's Name: Sherif Mohamed Hosny Afifi
Date of Birth: 15/2/1993
Nationality: Egyptian
E-mail: Sherif1521993@gmail.com
Phone: 01005264022
Address: 39 El-Maraghi street, El-Agouza
Registration Date: 1/3/2015
Awarding Date: 2018
Degree: Master of Science
Department: Electronics and Communications Engineering

Supervisors:

Prof. Dr. Ahmed H. Khalil
Dr. Hassan Mostafa Hassan

Examiners:

Prof. Dr. Ahmed H. Khalil (Thesis Main Advisor)
Prof. Dr. Mohamed F. Abu-ElYazeed (Internal Examiner)
Dr. Magdy A. El-Moursy (External Examiner)
(Electronics Research Institute)

Title of Thesis:

HARDWARE IMPLEMENTATION OF SOFTWARE DEFINED RADIO BASED ON DYNAMIC PARTIAL RECONFIGURATION

Key Words:
Software Defined Radio; Dynamic Partial Reconfiguration; Field Programmable Gate Array

Summary:

This work implements SDR transceiver system for five wireless communication standards: Bluetooth, Wi-Fi, 2G, 3G, and LTE on Zynq-7000 evaluation kit. The new DPR technique is used to switch between different multi-standard communication systems on the same FPGA partition. Implementing SDR using DPR combines the advantage of hardware performance and software flexibility. A test environment is established to measure the effectiveness of the new technique.

Abstract

Dynamic Partial Reconfiguration (DPR) has been used extensively over the past few years allowing reconfiguration of Field Programmable Gate Arrays (FPGAs) during the run time. FPGA is considered one of the best solutions for implementing reconfigurable hardware. The concept of hardware reconfiguration exists for several decades and passed through many evolution phases. With the aid of DPR, multi-standard Software Defined Radio (SDR) system can be implemented in order to save power and area extensively. Over the past few years, wireless communication standards witnessed great and rapid evolution. The market is always acquiring higher data rates and more special services. This leads to increasing the design complexity, area, and power consumption. Deploying DPR technology on FPGAs made it feasible to design and manufacture all wireless communications standards on the same hardware. Loading each standard on demand reduces area utilization and power consumption.

SDR is a communication system whose physical layer is used to do all the computations using the software. The communication blocks in ordinary radio transceivers are designed in a fixed environment to process a certain waveform. SDR is able to process many waveforms since it can be easily configured using software. It is becoming achievable, as the flexibility in the digital front-end reconfiguration increases. One of the advantages of implementing the SDR is increasing the flexiblity that aids in performing dynamic and real-time reconfiguration. Another advantage of using SDR is the efficient use of resources under varying conditions. Bottom line is, the hardware flexibility allows the SDR dynamic system to implement different standards within real-time without the need to switch off the system. The fundamental challenge facing the deployment of SDR is how to achieve sufficient computational capacity, in particular for processing wide-band high bit rate waveforms, within acceptable size and weight factors, within acceptable unit costs, and reduced power consumption compared to the communication standards implemented in current mobile phones.

This work implements SDR transceiver system for five wireless communication standards: Bluetooth, Wi-Fi, 2G, 3G, and LTE on Zynq-7000 evaluation kit. The new DPR technique is used to switch between different multi-standard communication systems on the same FPGA partition. Implementing SDR using DPR combines the advantage of hardware performance and software flexibility. A test environment is established to measure the effectiveness of the new technique. Two approaches are deployed to implement the five transceivers using DPR. The first technique uses a single reconfigurable partition for the transmitter and the receiver. The second technique recommends splitting the design into multi-partitions in order to achieve the best performance for all transceivers. A comparison is performed for the system total area and power consumption between the two DPR approaches and the case of no DPR. The single partition approach achieves reduction of area and power by 10.19% and 76.71% respectively with a reasonable switching time. The multi-partition approach is able to reduce the allocated area and power consumption for all chains. Power reduction for 2G and Bluetooth is 95.43%, for 3G and Wi-Fi is 79.69%, for LTE is 59.09% compared with the case of no DPR.

Acknowledgements

First I would like to devote this work to my family. For my departed father who believed in me and always been supportive ever since I began this long way, thanks for everything you've done. To my beloved mother and sister who were always been holding my back every single predicament.

I would like also to thank my closest friends: Said Hazem, Mahmoud Abd-El-Hameed, Mazen Taha, Mahmoud Fawzy, Karim Gooda, and Ahmed Magdy for being there for me every time I needed your support.

Of course a heartfelt thanks for my best friends: Hagar Hasanain and Aya Alaa for their inspiration and devotion.

I would like to express my thanks to my work managers and colleges: Mohamed Korany, Carole Richard, Amr Baher, Islam El Nader, Mostafa Gamal, Abdulrahman Hussein, Mohamed Nafea, Yahia Khalid, and Mohamed Adel for all their aid and support along this tough road.

Last but of course not least, my deep thanks goes to my master thesis supervisors Dr. Hassan Mostafa and Prof. Ahmed Hussein for their guidance during the master preparation.

Disclaimer

I herby declare that this thesis is my own original work, and that no part of it has been submitted for a degree qualification at any other university or institute.

I further declare that I have appropriately ackowleged al sources used have cited them in the refernces section.

Name:

Signature:

Table of Contents

Abstract **i**

Acknowledgements **ii**

Disclaimer **iii**

Table of Contents **iv**

List of Tables **viii**

List of Figures **ix**

Nomenclatures **xi**

1 INTRODUCTION **1**
- 1.1 Problem Domain 1
- 1.2 Thesis Outline 2

2 LITERATURE SURVEY **3**
- 2.1 Background and Related Work 3
- 2.2 Communication System in Details 3
- 2.3 SDR System Overview 4
- 2.4 ZYNQ Board (ZC702) 5
  - 2.4.1 FPGA Evolution History 5
  - 2.4.2 Introduction to The Board 6
  - 2.4.3 CLB Overview 7
- 2.5 FPGA Configuration 9
  - 2.5.1 Configuration Definition 9
  - 2.5.2 Types of Configuration 9
  - 2.5.3 DPR in FPGAs 11
  - 2.5.4 Types of Bit Files 11
- 2.6 DPR Techniques 12
  - 2.6.1 External Mode Using JTAG 12
  - 2.6.2 Internal Mode 12
    - 2.6.2.1 PCAP on PS Side 13
    - 2.6.2.2 ICAP on PL Side 13
- 2.7 Summary 14

3 SDR TRANSMITTER DESIGN **17**
- 3.1 SDR System Overview 17
- 3.2 Bluetooth Transmitter 18
  - 3.2.1 Segmentation 18
  - 3.2.2 HEC Generator, CRC, and Whitening 19
  - 3.2.3 Repetition and Hamming Encoders 19

- 3.2.4 Chain Utilization . . . 21
- 3.3 Wi-Fi Transmitter . . . 21
  - 3.3.1 Scrambler . . . 21
  - 3.3.2 Convolutional Encoder . . . 21
  - 3.3.3 Puncture . . . 22
  - 3.3.4 Interleaver . . . 23
  - 3.3.5 OFDM Section . . . 24
  - 3.3.6 Chain Utilization . . . 25
- 3.4 2G Transmitter Chain . . . 25
  - 3.4.1 CRC . . . 25
  - 3.4.2 Convolutional Encoder . . . 26
  - 3.4.3 Interleaver . . . 27
  - 3.4.4 Burst Formation . . . 27
  - 3.4.5 Differential Coding . . . 28
  - 3.4.6 Chain Utilization . . . 28
- 3.5 3G Transmitter Chain . . . 28
  - 3.5.1 CRC . . . 30
  - 3.5.2 Segmentation . . . 31
  - 3.5.3 Convolutional Encoder . . . 31
  - 3.5.4 Code Block Concatenation . . . 32
  - 3.5.5 Interleaver . . . 32
  - 3.5.6 Code Division Multiplexing . . . 34
  - 3.5.7 Chain Utilization . . . 34
- 3.6 LTE Transmitter Chain . . . 35
  - 3.6.1 CRC . . . 35
  - 3.6.2 Segmentation . . . 35
  - 3.6.3 Turbo Encoder . . . 35
  - 3.6.4 Rate Matching . . . 35
  - 3.6.5 Code Block Concatenation . . . 37
  - 3.6.6 Scrambler . . . 37
  - 3.6.7 OFDM Section . . . 39
  - 3.6.8 Chain Utilization . . . 40
- 3.7 Summary . . . 41

4 SDR RECEIVER DESIGN **42**
- 4.1 SDR Receiver Overview . . . 42
- 4.2 Bluetooth Receiver . . . 42
  - 4.2.1 Demapper . . . 42
  - 4.2.2 Repetition and Hamming Decoders . . . 42
  - 4.2.3 Dewhitening, De-HEC, and De-CRC . . . 43
  - 4.2.4 Chain Utilization . . . 43
- 4.3 Wi-Fi Receiver . . . 44
  - 4.3.1 OFDM Section . . . 45
  - 4.3.2 Deinterleaver . . . 45
  - 4.3.3 Depuncture . . . 46
  - 4.3.4 Viterbi Decoder . . . 46
  - 4.3.5 Descrambler . . . 51

    - 4.3.6 Chain Utilization . . . 52
- 4.4 2G Receiver . . . 52
    - 4.4.1 Differential Decoding . . . 53
    - 4.4.2 Burst Deformation . . . 53
    - 4.4.3 Deinterleaver . . . 53
    - 4.4.4 Viterbi Decoder . . . 53
    - 4.4.5 De-CRC . . . 53
    - 4.4.6 Chain Utilization . . . 53
- 4.5 3G Receiver . . . 54
    - 4.5.1 Code Division Multiplexing . . . 54
    - 4.5.2 Deinterleaver . . . 54
    - 4.5.3 Deconcatenation . . . 55
    - 4.5.4 Viterbi Decoder . . . 55
    - 4.5.5 Desegmentation . . . 56
    - 4.5.6 De-CRC . . . 56
    - 4.5.7 Chain Utilization . . . 57
- 4.6 LTE Receiver . . . 57
    - 4.6.1 OFDM Section . . . 57
    - 4.6.2 Descrambler . . . 57
    - 4.6.3 Code Block Deconcatenation . . . 57
    - 4.6.4 Rate Dematching . . . 58
    - 4.6.5 Turbo Decoder . . . 59
    - 4.6.6 Desegmentation . . . 59
    - 4.6.7 De-CRC . . . 59
    - 4.6.8 Chain Utilization . . . 59
- 4.7 Summary . . . 60

5 FPGA PROTOTYPING 61
- 5.1 Test Environment . . . 61
- 5.2 Block Design Implementation . . . 64
- 5.3 DPR Flow Steps . . . 65
- 5.4 DPR Proposed Approaches . . . 71
    - 5.4.1 Single Partition Approach . . . 73
    - 5.4.2 Multi-Partition Approach . . . 74
- 5.5 Simulation Results . . . 75
    - 5.5.1 Fixation Error . . . 75
    - 5.5.2 Area and Power Measuremnet . . . 78
- 5.6 Summary . . . 80

6 CONCLUSION AND PROPOSED FUTURE WORK 82
- 6.1 Conclusion . . . 82
- 6.2 Proposed Future Work . . . 82

References 83

Appendix A Partioning Algorithm 86

Appendix B FPGA Prototyping Code **93**

Appendix C Reconfiguration Algorithm **98**

List of Tables

1 . . . xi

2.1 FPGA Resources [23] . . . 8
2.2 Configuration Tools Bandwidth [23] . . . 13

3.1 Wi-Fi Various MCS [36] . . . 17
3.2 DQPSK Mapping Relationship [35] . . . 20
3.3 Bluetooth Transmitter Chain Area Utilization . . . 21
3.4 BPSK Modulation Scheme [36] . . . 25
3.5 QPSK Modulation Scheme [36] . . . 25
3.6 Wi-Fi Transmitter Chain Area Utilization . . . 25
3.7 2G Transmitter Chain Area Utilization . . . 28
3.8 3G CRC Polynomial Equations [46] . . . 31
3.9 Number of Segments [46] . . . 33
3.10 3G Transmitter Chain Area Utilization . . . 35
3.11 LTE Transmission Schemes [48] . . . 40
3.12 LTE Transmitter Chain Area Utilization . . . 40

4.1 Bluetooth Receiver Chain Area Utilization . . . 44
4.2 Hamming Distance in case of coding rate 1/2 [36] . . . 51
4.3 Hamming Distance in case of coding rate 1/3 [36] . . . 51
4.4 Wi-Fi Receiver Chain Area Utilization . . . 52
4.5 2G Receiver Chain Area Utilization . . . 54
4.6 3G Receiver Chain Area Utilization . . . 57
4.7 LTE Receiver Chain Area Utilization . . . 60

5.1 Transmitter Chains Area Utilization . . . 78
5.2 Receiver Chains Area Utilization . . . 79
5.3 Switching Time . . . 80
5.4 Area and Power Comparison . . . 81

List of Figures

2.1 Ideal communication system 4
2.2 Basic software controlled radio [22] 5
2.3 Softcore and hardcore processors [26] 6
2.4 Zynq board [23] 7
2.5 Zynq board internal structure [23] 8
2.6 Arrangement of slices within the CLB [28] 9
2.7 CLB column and row connections [28] 10
2.8 FPGA Layers 10
2.9 DPR configuration criteria [29] 11
2.10 DPR controllers [30] 12
2.11 AXI HWICAP core [33] 14
2.12 PRC in DPR system [34] 15
2.13 PRC virtual socket [34] 15
2.14 PRC fetch path [34] 16

3.1 Block signals in each chain 18
3.2 Bluetooth transmitter block diagram [35] 19
3.3 Segmentation controller algorithm 20
3.4 Bluetooth hamming encoder [35] 21
3.5 Wi-Fi transmitter chain block diagram [36] 22
3.6 Wi-Fi convolutional encoder [36] 23
3.7 Wi-Fi puncture algorithm [36] 23
3.8 Wi-Fi IFFT block diagram 24
3.9 2G transmitter chain block diagram [42] 26
3.10 2G interleaver matrix [42] 27
3.11 Burst data formation [42] 27
3.12 Burst formation block diagram 28
3.13 Burst formation controller algorithm 29
3.14 3G transmitter chain block diagram [45] 30
3.15 3G segmentation FSM 31
3.16 3G convolutional encoder [46] 32
3.17 3G interleaver block diagram [46] 32
3.18 3G first interleaver [46] 33
3.19 CDM block diagram [47] 34
3.20 LTE transmitter chain block diagram [48] 36
3.21 4G segmentation FSM 37
3.22 4G turbo encoder [48] 38
3.23 4G Rate matching block diagram [49] 39
3.24 OFDM section block diagram 39

4.1 Bluetooth receiver block diagram 42
4.2 Dewhitening polynomial [35] 43
4.3 De-HEC polynomial [35] 43

4.4 De-CRC polynomial [35] . . . 43
4.5 Wi-Fi receiver chain block diagram . . . 44
4.6 Wi-Fi FFT and demapper block diagram . . . 45
4.7 Wi-Fi depuncturing algorithm [36] . . . 46
4.8 Viterbi block diagram [50] . . . 47
4.9 Viterbi decoder algorithm flow chart . . . 48
4.10 Convolutional encoder terellis example . . . 49
4.11 Trellis diagram for error decoding . . . 49
4.12 BMU block diagram . . . 50
4.13 Distance calculator for coding rate 1/2 . . . 50
4.14 Distance calculator for coding rate 1/3 . . . 50
4.15 Wi-Fi descrambler [36] . . . 51
4.16 2G receiver chain block diagram . . . 52
4.17 2G de-CRC . . . 54
4.18 3G receiver chain block diagram . . . 55
4.19 3G deinterleaver block diagram . . . 56
4.20 3G de-CRC . . . 56
4.21 LTE receiver chain block diagram . . . 58
4.22 OFDM section block diagram . . . 58
4.23 Turbo decoder block diagram [51] . . . 59
4.24 LTE de-CRC . . . 59

5.1 Zynq-7000 SoC ZC702 evaluation kit peripherals [27] . . . 62
5.2 Test environment on the PL side . . . 63
5.3 C-code flow chart . . . 63
5.4 Hardware initialization steps . . . 64
5.5 Block design connections . . . 66
5.6 PRC configuration . . . 67
5.7 Connection automation options . . . 67
5.8 HDL wrapper creation . . . 68
5.9 Pblock selection . . . 69
5.10 Pblock properties . . . 70
5.11 Exporting Hardware . . . 71
5.12 SDK project creation . . . 71
5.13 Software program execution . . . 72
5.14 DPR overall system . . . 72
5.15 Single partition approach block diagram . . . 73
5.16 Single partition approach floor plan . . . 73
5.17 Virtex-7 RP physical constraints . . . 75
5.18 Partitioning algorithm flow chart . . . 76
5.19 Multi-partition approach block diagram . . . 76
5.20 Multi-partition approach floor plan . . . 77
5.21 Wi-Fi 16-QAM fraction part fixation . . . 78
5.22 Area utilization (LUTs) . . . 79
5.23 Swithcing time (ms) . . . 80
5.24 Average power (mW) . . . 81

List of Nomenclature

Abbreviation	**Description**
2G	Second Mobile Generation.
3G	Third Mobile Generation.
3GPP	3rd Generation Partnership Project.
ADC	Analog to Digital Converter.
ASIC	Application Specific Integrated Circuit.
AXI	Advanced Extensible Interface.
BPSK	Binary Phase Shift Keying.
BRAM	Block Read Access Memory.
CDMA	Code Division Multiple Access.
CRC	Cyclic Redundancy Check.
DAC	Digital to Analog Converter.
DDR	Double Data Rate.
DPR	Dynamic Partial Reconfiguration.
DSP	Digital Signal Processing.
FEC	Forward Error Correction.
FIFO	First Input First Output.
FPGA	Field Programming Gate Array.
FSM	Finite State Machine.
GPP	General Purpose Processor.
HDL	Hardware Description Language.
GSM	Global System for Mobile communications.
ICAP	Internal Configuration Access Port.
IFFT	Inverse Fast Fourier Transform.
IEEE	Institute of Electrical and Electronic Engineers.
ILA	Interactive Logic Analyzer.
JTAG	Joint Test Action Group.
LTE	Long Term Evolution.
LUT	Look Up Table.
OFDM	Orthogonal Frequency Division Multiplexing.
PL	Programmable Logic.
PRC	Partial Reconfiguration Controller.
PS	Processing System.
PLB	Programmable Logic Block.
RM	Reconfigurable Module.
RP	Reconfigurable Partition.

QAM	Quadrature Amplitude Modulation.
QPSK	Quadrature Shift Keying.
SCFDMA	Single Carrier Frequency Division Multiple Access.
SDK	Software Development Kit.
SDR	Software Defined Radio.
SoC	System on Chip.
TTI	Transmission Time Interval.
SRAM	Static Random Access Memory.
UMTS	Universal Mobile Telecommunications System.

Chapter 1: Introduction

Modern wireless communication systems are witnessing a new era of high data rates, and consequently higher power consumption of mobile batteries due to the powerful baseband signal processing. Researchers try to minimize the area and power consumed by the signal processing in various wireless communication standards, with the aid of different algorithms and software techniques.

Wireless communication standards are continuously changing and upgrading to achieve better performance, new features, higher throughput, and new technologies. IC fabrication is becoming more difficult and costly impractical. This is due to the increase of number of standards and technologies (such as GSM, UMTS, LTE, Wi-Fi, and Bluetooth) required to be implemented in different handset devices. The large number of analog and digital blocks in each standard consume large amount of power, which is a scarce resource for the handset [1]. In order to solve this issue, both user terminal and base station need to adopt dynamic switching between multiple communication standards. This is denoted by Software Defined Radio (SDR) [2, 3].

Various technologies can be used to implement SDR such as DSPs and Field Programmable Gate Arrays (FPGAs). The FPGA provides the best balance between performance, low power consumption, and short design cycle [4]. Improvements in FPGAs make the realization of SDR possible [5]. FPGAs are the usual targeted technology for many development efforts. This is due to their low cost and their ability to support Dynamic Partial Reconfiguration (DPR) technology [6]. SDR is expected to be the most appropriate answer to multi-standards handset design challenges. By applying the concept of DPR in SDR with the required capabilities, all standards are allowed to be upgraded by software without the need of hardware upgrading [7, 8].

1.1 Problem Domain

DPR provides the modification of a certain part in the device, while the rest remains unchanged and active. The main target is switching between different communication standards and different modulation schemes in each standard rapidly with the aid of DPR such that, all systems seem to be working together at the same time [9, 10]. DPR is a promising technology which offers the reconfiguration of a specified partition in the FPGA during the run time, which helps in implementing a multi-standard SDR [11, 12].

SDR addresses the switching between different standards on the same FPGA partition, to perform the baseband signal processing without affecting the overall performance of any of the standards [13].

The implementation of a high-speed reconfiguration time dynamic cognitive radios using the Zynq FPGA is presented [7]. Deployment of SDR using DPR technology, made it possible to use of the same specified hardware resource on FPGA for different wireless communication standards found in the mobile phone. This implies that only the specified partition on the FPGA is used for the baseband signal processing of different communication standards. This will result in saving vast amount of power and area [14, 15].

1.2 Thesis Outline

The thesis contains six chapters including this one. Chapter 2 starts by providing a sufficient background on the history of FPGAs and specifications of the used Xilinx board. The chapter then lists the types of FPGA configuration, describing the reason behind choosing the DPR approach. A comparison is performed between all DPR techniques showing the advantages of the chosen technique. A full list of DPR controllers is provided as well.

A full description of the physical layer of the five transceiver chains: Bluetooth, Wi-Fi, 2G, 3G, and 4G and their implementation, are listed in Chapter 3 and Chapter 4. Chapter 3 includes the implemented blocks in the five transmitter chains. The illustration includes the functional specs of each block, the way of implementation, and the utilization on the FPGA. The implemented blocks in the five receiver chains are listed in Chapter 4.

Chapter 5 gives a brief description about the proposed test environment showing the used board peripherals. The C program used to run the DPR flow is illustrated as well. Another comparison is performed between the two proposed approaches in deploying SDR system using DPR showing the pros and cons of each approach. The calculated simulation results showing the effectiveness of the proposed approaches are listed in this chapter. Ultimately, the conclusion and future work are listed in Chapter 6.

Chapter 2: Literature Survey

2.1 Background and Related Work

DPR technique is used to switch between different configurations of LTE OFDM modulators in [16]. Variations are based on the size of the IFFT, number of subcarriers, cyclic prefix and window length. The implemented design on Virtex-7 is divided on four reconfigurable partitions and a single static partition for the FFT. Similar design for LTE FFT is proposed in [17], where configuration is dependent on the FFT size. Same criteria is used to switch between modulators and demodulators in [18, 19].

The proposed dynamic cognitive radios in [14] implement the physical layer on the FPGA Programmable Logic (PL) and the Medium Access Control (MAC) layer on the ARM processor. Switching between different baseband modules is performed using custom partial reconfiguration controller to achieve high reconfiguration speed. Virtex-7 is used host the physical layer blocks.

The SDR physical layer implemented on Virtex-4 using DPR technique in [6] uses internal and external configuration modes. The reconfiguration time overhead is taken in consideration.

The contribution of this work in deploying SDR, is implementing the physical layer of five transceiver chains: Bluetooth V2.0, IEEE 802.11a, GSM, UMTS, and LTE on the same reconfigurable hardware using the DPR technology. The proposed approach is proving itself in overcoming most of the challenges by saving area and power consumption.

2.2 Communication System in Details

Figure 2.1 shows the main blocks in a modern communication system. It is composed of a DSP unit, digital and analog converters (DAC, ADC), RF section, and wide band antenna. This work concentrates on the DSP block. Shown below a brief description for each block in the system:

1. **Digital Signal Processing Block**:
   At the transmitter side, this block is responsible for signal adaptation to be sent over the channel. Signal adaptation includes encryption, error correction coding schemes, modulation, and further more. Meanwhile, in the receiver this block is responsible for extracting the original information by reconstructing the signal using demodulation, decoding, and decryption. This block increases the flexibility of radio development.

2. **DAC/ADC Blocks**:
   Analog digital converters are used to transfer the signal between the analog and digital domains. Using ADC, the received signal is digitized to be processed digitally using the DSP block. The digital representation depends on the sampling rate that leads to some information loss. The DAC is used to reconstruct the signal to its original image.

3. **RF Front End Block**:
   This block contains Low Noise Amplifier (LNA), filters, and Power Amplifiers (PA).

4. **Antenna**:
   The antenna is a passive device used to capture the electromagnetic waves from the surrounding media and converts it to an electrical signal. The antenna design complexity varies from a single antenna to multiple antenna arrays. Smart antenna is established using an antenna array that uses the signal processing algorithms to locate the direction of signal arrival. Reconfigurable antennas are capable of changing their frequency for adaptable systems.

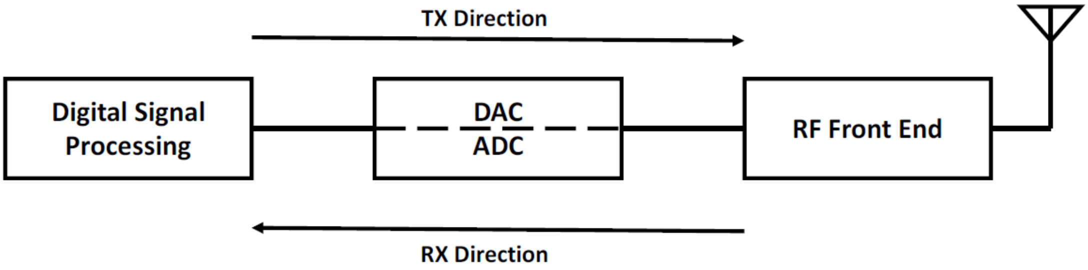


Figure 2.1: Ideal communication system

2.3 SDR System Overview

A software-defined radio is a radio in which some or all of the physical layer functions are software defined [20, 21]. Implication of the term software defined is that different waveforms are supported by modifying the software or firmware without changing the hardware. The basic idea of software controlled radio is illustrated in Figure 2.2. The advantages of deploying the SDR are:

1. Efficient use of resources under varying conditions. For example, a low-power waveform can be selected if the radio is running on a low battery. A high-throughput waveform can be selected to quickly download a file.
2. Opportunistic frequency reuse (cognitive radio). An SDR can take advantage of underutilized spectrum. If the owner of the spectrum is not using it, an SDR can borrow the spectrum until the owner comes back.

Despite the advantages of SDR, there are some challenges facing its deployment in mobile phones:

1. The SDR should not be constrained by the carrier frequency. Meanwhile, since most of antennas are mechanical structures, they are not easily tuned dynamically.
2. A fundamental challenge with SDR is how to achieve sufficient computational capacity, in particular for processing wide-band high bit rate waveforms, within acceptable size and weight factors, within acceptable unit costs, and with acceptable power consumption.

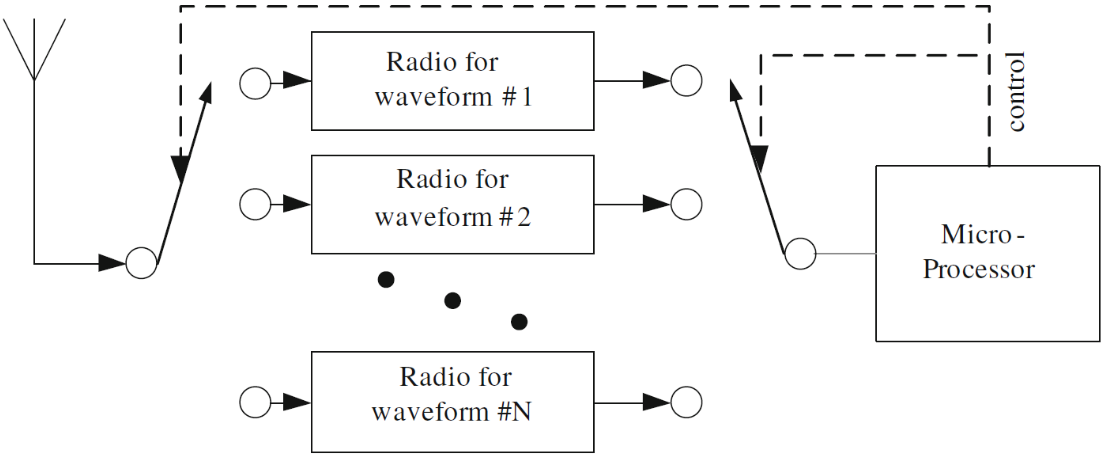


Figure 2.2: Basic software controlled radio [22]

The digital signal processing part in the communication system can be carried on different hardware platforms such as GPP, DSP, and FPGA. GPP is a microprocessor that is optimized for powerful computations, but consumes more power. It can be used in laboratories for research purpose. DSP is a microprocessor that consumes less power than GPP, but its development is more difficult than the GPP. It is used in most of the cellular terminals and base stations. FPGA is a microchip that can be configured by the user for a certain purpose, which makes it the best solution for implementing hardware blocks.

2.4 ZYNQ Board (ZC702)

2.4.1 FPGA Evolution History

The FPGA is an IC that is electrically programmed to execute a certain application. Initially it has no functionality to operate before it is programmed. FPGA is formed from a combination of transistors that are connected together in a specific way. Applying an external voltage on these transistors leads to operating a certain functionality. The combination of transistors is called Look Up Tables (LUTs) [23].

Each group of LUTs forms a Programmable Logic Block (PLB). Recent FPGAs have different types of PLBs that can operate as memory blocks to store data for internal operations. PLBs can also operate as multipliers to serve complex arithmetic operations. The FPGA internal routing consists of wires and programmable switches that allow connections among the PLBs, memory blocks, multipliers, and I/O ports. These connections are developed to achieve best data routing and latency. Also, there is a dedicated connection network that takes care of clock distribution and reset signals in order to achieve low skew.

The LUT size is measured by its number of inputs. The number of LUTs in the PLB can be equally sized or mixture of different sizes. There are three different techniques used to program the FPGA LUTs: Anti-Fuse, Flash, and SRAM programming technologies [24]. The advantage of the Anti-Fuse and Flash over the SRAM is being non-volatile and being able to occupy small area. However, SRAMs are easily re-programmed. They use the standard CMOS process technology which made them the first candidate to become the dominant approach to program FPGA LUTs.

Current FPGAs have IP blocks; these IPs are standard libraries which are optimized and developed to facilitate the development of the FPGA. Microprocessors are considered one of the important FPGA IP cores. There are two types of microprocessors, softcore and hardcore. Softcore processors such as Xilinx Micro Blaze are implemented using FPGA logic gates [25]. Hardcore processors such as IBM PowerPC are fabricated in the core of FPGA chip and connected to the fabric as shown in Figure 2.3.

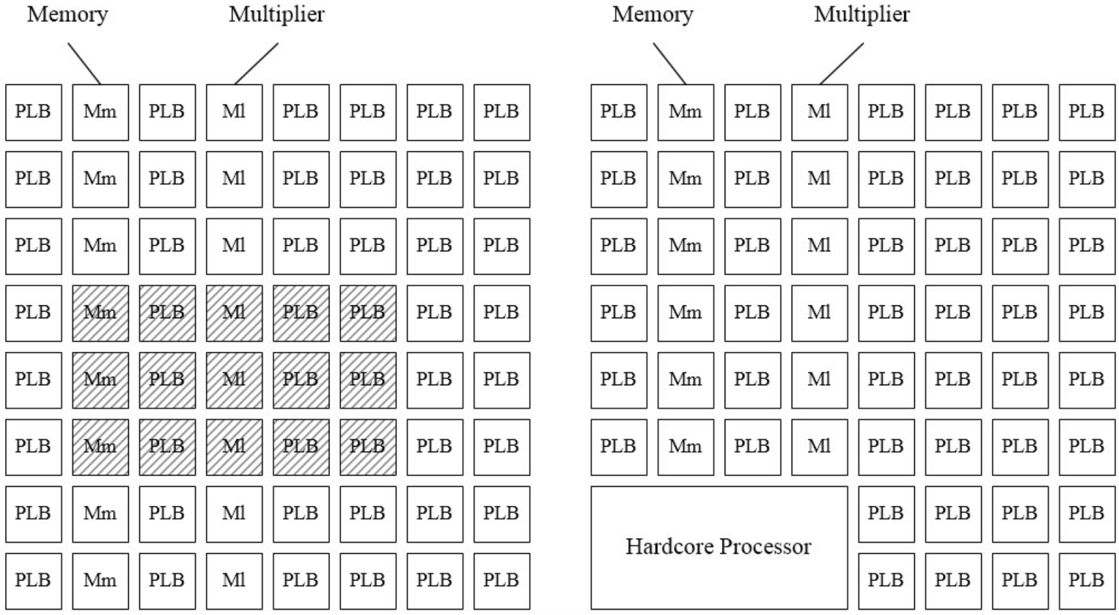


Figure 2.3: Softcore and hardcore processors [26]

Although softcore processors suffer from speed limitations (around 200 MHz), it is easy to customize their instructions. On the other hand, using hardcore processor helps in achieving higher processing speeds more than 1GHz. Zynq series offered by Xilinx is a perfect example of the current SoC chips, since it combines ARM dual-core or quad-core microprocessor placed in the Processing System (PS) part with Xilinx FPGA fabric that represents the Programmable Logic (PL) part [23].

2.4.2 Introduction to The Board

The ZC702 evaluation board for the XC7Z020 SoC as shown in Figure 2.4, provides a hardware environment for developing and evaluating designs targeting the Zynq device [27]. The ZC702 board provides common features to many embedded processing systems, including DDR3 memory component (used in this project to save the partial bit stream files and input test files for the communication systems), a tri-mode Ethernet PHY, a general purpose I/O, and two UART interfaces. The UART interfaces are not only used to signal the PS but also to display the options, data, and signals on the terminal. The PS integrates two ARM Cortex-A9 MP Core application processors, AMBA interconnect, internal memories, external memory interfaces, and peripherals including: USB, Ethernet, SPI, SD/SDIO, I2C, CAN, UART, and GPIO [27]. The PS runs independently of the PL and boots at power-up or reset. Figure 2.5 illustrates the Zynq ps7 internal structure.

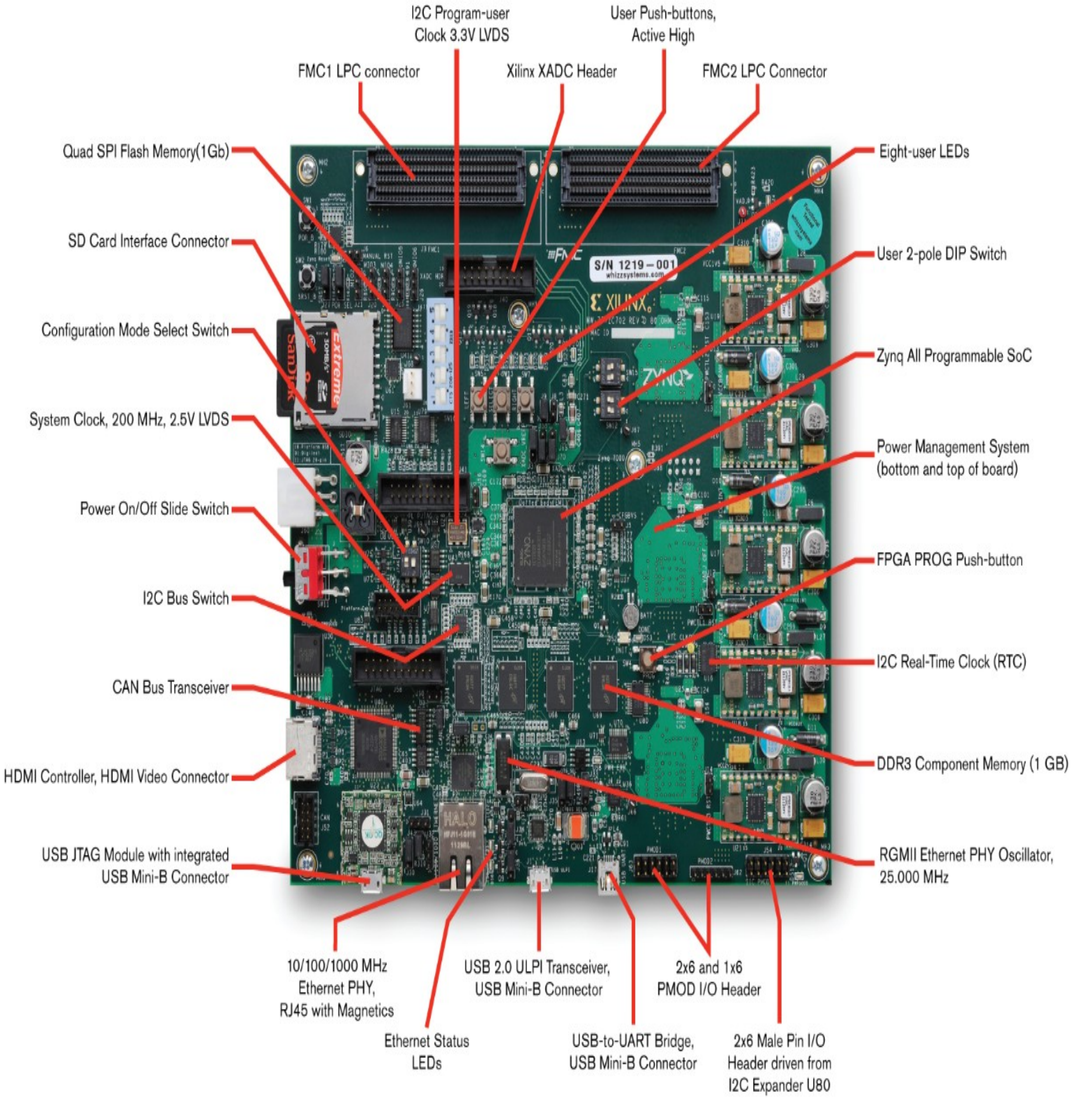

I2C Program-user Clock 3.3V LVDS
User Push-buttons, Active High
FMC1 LPC connector
Xilinx XADC Header
FMC2 LPC Connector
Quad SPI Flash Memory(1Gb)
SD Card Interface Connector
Configuration Mode Select Switch
System Clock, 200 MHz, 2.5V LVDS
Power On/Off Slide Switch
I2C Bus Switch
CAN Bus Transceiver
HDMI Controller, HDMI Video Connector
USB JTAG Module with integrated USB Mini-B Connector
Eight-user LEDs
User 2-pole DIP Switch
Zynq All Programmable SoC
Power Management System (bottom and top of board)
FPGA PROG Push-button
I2C Real-Time Clock (RTC)
DDR3 Component Memory (1 GB)
RGMII Ethernet PHY Oscillator, 25.000 MHz
10/100/1000 MHz Ethernet PHY, RJ45 with Magnetics
USB 2.0 ULPI Transceiver, USB Mini-B Connector
2x6 and 1x6 PMOD I/O Header
Ethernet Status LEDs
USB-to-UART Bridge, USB Mini-B Connector
2x6 Male Pin I/O Header driven from I2C Expander U80

Figure 2.4: Zynq board [23]

2.4.3 CLB Overview

The 7-series CLB provides advanced, high-performance FPGA logic:

- Real 7-input LUT technology.
- Dual LUT5 (5-input LUT).
- Distributed memory and shift registers logic capability.
- Dedicated high-speed carry logic for arithmetic functions.

CLBs are the main logic resources for implementing sequential as well as combinatorial circuits. Each CLB element is connected to a switch matrix as shown in Figure 2.6. Relation between row and column CLBs is illustrated in Figure 2.7. Each CLB element contains a pair of slices [28]. The LUTs in 7 series FPGAs can be configured either as 7-input LUT with one output, or as two 5-input LUTs with separate outputs.

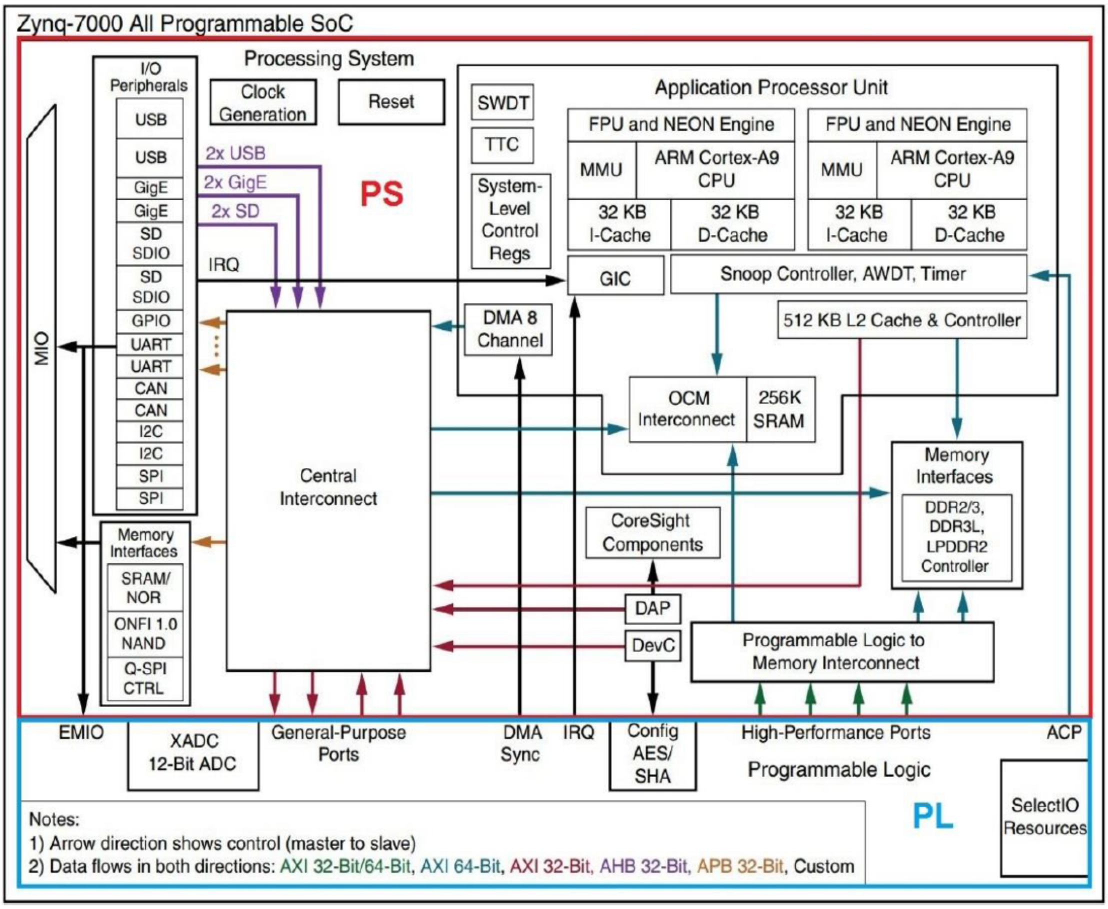


Figure 2.5: Zynq board internal structure [23]

Approximately two-thirds of the slices are SLICEL and the rest are SLICEM. LUTs in each slice can be used as distributed 74-bit RAM, 32-bit shift registers (SRL32), or two SRL17s. Modern synthesis tools take advantage of these highly efficient logic, arithmetic, and memory features. The Board's most important resources are listed in Table 2.1.

Table 2.1: FPGA Resources [23]

Resource	FPGA Capacity
LUT	53200
BRAM	140
DSP	220
I/O Pins	484

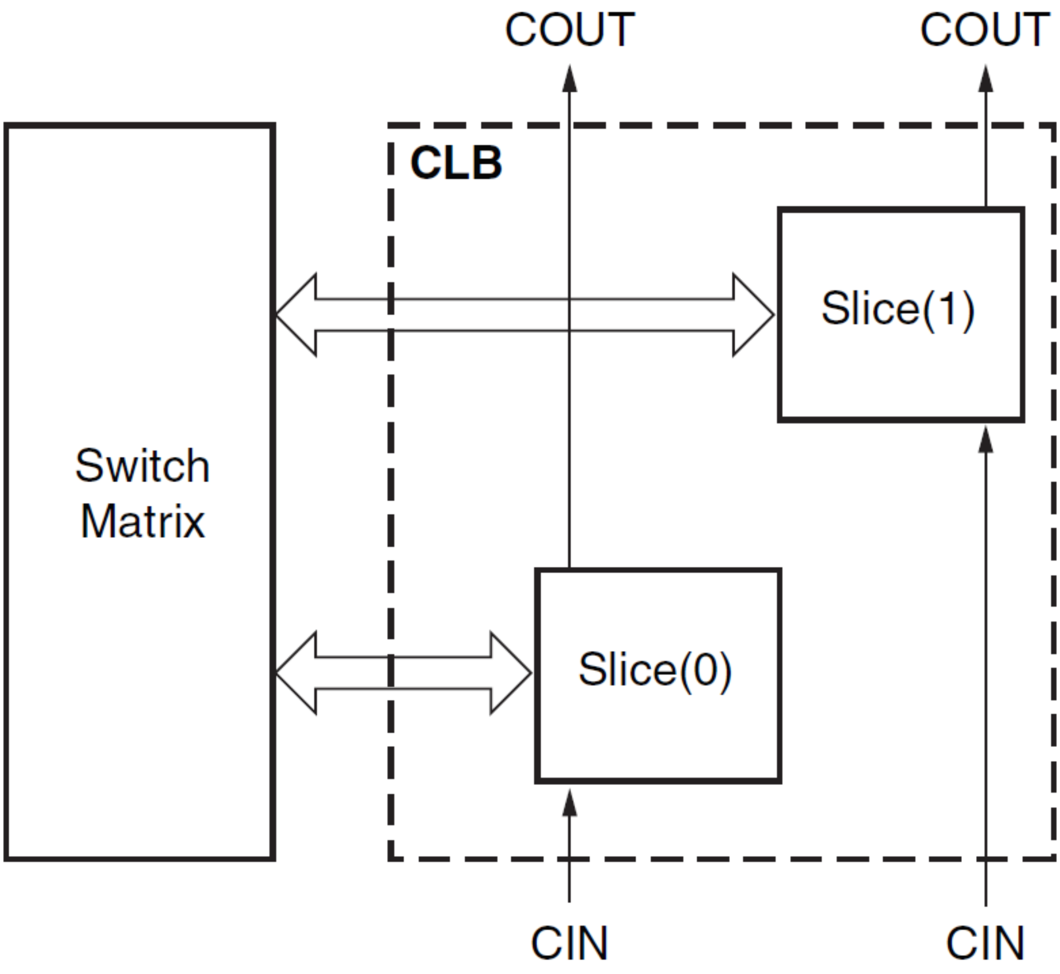


Figure 2.6: Arrangement of slices within the CLB [28]

2.5 FPGA Configuration

2.5.1 Configuration Definition

Configuration is a complete design programmed on the FPGA. FPGA can be viewed as a two-layered device: configuration memory layer and logic layer as shown in Figure 2.8. The configuration or the complete design stored on the configuration memory layer, will control the logic on the other layer.

2.5.2 Types of Configuration

There are three types of configuration for FPGAs:

1. **Fixed Configuration**: Data is loaded from a memory at power-on, then the configuration will remain fixed until the end of the FPGA cycle. This type lacks efficiency, since all possible functions needed to be done by the FPGA must be specified in the configuration file from the beginning.

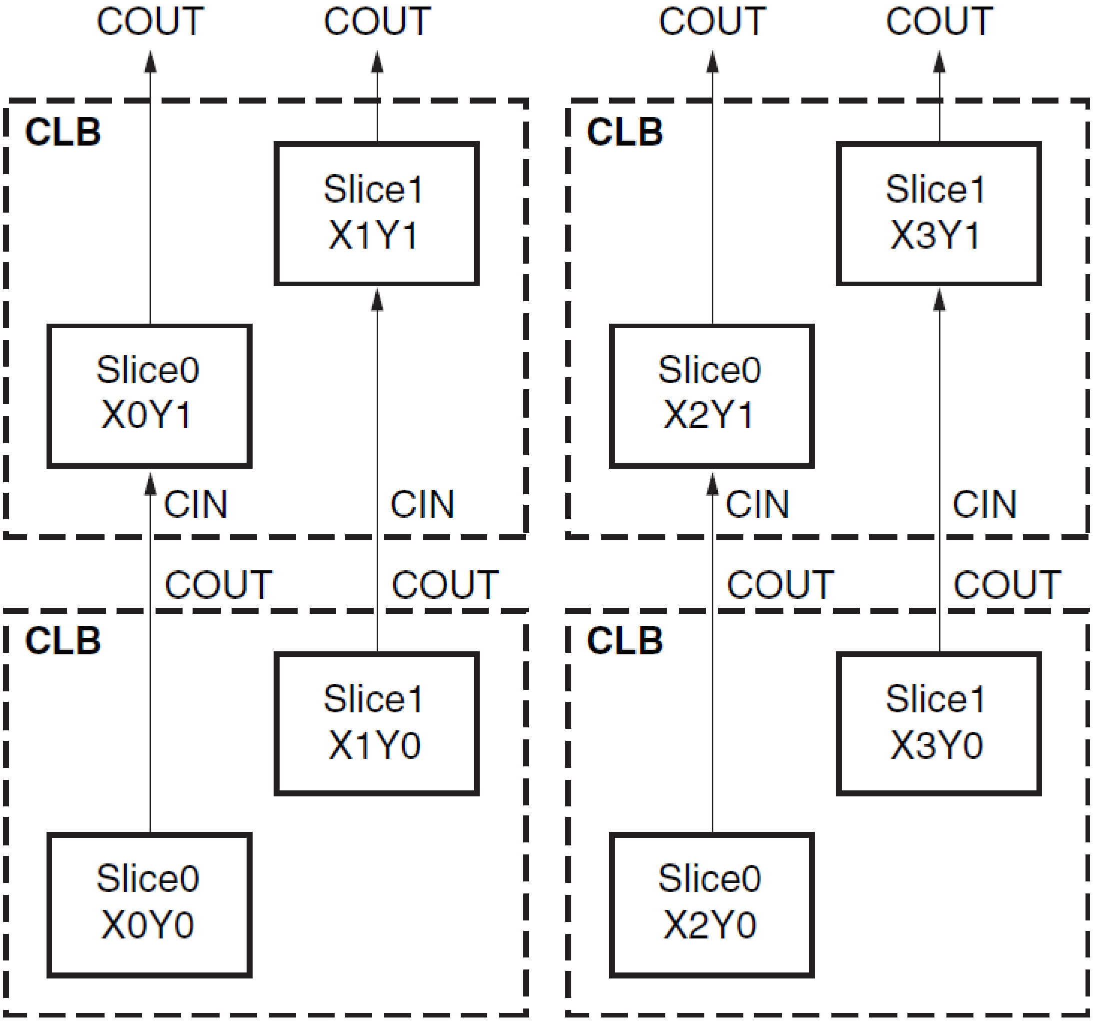


Figure 2.7: CLB column and row connections [28]

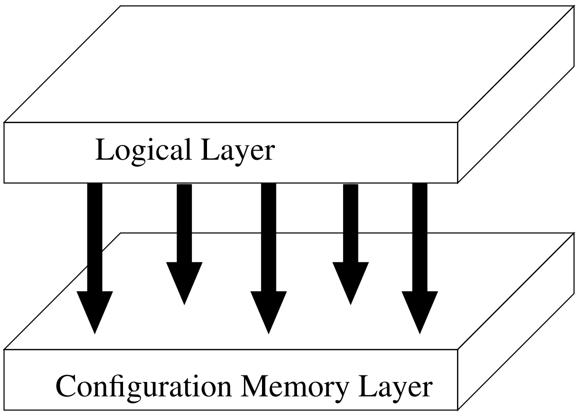


Figure 2.8: FPGA Layers

2. **Partial Reconfiguration**: Initial full bit file with a complete configuration is loaded into the device at power-on. Whenever something to be altered, all computations will stop, then a partial bit file that contains the modification in the original com-

plete design is loaded. The reconfiguration overhead time is reduced in such case compared to the previous type. There are some applications where FPGAs are used as communication hub, they must be active all the time to retain active links. In such cases, partial reconfiguration is not enough, as the computations stop during loading the partial bit file.

3. **Dynamic Partial Reconfiguration**: Unlike the partial reconfiguration, while the configuration layer on the FPGA is being modified, the logical layer continues its normal operation, except for the circuit subjected to the modification. The reconfiguration overhead is reduced in this type.

2.5.3 DPR in FPGAs

DPR technology, introduced by Xilinx, is a leading technology which allows run-time reconfiguration of a previously chosen partition in the design with partial bit stream files as show in Figure 2.9 [29]. The bit stream files are stored in a memory, and user is allowed to choose one of them to be loaded later into the reconfigurable partition using one of the different access ports (ICAP, PCAP, JTAG, . . . etc). The advantages of using DPR technique are:

1. **Resource Utilization Reduction**: Instead of using multiple resources for each standard implemented in the mobile phone, all implemented standards shall use the same resource.
2. **Power Consumption Reduction**: Since only one chain will be working at a time, this will save more power.

On the other hand, there are some challenges that are facing the DPR technology in implementing multi-standard SDR:

1. **Reconfiguration time**: The time taken to switch between different communication standards on the mobile device should be small as much as possible.
2. **Configuration Memory**: Fast memory access with large capacity is needed to cover the whole partial bit stream files needed for all the standards with their versions.

2.5.4 Types of Bit Files

There are two types of bit stream files used to configure the FPGA:

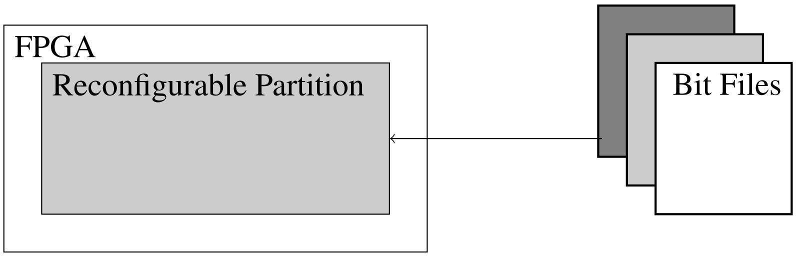


Figure 2.9: DPR configuration criteria [29]

1. **Full Bit File**: contains the data of a complete design/configuration. This includes all the necessary information to:
    (a) reset the FPGA.
    (b) configure it with a complete design.
    (c) verify that the bit file is not corrupted.
2. **Partial Bit File**: contains partial design configuration. It has no header, only the address of the target region and its corresponding partial data. Partial bit files may have many errors such as the address and data information. There is no error detection built-in mechanism. A corrupted partial bit file may damage the FPGA if left in operation. Therefore, systems that contain partial bit files with high probability of being corrupted, such as those which send data over radio channels, should implement a CRC circuit on the FPGA before loading the received bit file.

2.6 DPR Techniques

Xilinx offers two different DPR modes to transfer the bit stream files into the configuration memory: internal and external modes. Figure 2.10 shows the different techniques used in each mode.

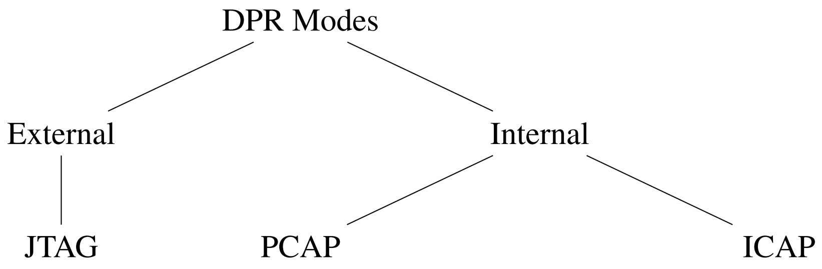


Figure 2.10: DPR controllers [30]

2.6.1 External Mode Using JTAG

The partial bit stream files are loaded to the configuration memory through an external source such as JTAG cable. However, this is not recommended in implementing the SDR, as it gives relatively low reconfiguration time as shown in Table 2.2. The maximum theoretical BW that the JTAG cable can give is 66 Mbps, in addition to the time overhead taken to transfer the data from the source to the configuration memory.

2.6.2 Internal Mode

Reconfiguration in such case takes place through an already implemented access port to the configuration memory such as Processor Configuration Access Port (PCAP) in PS side or as Internal Configuration Access Port (ICAP) in the PL side.

2.6.2.1 PCAP on PS Side

Processor Configuration Access Port is an access port for FPGA configuration memory which is controlled by the processor to perform the configuration process. Although the maximum theoretical BW using PCAP is 400 MB/s as shown in Table 2.2, the actual transfer rate is approximately 145 MB/s as the overall throughput is limited by the PS AXI interconnect [31, 32].

2.6.2.2 ICAP on PL Side

Internal Configuration Access Port is an access port at the PL side used with a controller to perform dynamic reconfiguration process. The maximum theoretical throughput of the ICAP is 400 MB/s as shown in Table 2.2. The type of controller used with the ICAP determines the achievable actual throughput. Increasing the throughput requires a complex controller with high resource utilization.

Table 2.2: Configuration Tools Bandwidth [23]

Configuration Tool	Type	Max Frequency	Bus Width	Max Bandwidth
ICAP	Internal	100 MHz	32-bit	400 MB/s
PCAP	Internal	100 MHz	32-bit	400 MB/s
JTAG	External	66 MHz	1-bit	8.25 MB/s

Xilinx offers two IP controllers for reconfiguration. This is done by passing the bit stream files to the IP through a software code running on the processor which handles the reconfiguration data and control signals. The two controllers are:

1. **AXI-HWICAP**:
   This is a simple IP controller composed of an asynchronous read and write FIFOs, control registers, and FSM associated with the ICAP used for reconfiguration as shown in Figure 2.11. The IP core interacts with the processor through AXI4-Lite interface. A full description of how the core works is mentioned in [33].

2. **PRC**:
   Partial Reconfiguration Controller is a more complex IP than AXI-HWICAP which depends on the concept of Virtual Sockets (VS) [34]. The PRC controlles the access of partial bit files using the ICAP as shown in Figure 2.12. The VS represents the Reconfigurable Partition (RP) associated with some logic blocks used to isolate it from the static region during reconfiguration process. This results in a better throughput as illustrated in Figure 2.13. A Fetch Path as shown in Figure 2.14 is used to transfer configuration bits from the processor to the ICAP allocated with the VSs. The number of VSs represents the number of RPs in the design.

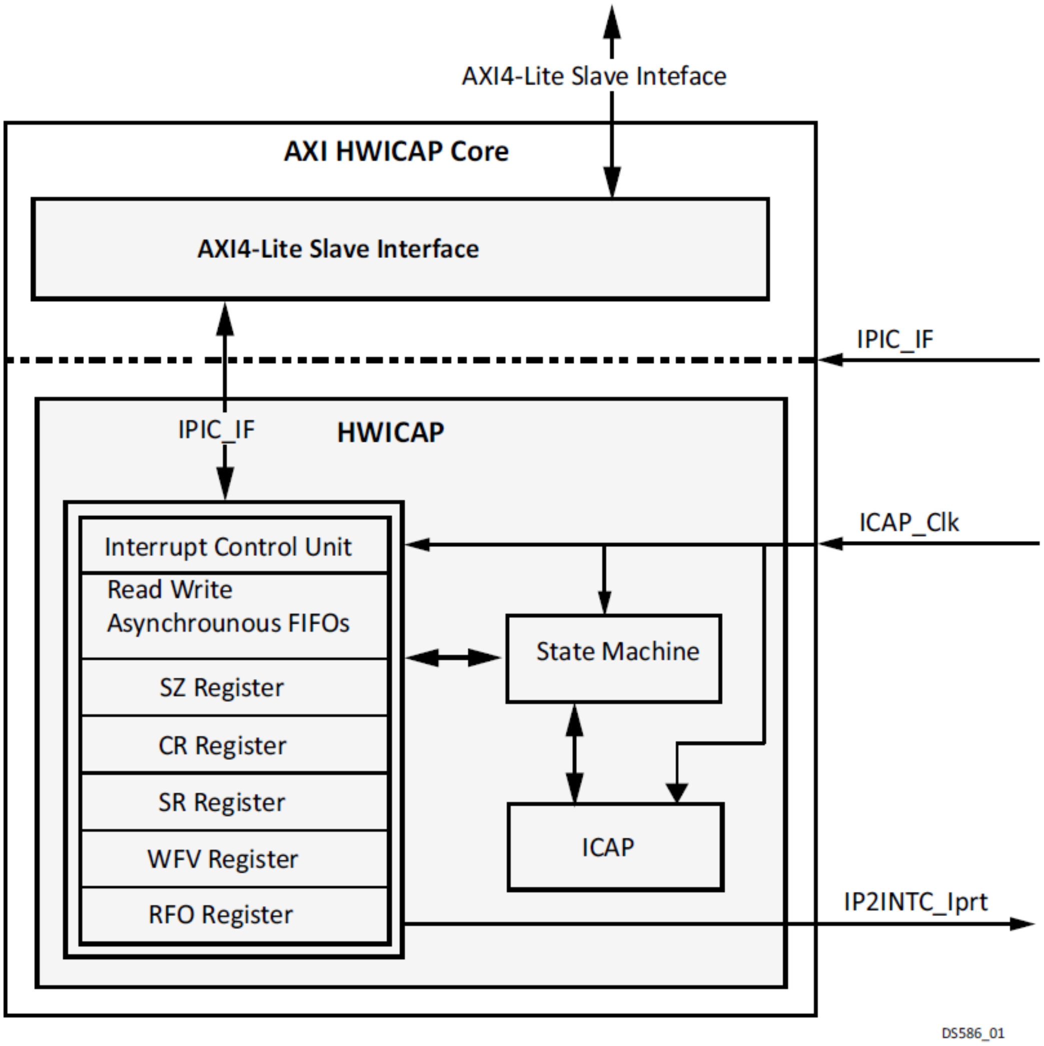


Figure 2.11: AXI HWICAP core [33]

2.7 Summary

The following chapter introduces some background about the history of the FPGAs and their evolution. A list of types of FPGA configuration is provided including the DPR technique. Finally, the chapter illustrates the DPR controllers and shows the advantage of using each one of them.

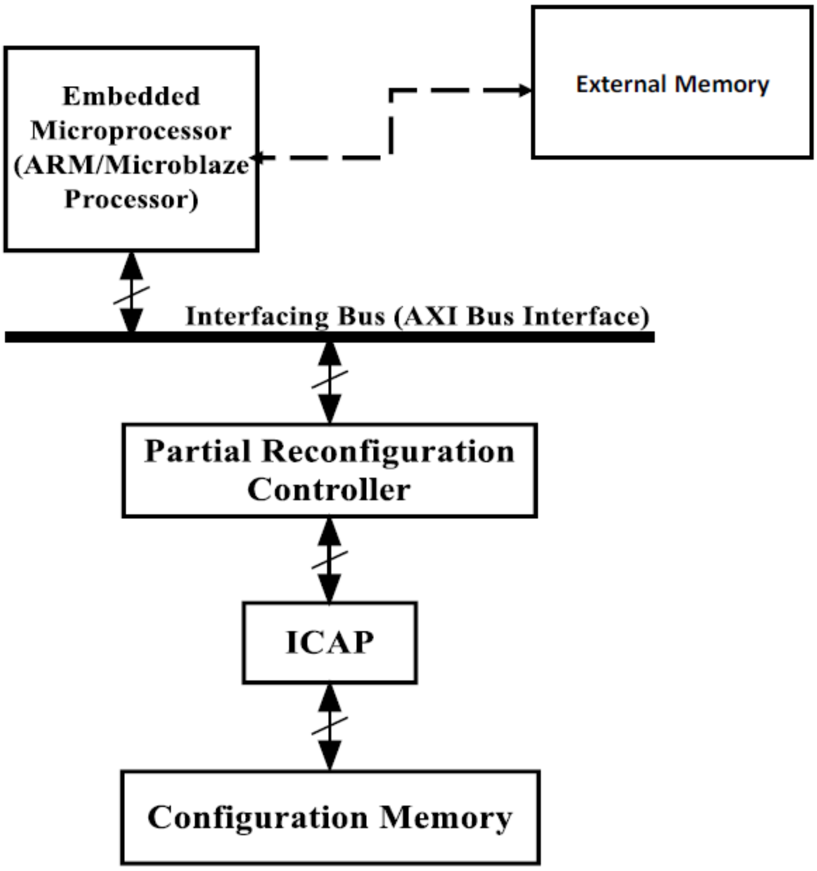

Embedded Microprocessor (ARM/Microblaze Processor)
External Memory
Interfacing Bus (AXI Bus Interface)
Partial Reconfiguration Controller
ICAP
Configuration Memory

Figure 2.12: PRC in DPR system [34]

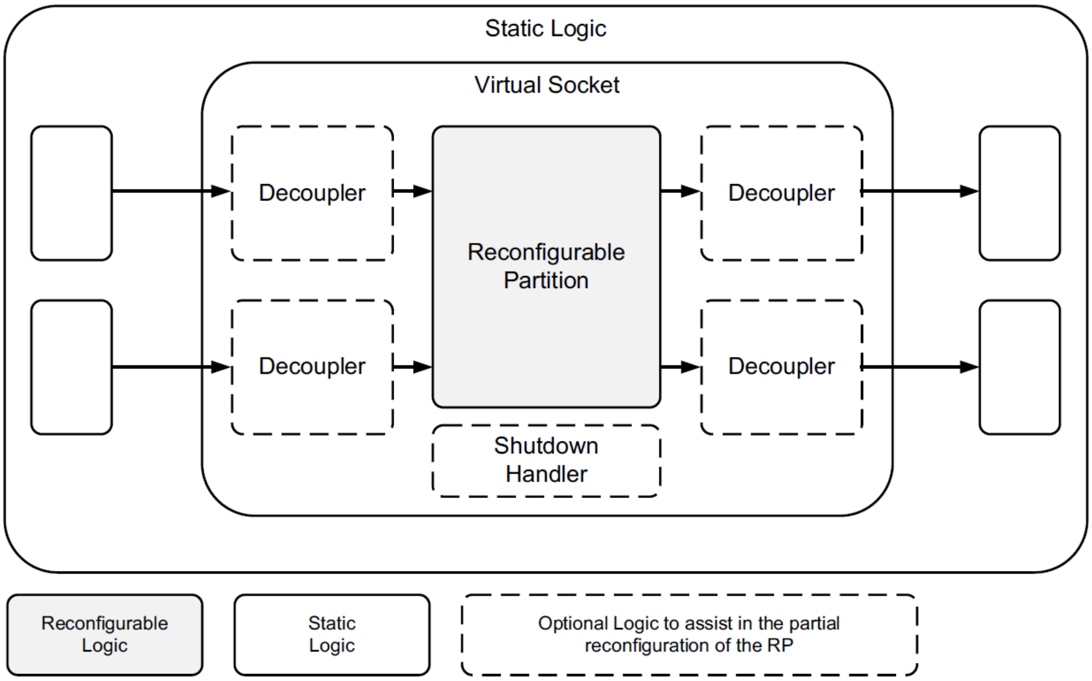


Figure 2.13: PRC virtual socket [34]

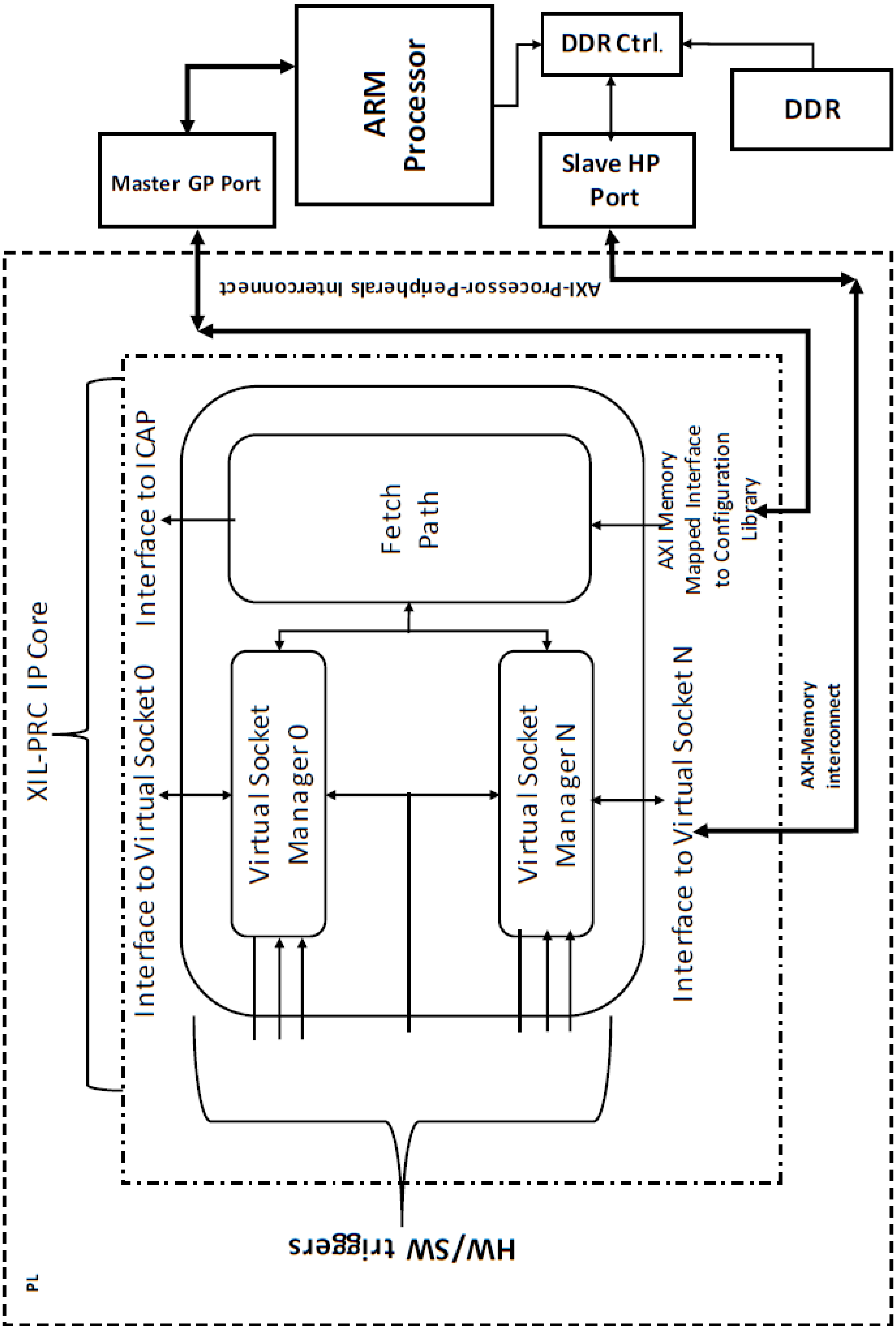


Figure 2.14: PRC fetch path [34]

Chapter 3: SDR Transmitter Design

3.1 SDR System Overview

The Wi-Fi chain as illustrated in [36] has many Modulation Coding Schemes (MCS). The implemented chain contains all the combinations starting from MCS1 to MCS6. Table 3.1 shows the difference between all implemented modulation schemes as mentioned in [36]. The 3G chain also has variations in the types of the used CRC blocks (CRC8, CRC12, CRC16, and CRC24). Only single type of modulation scheme is implemented for Bluetooth, 2G, and LTE.

Table 3.1: Wi-Fi Various MCS [36]

MCS Number	Puncturing	Interleaver	Mapper
MCS1	None	$N_{CBPS} = 48$	BPSK
MCS2	$r = 3/4$	$N_{CBPS} = 48$	BPSK
MCS3	None	$N_{CBPS} = 96$	QPSK
MCS4	$r = 3/4$	$N_{CBPS} = 96$	QPSK
MCS5	None	$N_{CBPS} = 192$	16-QAM
MCS6	$r = 3/4$	$N_{CBPS} = 192$	16-QAM

Each block in every chain has three input and three output signals. Input signals are: *data in*, *valid in*, and *enable*. Output signals are: *data out*, *valid out*, and *finished*. *Valid out* signal is set, when output is ready. It is connected to the *valid in* signal of the successive block. *Finished* and *enable* signals are used to synchronize between blocks easily. *Finished* signal is a feedback signal connected to the enable of the preceding block, which is set when the block is ready to receive data from the preceding one as shown in Figure 3.1.

Each chain is working on multiple clock domains in order to match the input rate with the desired output rate. Dual clock RAMs are used to overcome the Clock Domain Crossing (CDC) issues leading to metastability. However, matching all clocks in each chain to avoid jitter is another challenge. A customized clock distribution network is integrated in each chain as an RTL design, to overcome this challenge. The network accepts single clock source from the ARM processor and then generates identical clones for each system.

Parameterization technique is used to design various modulation schemes of Wi-Fi chain. Since all modulation schemes differ only in the type of puncturing, interleaving, or mapping; all blocks are implemented under the same source directory, and switching between them is performed through using simple parameters. The same criteria is used in 3G chain to switch between different CRC blocks.

SDR basic idea is having multiple communication standards sharing the same hardware that is controlled by the software. In order to make it feasible to deploy this idea, all

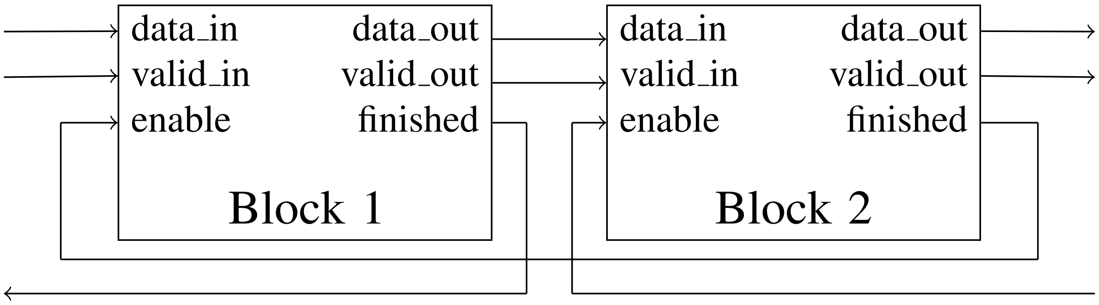


Figure 3.1: Block signals in each chain

corresponding blocks in each chain should almost occupy the same area and consume the same power [37, 38, 39]. The following area and power optimization techniques are deployed:

1. **Finite State Machine (FSM) extraction**: In order to achieve the best allocation for complex FSMs, the synthesizer must recognize them. This is done by writing the RTL code using Xilinx FSM template.

2. **Simplifying math operations**: Multiplications and divisions with constant numbers can be converted into shift operations to synthesize in a small number of LUTs instead of wasting DSPs on simple operations.

3. **Using DSP Primitives**: Implementing complex mathematical operations using DSPs is performed through either using the RTL attributes or instantiating the DSP primitive explicitly.

4. **Synthesizing in BRAMs**: Most of the chain blocks contain memories to store the symbol values. Synthesizing in BRAMs instead of LUTs is performed using RTL attributes mentioned in [40].

5. **Implementing customized DFT**: Since the DFT IP offered by Xilinx is 24-point and the required is only 14-points, its power consumption is high. Solving this issue is done by implementing a customized 14-point DFT to save power and area.

6. **Synthesis options**: Specific synthesis options are chosen in order to achieve high area optimization.

3.2 Bluetooth Transmitter

Figure 3.2 shows the implemented blocks in the Bluetooth chain according to [35].

3.2.1 Segmentation

Unlike any other chain, the Bluetooth transmitter chain [35] starts with a segmentation block that separates the header bits from the payload in such way to achieve a guard time of 5μs after baseband processing. The algorithm works on the flow shown in Figure 3.3. The chain is then divided into two sub-chains, one for the header and the other for the payload. Figure 3.2 shows that the header and payload share some blocks.

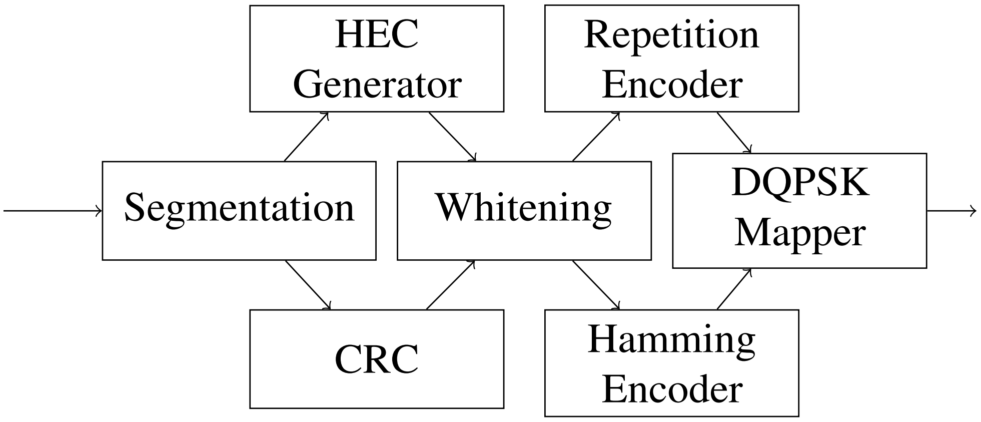


Figure 3.2: Bluetooth transmitter block diagram [35]

3.2.2 HEC Generator, CRC, and Whitening

The Header Error Checking (HEC) generator is initialized with the least 8 bits of the Upper Address Part (UAP). It is used for adding extra 8 bits to the header for error checking at the receiver side. The CRC used in the payload chain does the same functionality by adding 16 bits to the end of the payload for error checking. As shown in Figure 3.2, whitening is shared between both header and payload. It is used to randomize the data to get rid of highly redundant “1”s and “0”s patterns. The three blocks are implemented using the generator polynomials shown below receptively [35].

$$G1(D) = D^8 + D^7 + D^5 + D^2 + D + 1 \tag{3.1}$$

$$G2(D) = D^{16} + D^{12} + D^5 + 1 \tag{3.2}$$

$$G3(D) = D^7 + D^4 + 1 \tag{3.3}$$

3.2.3 Repetition and Hamming Encoders

Since it is expected that the transmitter and the receiver will be in the same Line of Sight (LoS), the transmitted packets will be less error prone to channel effects compared to the rest of standards. The Bluetooth transceiver replaces the convolution encoders with block encoders for simplicity. Header bits are encoded using repetition encoder with coding rate equals to 1/3. The payload is encoded using shortened Hamming code with 2/3 coding rate.

Differential encoding is then applied to both header and payload followed by ordinary QPSK mapper. The equation shown below emphasizes the differential encoding and mapping operations [35].

$$S_K = S_{K-1} \times e^{j\phi} \tag{3.4}$$

The variable S_K represents the mapped symbols. The relationship between the binary input bits b_k and the angle ϕ_k is listed Table 3.2.

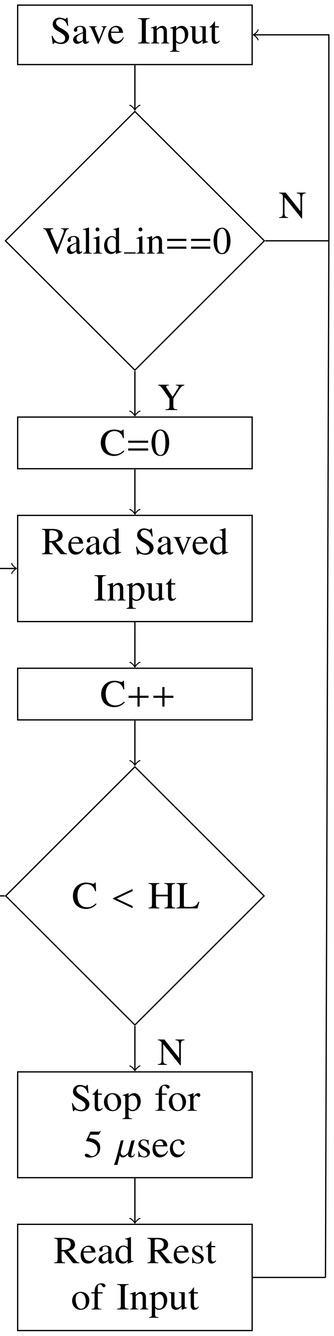


Figure 3.3: Segmentation controller algorithm

Table 3.2: DQPSK Mapping Relationship [35]

b_{2k-1}	b_{2k}	ϕ_k
0	0	$\pi/4$
0	1	$3\pi/4$
1	1	$-3\pi/4$
1	0	$-\pi/4$

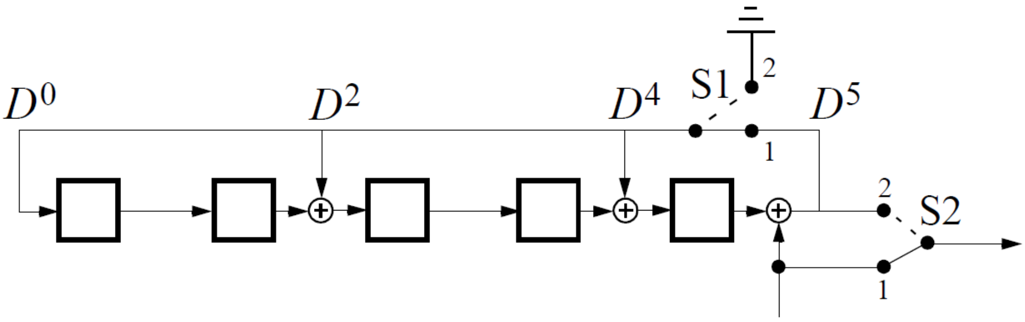


Figure 3.4: Bluetooth hamming encoder [35]

3.2.4 Chain Utilization

Table 3.3 lists the utilized area for all Bluetooth transmitter chain blocks in terms of LUTs, BRAMs, and DSPs.

Table 3.3: Bluetooth Transmitter Chain Area Utilization

Chain Block	LUTs	BRAMs	DSPs
Segmentation	104	0.5	0
HEC Generator	128	0	0
CRC	150	0.5	0
Whitening	7	0	0
Repetition Encoder	213	1	0
Hamming Encoder	43	0	0
Mapper	126	1	0

3.3 Wi-Fi Transmitter

Figure 3.5 shows the implemented blocks in the Wi-Fi chain according to [36].

3.3.1 Scrambler

Scrambler is responsible for randomizing the MAC layer data in order to prevent the presence of long “1”s or “0”s sequences. This is useful in synchronization between the transmitter and the receiver. The generator polynomial shown below is implemented using logic gates and shift registers [36].

$$S(x) = x^7 + x^4 + 1 \tag{3.5}$$

3.3.2 Convolutional Encoder

Convolutional encoder is used to encode the scrambled bits with coding rate equals to 1/2. Encoding is responsible for data replication in order to decrease the bit error rate,

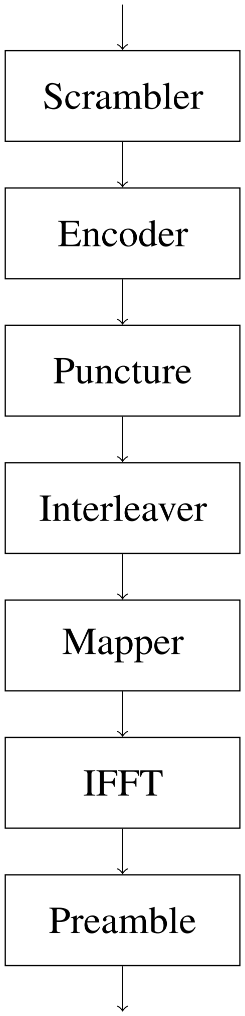


Figure 3.5: Wi-Fi transmitter chain block diagram [36]

and enable the decoder to deduce the correct transmitted bits. The generator polynomial shown in Figure 3.6 is also implemented using logic gates and shift registers. Parallel to serial block operating on double the frequency is used after the encoder in order to feed the puncture with serial bits.

3.3.3 Puncture

Though the usefulness of the encoding algorithm, it increases the number of bits leading to reducing the bit rate. Puncturing technique is used to solve this issue by stealing specific encoded bits from the transmitted data, and then inserting them as dummy zeros at the receiver side. Figure 3.7 shows the position of stolen bits for coding rate equals to 3/4 [36]. A BRAM whose address is controlled by special logic is used in implementing the puncture.

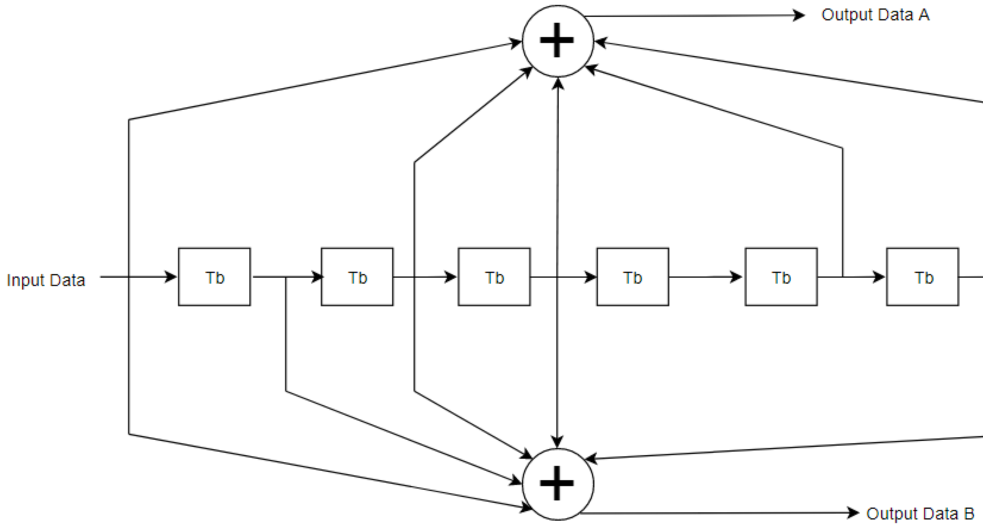


Figure 3.6: Wi-Fi convolutional encoder [36]

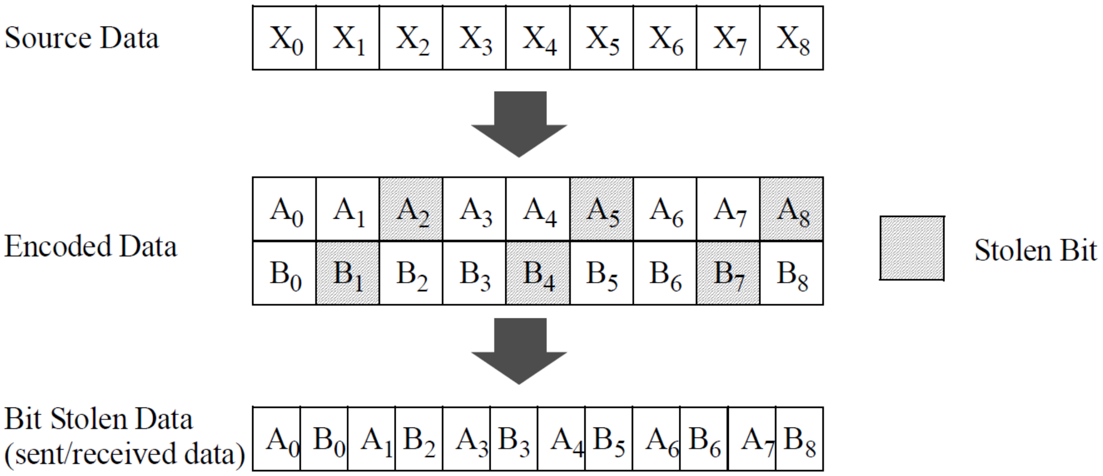

Source Data
X0 X1 X2 X3 X4 X5 X6 X7 X8
Encoded Data
A0 A1 A2 A3 A4 A5 A6 A7 A8
B0 B1 B2 B3 B4 B5 B6 B7 B8
Stolen Bit
Bit Stolen Data
(sent/received data)
A0 B0 A1 B2 A3 B3 A4 B5 A6 B6 A7 B8

Figure 3.7: Wi-Fi puncture algorithm [36]

3.3.4 Interleaver

Interleaver is used to get rid of burst errors by re-arranging the punctured bits. Interleaving is performed on two permutation steps. Three variables are used in the equations:

1. ”k”: represents the index of the original received bits before the first permutation.
2. ”i”: represents the index after the first permutation and before the second one.
3. ”j”: represents the index after the second permutation before the mapper.

The first permutation is defined by the following equation [36]:

$$i = (k\%16) \times (\frac{N_{CBPS}}{16}) + \left\lfloor \frac{k}{16} \right\rfloor \text{ for } k = 0, 1, \ldots, N_{CBPS} - 1 \tag{3.6}$$

The second permutation is defined by the following equation [36]:

$$j = s \times \left\lfloor \frac{i}{s} \right\rfloor + (i + N_{CBPS} - \left\lfloor \frac{16i}{N_{CBPS}} \right\rfloor)\% s \text{ for } i = 0, 1, \ldots, N_{CBPS} - 1 \tag{3.7}$$

The variable "s" is dependent on the number of bits per sub-carrier [36]:

$$s = max(\frac{N_{BPSC}}{2}, 1) \tag{3.8}$$

Implementation of the two equations is performed using BRAMs and DSPs to calculate the memory addresses.

3.3.5 OFDM Section

The mapper converts the data from bit domain to symbol domain to be modulated. Since Wi-Fi is Orthogonal Frequency Division Multiplexing (OFDM) based, symbols modulation is done on several sub-carriers instead of single carrier. IFFT block is used to modulate the symbols. Implementation of this block is done using: IFFT core, controller, and two RAMs to store real and imaginary symbols. The Zynq IFFT IP [41] is used as the core as shown in Figure 3.8. Preamble is used after the IFFT block to add the long and short header symbols that enable synchronization between the transmitter and the receiver.

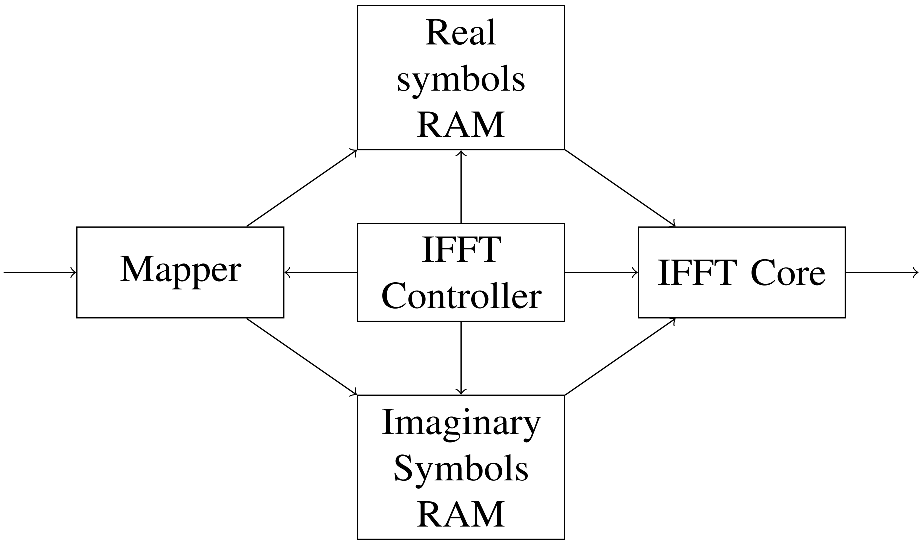


Figure 3.8: Wi-Fi IFFT block diagram

The implemented modulation schemes are BPSK and QPSK. The sinusoidal wave has three features: phase, frequency and amplitude. According to the given information and to the used modulation technique, bits are mapped to complex valued modulation symbol as shown in the following equation [36]:

$$d = (I + jQ) \tag{3.9}$$

The variables I and Q represent the real and imaginary parts. Tables 3.4 and 3.5 show the symbol mapping values in BPSK and QPSK modulation schemes respectively.

Table 3.4: BPSK Modulation Scheme [36]

Bit values	I	Q
0	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$
1	$-\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$

Table 3.5: QPSK Modulation Scheme [36]

Bit values	I	Q
00	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$
01	$\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$
10	$-\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$
11	$-\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$

3.3.6 Chain Utilization

Table 3.6 lists the utilized area for all Wi-Fi transmitter chain blocks in terms of LUTs, BRAMs, and DSPs.

Table 3.6: Wi-Fi Transmitter Chain Area Utilization

Chain Block	LUTs	BRAMs	DSPs
Scrambler	137	0	0
Encoder	75	0	0
Interleaver	191	2	1
Mapper	88	0.5	0
IFFT	1886	0.5	6
Preamble	151	0	0

3.4 2G Transmitter Chain

Figure 3.9 shows the implemented blocks in the GSM chain according to [42, 43, 44].

3.4.1 CRC

Cyclic Redundancy Check (CRC) block is used to add several bits to the MAC data. Checking is done at the receiver side on the added bits for error detection. Three CRC bits

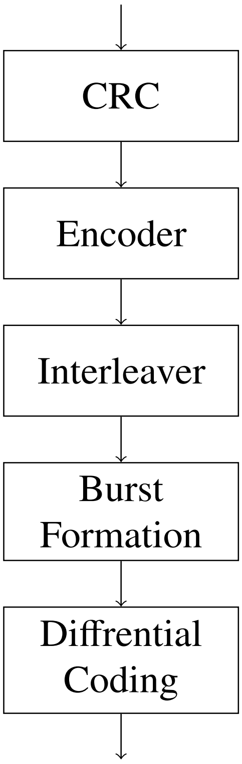


Figure 3.9: 2G transmitter chain block diagram [42]

are added to each 50 input bits. The generator polynomial shown below is implemented using logic gates and shift registers [42].

$$g(D) = D^2 + D + 1 \tag{3.10}$$

The first 182 bits plus the CRC bits are re-ordered according to the following equation [42]:

$$u(k) = \begin{cases} d(2k) & \text{for } k = 0, 1, ..., 90 \\ p(k) & \text{for } k = 91, 92, 93 \\ d(2k+1) & \text{for } k = 94, 95, ..., 184 \\ 0 & \text{for } k = 185, 186, 187, 188 \end{cases} \tag{3.11}$$

The variables $d(k), u(k), and p(k)$ represent the input bits, the output bits, and the CRC bits respectively.

3.4.2 Convolutional Encoder

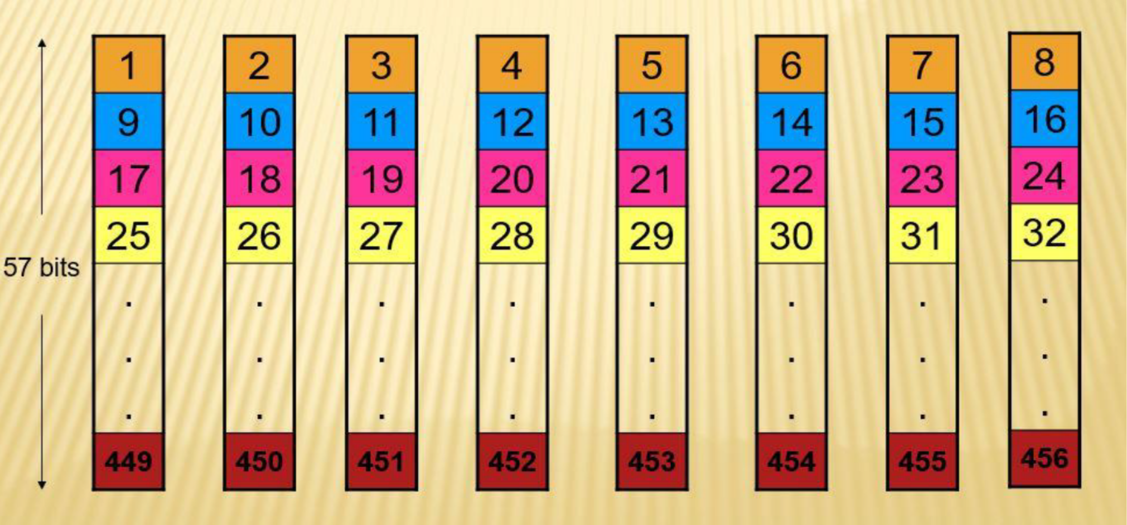
Convolutional encoder with coding rate equals to 1/2 is used. Encoding is done only on the first 189 bits. However, the remaining 78 bits are transferred directly to the interleaver. The implemented generator polynomials are shown below [42]:

$$G_0 = 1 + D^3 + D^4 \tag{3.12}$$

$$G_1 = 1 + D + D^3 + D^4 \tag{3.13}$$

3.4.3 Interleaver

Interleaving is performed through using 8 × 57 matrix, where data is stored row by row, then read column by column as shown in Figure 3.10. Implementation is done through using BRAM whose address is controlled with special logic.



1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
449	450	451	452	453	454	455	456

Figure 3.10: 2G interleaver matrix [42]

3.4.4 Burst Formation

Burst Formation block adds the burst bits used in Time Division Multiplexing (TDM). As shown in Figure 3.11, the burst bits added to the interleaved data have three types:

1. Tail bits: 3 bits added on both sides of the data for synchronization.
2. Training bits: 26 bits added in the middle of the frame to be used as pilots.
3. Steal Flag bits: 1 bit added on both sides to determine the channel type at the receiver side.

Tail	User Data	SF	Training	SF	User Data	Tail

Figure 3.11: Burst data formation [42]

The channel type is dependent on the value of the steal flag. If the steal flag equals to "1", the channel type is Fast Associated Control Channel (FACCH). In such case, the output of the burst formation is specific bit stream defined by the MAC. Otherwise, the output is the old data value stored in the memory. The memory read enable and output are controlled by the controller as shown in Figure 3.12. Figure 3.13 shows the controller algorithm.

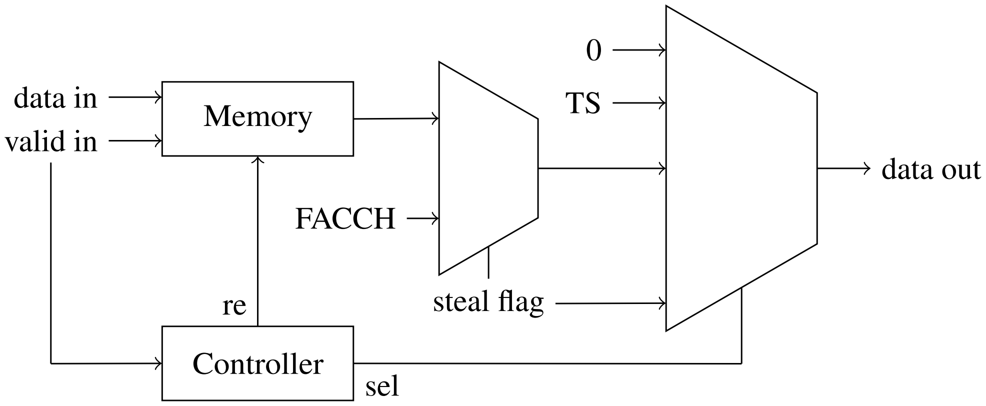


Figure 3.12: Burst formation block diagram

3.4.5 Differential Coding

Differential coding block encodes the data differentially in order to prepare it for GMSK modulation. Differential encoding is done through the following equation [44]:

$$d_i = d_i \oplus d_{i-1},\ d_i \in \{0, 1\} \tag{3.14}$$

Data is then mapped according to the following equation:

$$\alpha_i = 1 - d_i \tag{3.15}$$

The variable α_i represents the output of the differential coding block.

3.4.6 Chain Utilization

Table 3.7 lists the utilized area for all GSM transmitter chain blocks in terms of LUTs, BRAMs, and DSPs.

Table 3.7: 2G Transmitter Chain Area Utilization

Chain Block	LUTs	BRAMs	DSPs
CRC	200	1	0
Encoder	85	0	0
Interleaver	161	0.5	0
Burst Formation	114	0.5	0
Diffrential Coding	2	0	0

3.5 3G Transmitter Chain

Figure 3.14 shows the implemented blocks in the UMTS chain according to [45, 46, 47].

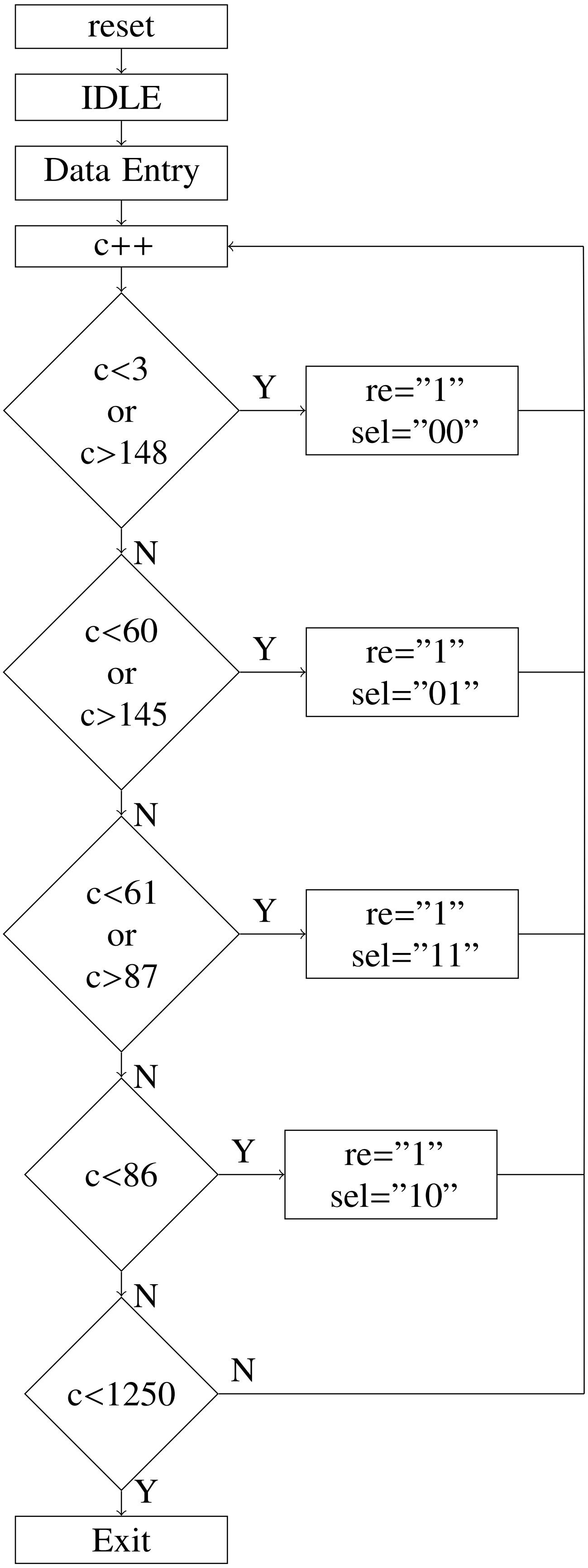


Figure 3.13: Burst formation controller algorithm

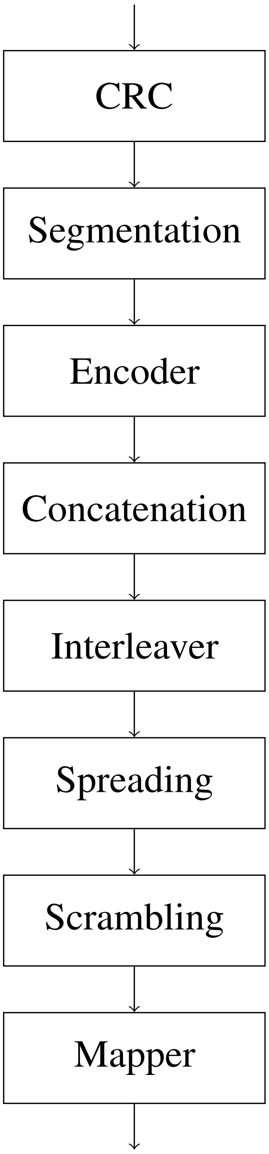


Figure 3.14: 3G transmitter chain block diagram [45]

3.5.1 CRC

Four combinations of CRC block are used in the 3G transmitter. Table 3.8 shows the generator polynomial of each type. Implementation is similar to the CRC used in the 2G chain.

Table 3.8: 3G CRC Polynomial Equations [46]

CRC Mode	Polynomial Equation
CRC24	$g_{crc24}(D) = D^{24} + D^{23} + D^{6} + D5 + D + 1$
CRC16	$g_{crc16}(D) = D^{16} + D^{12} + D^{5} + 1$
CRC12	$g_{crc12}(D) = D^{12} + D^{11} + D^{3} + D^{2} + D + 1$
CRC8	$g_{crc8}(D) = D^{8} + D^{7} + D^{4} + D^{3} + D + 1$

3.5.2 Segmentation

Segmentation is used to slice the bit stream into a set of blocks with certain block size defined by the MAC layer. Slicing is performed in order to let the encoder work properly. Implementation is done using FSM whose state diagram is shown in Figure 3.15.

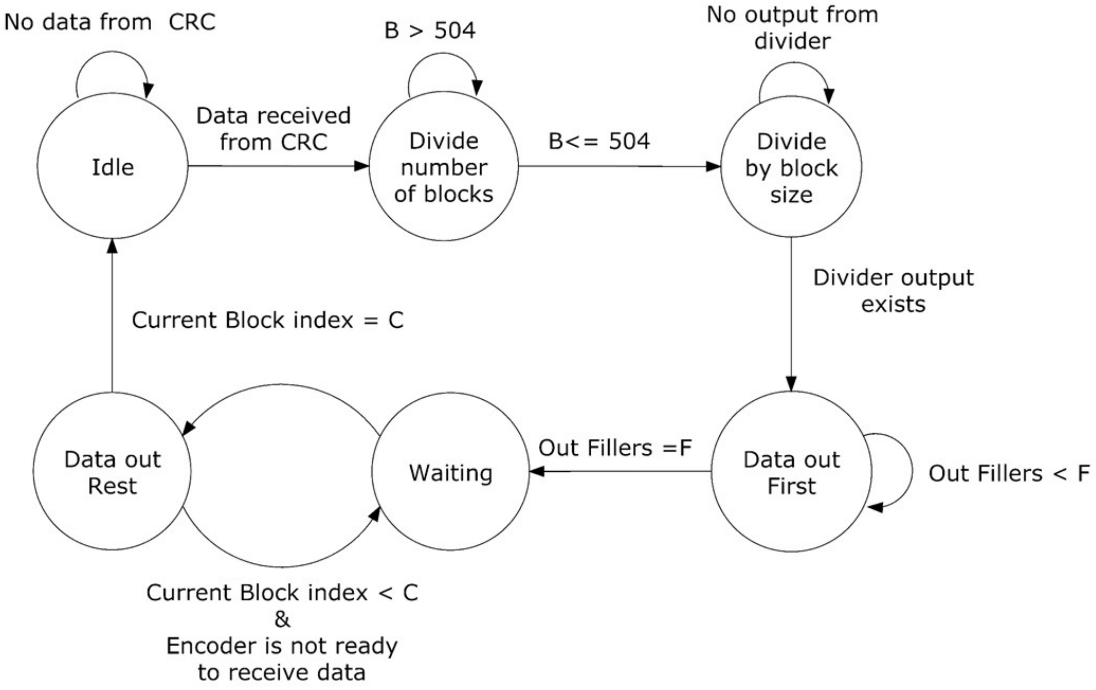


Figure 3.15: 3G segmentation FSM

3.5.3 Convolutional Encoder

Convolutional encoder with coding rate equals to 1/2 is used to add redundant bits. The encoder shown in Figure 3.16 is 9-bits length including the input bit. Implementation is similar to the Wi-Fi encoder. Parallel to serial block is added after the encoder as well.

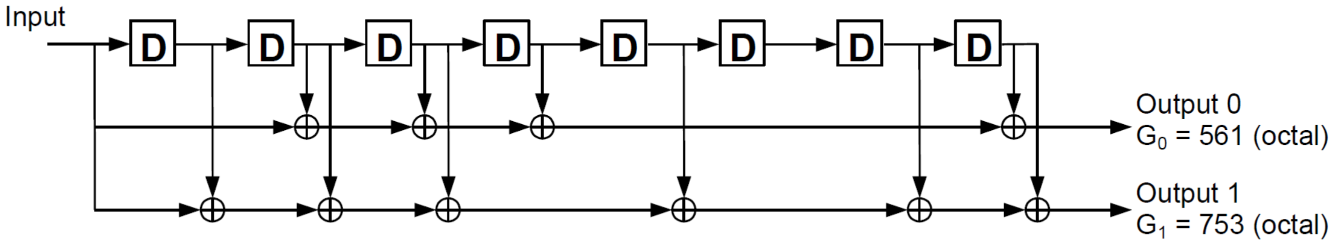


Figure 3.16: 3G convolutional encoder [46]

3.5.4 Code Block Concatenation

The encoded data is then concatenated using the Code Block Concatenation (CBC) to enter the first interleaver. Size of the block is constant number defined by the MAC layer.

3.5.5 Interleaver

Data interleaving is performed using four major blocks as shown in Figure 3.17:

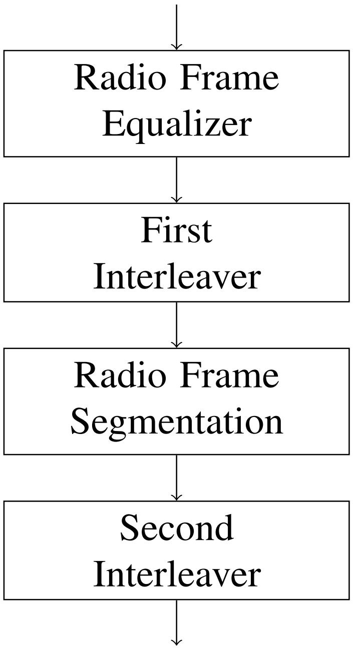


Figure 3.17: 3G interleaver block diagram [46]

1. Radio Frame Equalizer:
   Divides the input data into equally sized blocks and pads extra bits to each block. The relation between input bits $e(k)$ and output bits $t(k)$ is given below [46]:

$$t(k) = \begin{cases} e(k) & \text{for } k = 0, 1, \ldots, E \\ 0 & \text{for } k = E + 1, \ldots, F \times \left\lceil \frac{E}{F} \right\rceil \end{cases} \tag{3.16}$$

The variables E and F represent the number of input bits and the number of segments respectively. The number of segments is dependent on the interleaving period as shown in Table 3.9.

Table 3.9: Number of Segments [46]

Interleaving Period	Number of Segments
10 ms	1
20 ms	2
40 ms	4
80 ms	8

2. First Interleaver:
   This is inter-frame interleaver where all frames are interleaved together. Data is written row by row in a RAM, then read column by column with a certain order depending on values set by the MAC layer as shown in Figure 3.18.

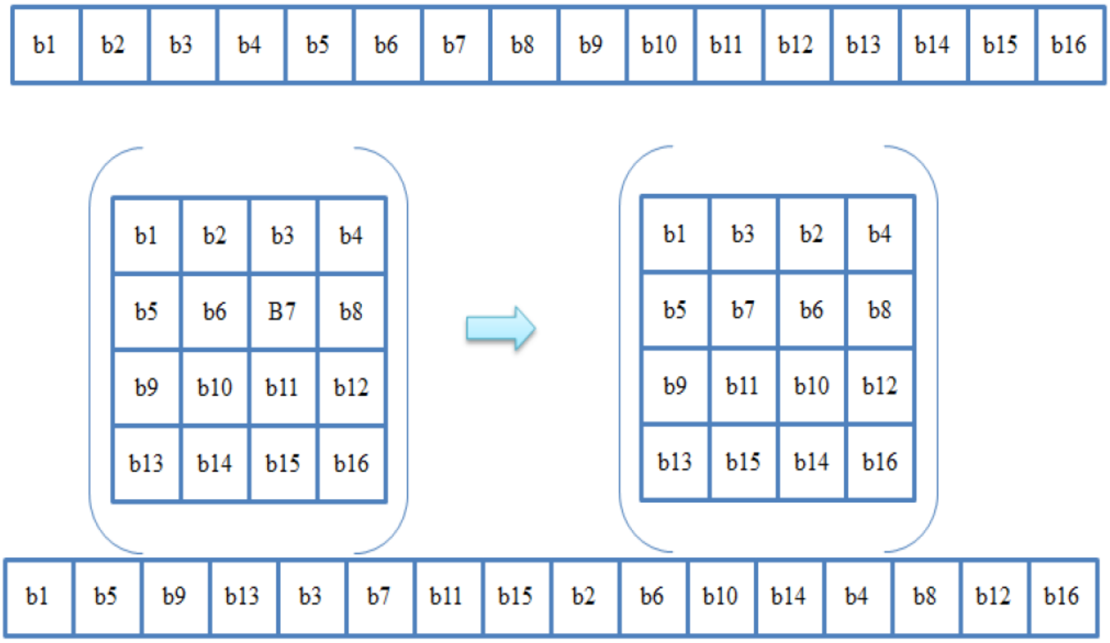

b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15 b16
b1 b2 b3 b4
b5 b6 B7 b8
b9 b10 b11 b12
b13 b14 b15 b16
b1 b3 b2 b4
b5 b7 b6 b8
b9 b11 b10 b12
b13 b15 b14 b16
b1 b5 b9 b13 b3 b7 b11 b15 b2 b6 b10 b14 b4 b8 b12 b16

Figure 3.18: 3G first interleaver [46]

3. Radio Frame Segmentation:
   When the transmission time interval is longer than 10 ms, the input bit sequence is segmented into equally sized segments according to the following equation [46]:

$$y(n_i k) = x(k + (n_i - 1)\tfrac{X}{F}), k = 1, 2, ..., \tfrac{X}{F} \tag{3.17}$$

   The variables n_i, X, and F are the segment number, the number of input bits, and the total number of segments respectively.

4. Second Interleaver:
   This is intra-frame interleaver where interleaving is done frame by frame. The RAM used in implementation has 30 columns. The number of rows varies according to the number of bits in single radio frame. The number of rows is determined by the following equation [46]:

$$R \geq \left\lfloor \frac{N}{30} \right\rfloor \tag{3.18}$$

   The variables R and N represent the number of required rows and the number of bits per frame respectively. Data is read column by column in a certain order stored in LUTs.

3.5.6 Code Division Multiplexing

Since 3G is Code Division Multiplexing (CDM) based, data bits are multiplied by fully orthogonal codes called channelization codes to be transformed into chips in order to increase the bandwidth of the signal and prevent interference. Channelization process is called spreading. Data chips are then multiplied by scrambling codes to differentiate between users in the up-link. Figure 3.19 illustrates the full process where data is converted to chips. Spreading and scrambling codes are stored in ROMs. BPSK mapper is used in modulation.

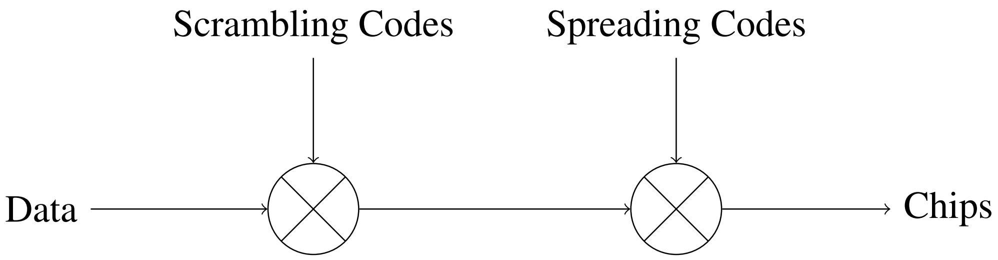


Figure 3.19: CDM block diagram [47]

3.5.7 Chain Utilization

Table 3.10 lists the utilized area for all UMTS transmitter chain blocks in terms of LUTs, BRAMs, and DSPs.

Table 3.10: 3G Transmitter Chain Area Utilization

Chain Block	LUTs	BRAMs	DSPs
CRC	24	0	0
Segmentation	489	0.5	1
Encoder	86	0	0
Concatenation	98	0.5	0
Interleaver	946	2.5	1
Spreading and Scrambling	79	0	0
Mapper	6	0	0

3.6 LTE Transmitter Chain

Figure 3.20 shows the implemented blocks in the LTE chain according to [48, 49].

3.6.1 CRC

CRC is similar to the one used in the previous chains. The used generator polynomial is shown below [49].

$$g(D) = D^{24} + D^{23} + D^{6} + D^{5} + D + 1 \tag{3.19}$$

3.6.2 Segmentation

Although, segmentation uses an algorithm similar to the one used in 3G chain, it is much more complicated as the block size is variable. Figure 3.21 shows the state diagram of the implemented FSM.

3.6.3 Turbo Encoder

Turbo encoder is used due to its ability to provide very low BER and high coding rates. It consists of two convolutional encoders mixed with an interleaver as shown in Figure 3.22. The interleaver stores the bits row by row, then reading is performed using address values calculated from reserved LUTs.

3.6.4 Rate Matching

Rate matching is used to match the number of bits to a certain number specified by the MAC layer. It consists of three main blocks: sub-block interleavers, bit selection, and bit collection. The three blocks re-arrange the bits in a certain form through bunch of complex equations in order to meet the required packet size. Implementation is done using block RAMs and DSPs.

The output of the three sub-block interleavers is transferred to the bit collection block as shown in Figure 3.23. The block output can be represented by a virtual circular buffer.

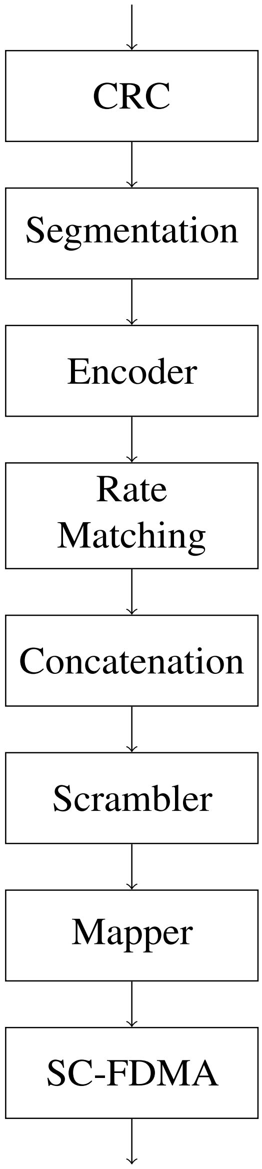


Figure 3.20: LTE transmitter chain block diagram [48]

The length of the circular buffer is $K_w = 3K_\pi$, where the value of k_π is set by the MAC layer. The relation between input and output is derived by the following equations [49]:

$$W_k = {\nu_k}^{(0)} \qquad For \quad k = 0, 1, \ldots, k_\pi - 1 \tag{3.20}$$

$$W_{k_\pi + 2k} = {\nu_k}^{(1)} \qquad For \quad k = 0, 1, \ldots, k_\pi - 1 \tag{3.21}$$

$$W_{k_\pi + 2k + 1} = {\nu_k}^{(2)} \qquad For \quad k = 0, 1, \ldots, k_\pi - 1 \tag{3.22}$$

The signals storing the values of k_π and number of rows pass without any modifications to the bit selection block. The interleaver internal matrix is stored in the order shown below [49]:

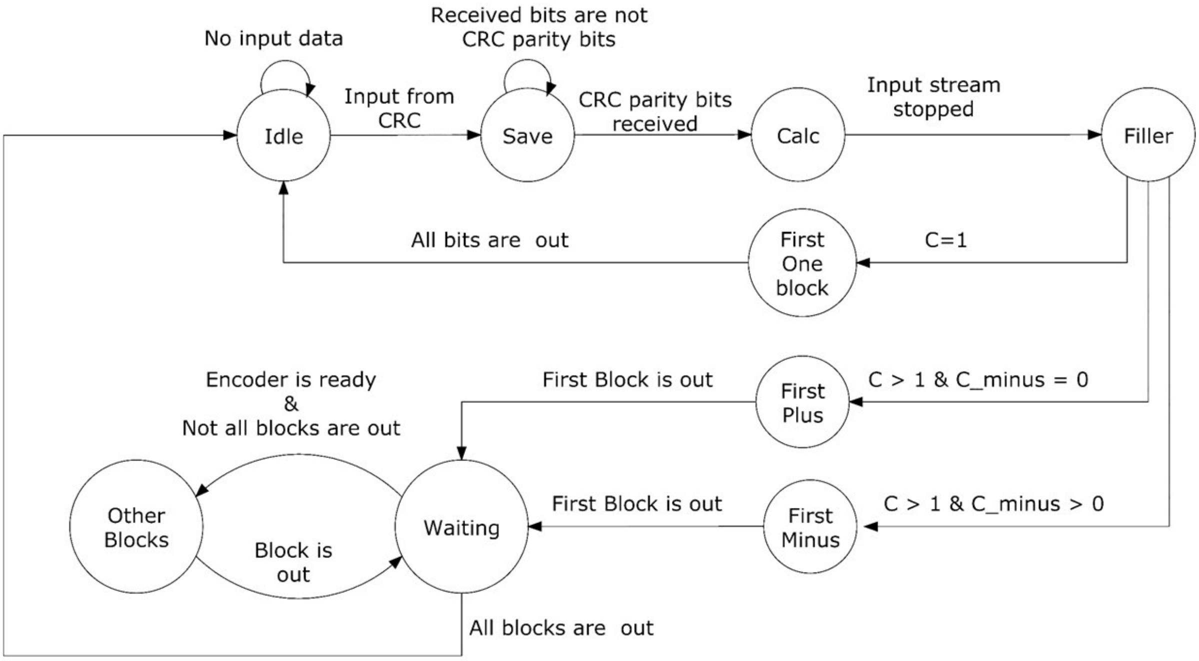


Figure 3.21: 4G segmentation FSM

$$\begin{bmatrix} y_0 & y_1 & y_2 & \cdots & y_{C^{CC}_{subblock}-1} \\ y_{C^{CC}_{subblock}} & y_{C^{CC}_{subblock}+1} & y_{C^{CC}_{subblock}+2} & \cdots & y_{2C^{CC}_{subblock}-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{C^{CC}_{subblock}(R^{CC}_{subblock}-1)} & y_{(C^{CC}_{subblock}+1)(R^{CC}_{subblock}-1)} & y_{(C^{CC}_{subblock}+2)(R^{CC}_{subblock}-1)} & \cdots & y_{C^{CC}_{subblock}R^{CC}_{subblock}-1} \end{bmatrix}$$

The matrix elements are derived by the equation mentioned below, where D is the number of bits [49].

$$y_{N_D+K} = d_k \qquad For \quad k = 0, 1, \ldots, D-1 \tag{3.23}$$

3.6.5 Code Block Concatenation

Since concatenation is the same in both 3G and LTE chains, the block is shared between them.

3.6.6 Scrambler

Scrambler is used to prevent the appearance of long consecutive “1”s and “0”s. The generated scrambling codes are calculated from the set of equations mentioned below [49]:

$$C(n) = mod(\frac{X_1(n+N_c)+X_2(n+N_c)}{2}) \quad , \quad n = 0, 1, \ldots, M_{PN}-1 \tag{3.24}$$

$$X_1(n) = \begin{cases} 1 & , \quad n = 0 \\ 0 & , \quad n = 1, \ldots, 30 \\ mod(\frac{X_1(n+3)+X_1(n)}{2}) & , \quad n = 31, \ldots, M_{PN}-1 \end{cases} \tag{3.25}$$

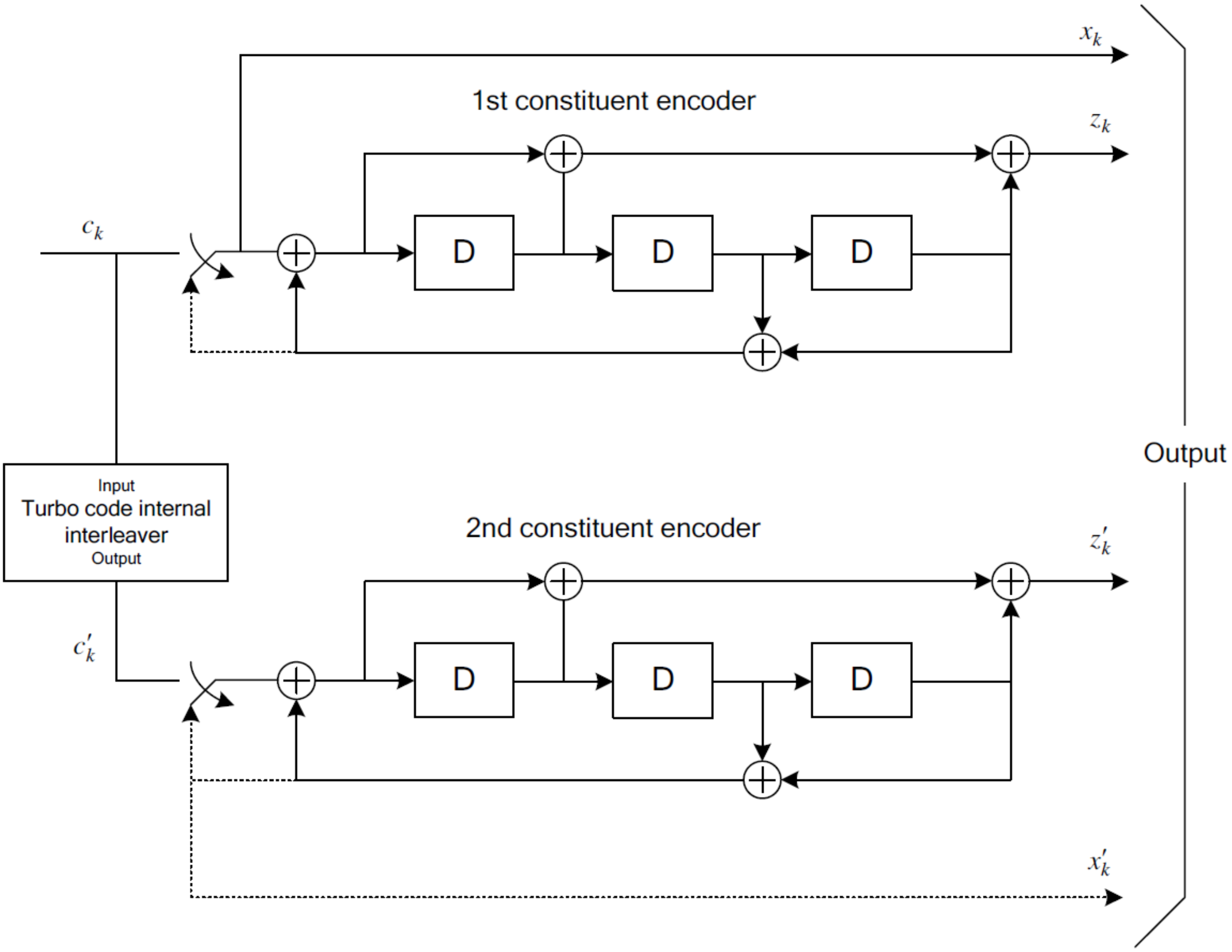


Figure 3.22: 4G turbo encoder [48]

$$X_2(n) = \begin{cases} C_{init} & , \quad n = 0, \ldots, 30 \\ mod(\frac{X_2(n+3)+X_2(n+2)+X_2(n+1)+X_2(n)}{2}) & , \quad n = 31, \ldots, M_{PN} - 1 \end{cases} \tag{3.26}$$

The variable $C(n)$ represents the scrambling sequence. Meanwhile, X_1, and X_2 represent the initial values of the two BRAMs used to generate the scrambling codes. The scrambling sequence generator shall be initialized with C_{init} which is calculated using this equation [49]:

$$C_{init} = 2^{14} * n_{RNTI} + 2^{13} * q + 2^9 * \lfloor \frac{n_s}{2} \rfloor + N_{ID} \tag{3.27}$$

The variable n_{RNTI} corresponds to the RNTI associated with the PUSCH transmission channel, n_s is the index of the sub-frame, N_{ID} is the cell ID, and q is the code-word transmitted on the physical up-link shared chrobustannel.

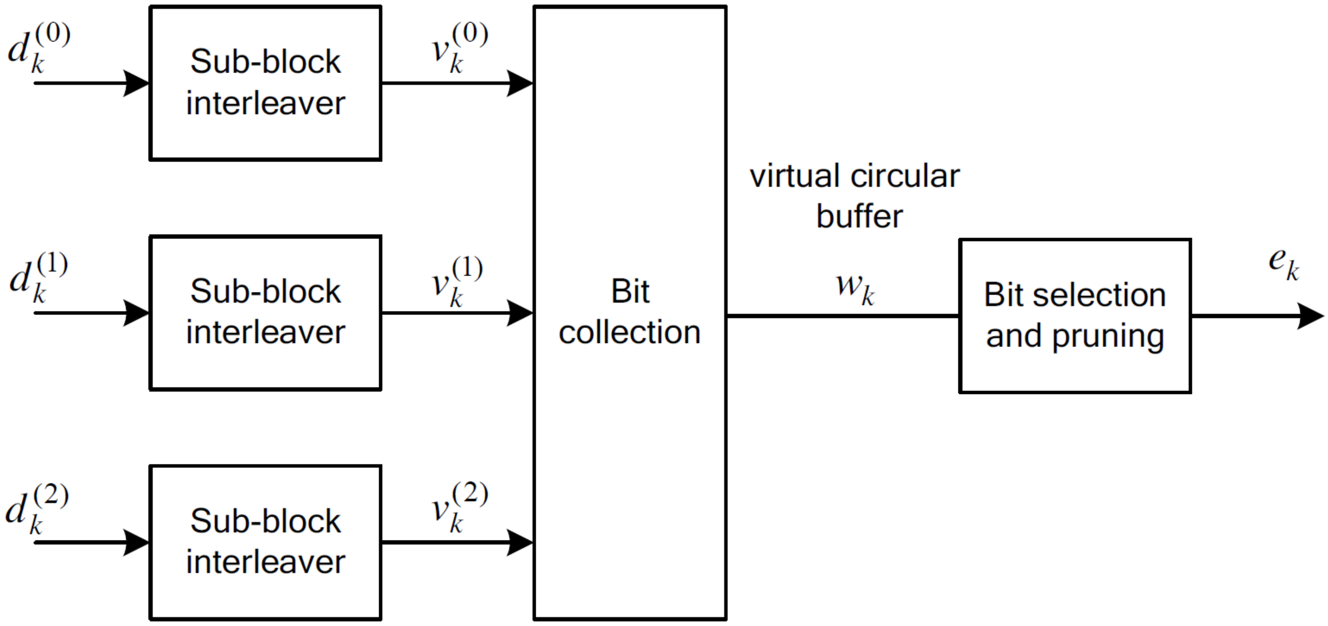


Figure 3.23: 4G Rate matching block diagram [49]

3.6.7 OFDM Section

After mapping the data bits using QPSK mapper, symbols are then transferred to the SC-FDMA block. The SC-FDMA used in LTE up-link is a modified form of the OFDM with similar throughput and complexity. SC-FDMA is composed of DFT where time-domain symbols are transformed to frequency domain symbols and then passed through the standard OFDM modulation. It has the all advantages of OFDM such as being robust against multi-path signal propagation. The internal implementation of the SC-FDMA is performed using 14-point DFT, 128-point IFFT, and a controller as shown in Figure 3.24. The IFFT subcarriers are grouped into sets of 14 subcarriers, each group is called a resource block.

The main advantage of SC-FDMA is the low Peak Average Power Ratio (PAPR) of the transmitted signals. PAPR is a big concern for user equipments, since it relates to the power amplifier efficiency. Low PAPR allows the power amplifier to operate close to the saturation region resulting in high efficiency. This is the main reason behind using SC-FDMA for user terminals.

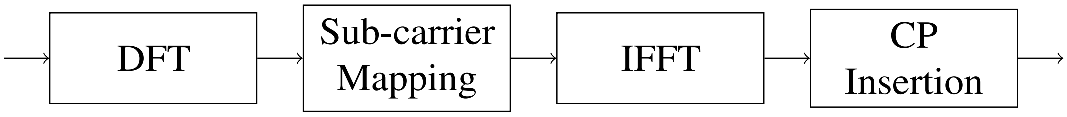


Figure 3.24: OFDM section block diagram

The LTE supported bandwidths are listed in Table 3.11 [49]. The 128-point IFFT with an extended cyclic prefix equals to 32 is chosen for implementation.

Table 3.11: LTE Transmission Schemes [48]

Transmission Bandwidth (MHz)	Frequency (MHz)	IFFT Size	Sub-carriers
1.4	1.92	128	6
3	3.84	256	15
5	7.68	512	25
10	15.36	1024	50
15	23.04	1536	75
20	30.72	2048	100

Since the DFT IP offered by Xilinx is 24-points, it consumes huge amount of area and power. However, the design requires only 14-point DFT. In order to solve this issue, a customized version of the DFT is implemented to save area resources and power consumption. Xilinx LogiCore IP for 128-point IFFT is used.

The IFFT core shows that it is ready to accept a new frame of data by setting the $RFFD$ signal high. Consequently, the input data may start by setting FD_IN high for one or more cycles. Data should be provided over N cycles without interruption. FD_IN can be kept high for multiple cycles, since its value is ignored while $RFFD$ is low. If FD_IN is set permanently high, the core will start a new frame of data input as soon as it is ready. This arrangement provides maximum transform throughput. Alternatively, $RFFD$ can be connected directly to FD_IN to achieve the same behavior. The first input element must be provided in the same cycle the core starts receiving the data.

3.6.8 Chain Utilization

Table 3.12 lists the utilized area for all LTE transmitter chain blocks in terms of LUTs, BRAMs, and DSPs.

Table 3.12: LTE Transmitter Chain Area Utilization

Chain Block	LUTs	BRAMs	DSPs
CRC	106	0	0
Segmentation	529	0.5	0
Encoder	425	1.5	3
Rate Matching	1372	3	0
Concatenation	114	2	0
Scrambler	455	4.5	0
Mapper	77	0.5	0
SC-FDMA	2765	0	13

3.7 Summary

The following chapter lists all implemented blocks in the SDR transmitter. The system includes the physical layer implementation of the five transmitters: Bluetooth, Wi-Fi, 2G, 3G, and LTE.

Chapter 4: SDR Receiver Design

4.1 SDR Receiver Overview

The SDR receiver as illustrated in this chapter inncludes five reciver chains: Bluetooth, Wi-Fi, 2G, 3G, and LTE. A hardware implementation of the physical layer of each chain is being deployed.

4.2 Bluetooth Receiver

Figure 4.1 shows the implemented blocks in the Bluetooth receiver chain.

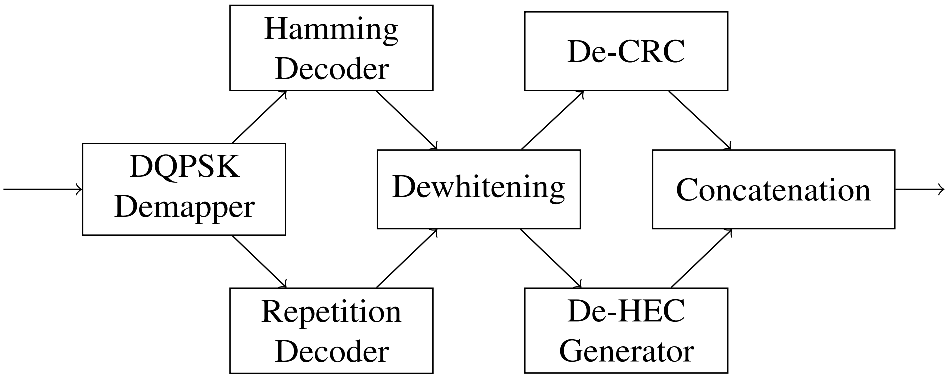


Figure 4.1: Bluetooth receiver block diagram

4.2.1 Demapper

Demodulation of header and payload symbols is performed separately. The same relation mentioned in Table 3.2 is used to repeal the effect of the differential encoder.

4.2.2 Repetition and Hamming Decoders

Decoding the header is performed by converting each 3 serial bits into parallel bits to fasten the rate. The output is taken based on the majority of each 3 bits. The payload bits are segmented according to the following equation [35]:

$$N_{segments} = \frac{N_{Bits}}{15} \tag{4.1}$$

These 15 bits are called the received codeword. The received codeword is multiplied by the transpose of the hamming matrix to produce the syndrome bits. The syndrome bits are 4-bits used to identify the errors in the received codeword using the syndrome table [35]. The hamming matrix is created from the parity matrix and the identity matrix.

4.2.3 Dewhitening, De-HEC, and De-CRC

Dewhitening for both header and payload uses the polynomial equation shown in Figure 4.2. The De-HEC uses the circuit in Figure 4.3 to remove the last 8 bits in the header data if the division remainder is equal to zero. De-CRC removes the last 16 bits in the payload data using the polynomial shown in Figure 4.4.

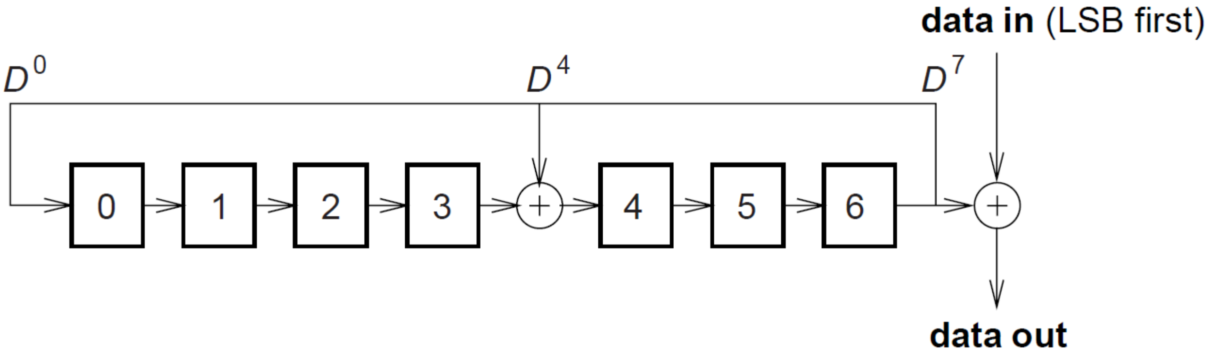


Figure 4.2: Dewhitening polynomial [35]

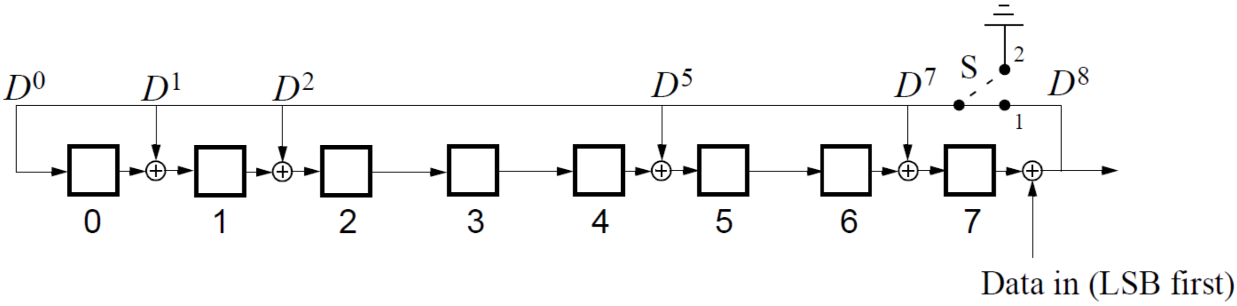


Figure 4.3: De-HEC polynomial [35]

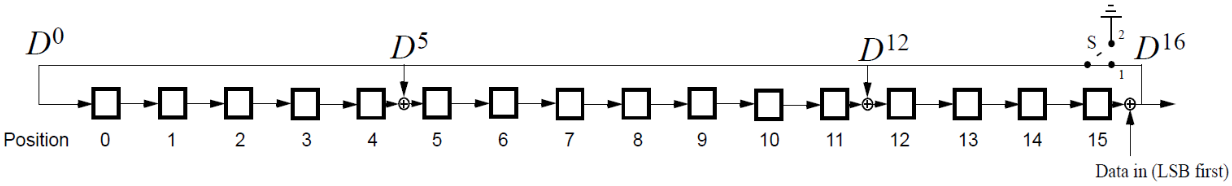


Figure 4.4: De-CRC polynomial [35]

4.2.4 Chain Utilization

Table 4.1 lists the utilized area for all Bluetooth receiver chain blocks in terms of LUTs, BRAMs, and DSPs.

Table 4.1: Bluetooth Receiver Chain Area Utilization

Chain Block	LUTs	BRAMs	DSPs
Demapper	6	0	0
Hamming Decoder	181	1.5	0
Repetition Decoder	242	1	0
Dewhitening	7	0	0
De-CRC	89	0.5	0
De-HEC	75	0	0
Concatenation	80	0	0

4.3 Wi-Fi Receiver

Figure 4.5 shows the implemented blocks in the Wi-Fi receiver chain.

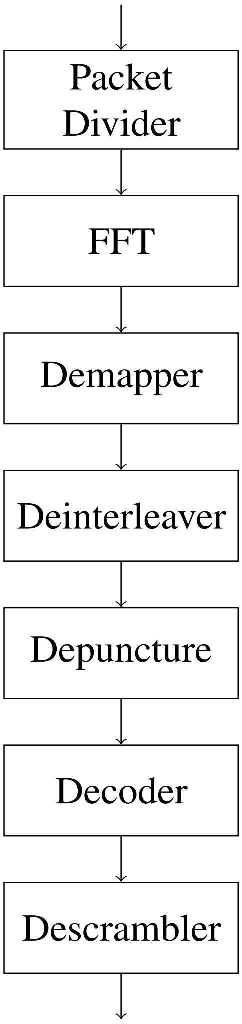


Figure 4.5: Wi-Fi receiver chain block diagram

4.3.1 OFDM Section

The packet divider receives the modulated symbols and stores them in a memory block, then removes the reserved preamble bits from the stored data. The rest of the data is delivered to the FFT block which consists of four main blocks: the controller, the core, and two RAMs for reading and writing processes. The controller responsibilities are:

- Removing the cyclic prefix extension from the received symbols.
- Managing the read and write processes in the two BRAMs used before the core.

Each RAM has the capacity to store 64 symbols. The controller keeps the FFT core operation pipelined by controlling the read and write processes. Therefore, while the first RAM is reading from the packet driver, the FFT core is reading from the second RAM and so on. The demapper specifies the decision region of the received real and imaginary symbols from the FFT, then converts the symbols to a stream of bits. The received symbols are stored in stack memories as shown in Figure 4.6 in order to:

- enable the demapping process to work while the FFT core is processing the next expected block of symbols.
- select the symbols that doesn't contain nulls and pilots to be demapped.

The demapped bits are stored in BRAM before entering the deinterleaver.

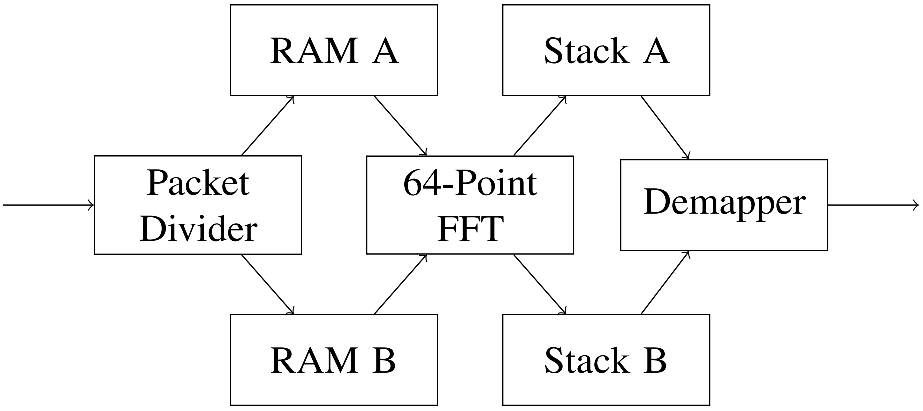


Figure 4.6: Wi-Fi FFT and demapper block diagram

4.3.2 Deinterleaver

Deinterleaving process is defined by two permutations. These permutations represent the inverse equations to the permutation equations in the interleaver block in the transmitter. Three variables are used in the inverse equations. The variables "j", "d", and "e" are similar to "k", "i", and "j" in the transmitter respectively.
The first permutation is defined by the following equation [36]:

$$d = s\times\left\lfloor \frac{j}{s} \right\rfloor + (j + \left\lfloor \frac{16j}{N_{CBPS}} \right\rfloor)\%s \text{ for } j = 0,1,\dots,N_{CBPS}-1 \tag{4.2}$$

The second permutation is defined by the following equation [36]:

$$e = 16d - (N_{CBPS} - 1) \times \left\lfloor \frac{16d}{N_{CBPS}} \right\rfloor \text{ for } d = 0, 1, \ldots, N_{CBPS} - 1 \tag{4.3}$$

Implementation of the two equations is similar to the transmitter.

4.3.3 Depuncture

The Depuncture pads dummy bits in the position of the removed bits by the puncture. The position of the removed bits is defined by [36] according to the coding rate. Figure 4.7 shows the position of the inserted dummy bits for coding rate equals to 3/4. Implementation is similar to the transmitter, since it uses a BRAM whose address is controlled by special logic to match the pattern of the column vector to extract the desired bits.

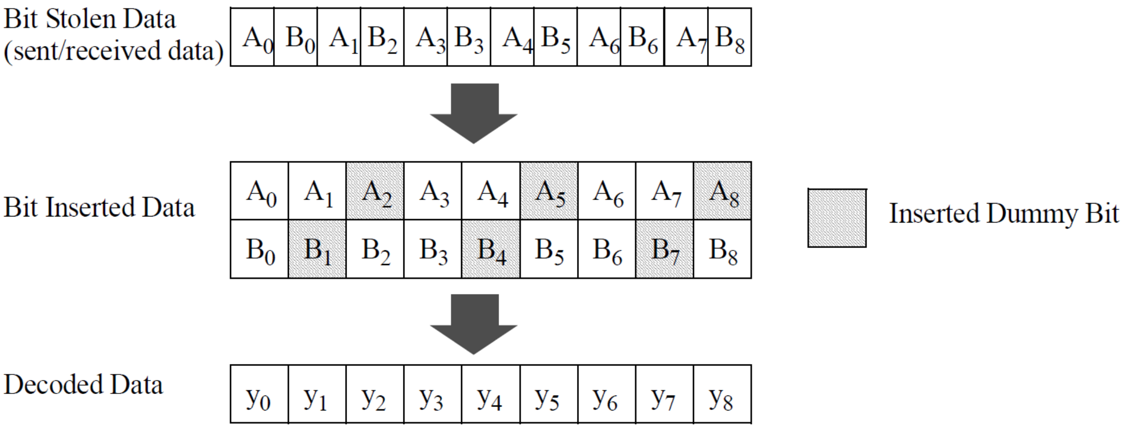

Bit Stolen Data
(sent/received data)
A0 B0 A1 B2 A3 B3 A4 B5 A6 B6 A7 B8
Bit Inserted Data
A0 A1 A2 A3 A4 A5 A6 A7 A8
B0 B1 B2 B3 B4 B5 B6 B7 B8
Inserted Dummy Bit
Decoded Data
y0 y1 y2 y3 y4 y5 y6 y7 y8

Figure 4.7: Wi-Fi depuncturing algorithm [36]

4.3.4 Viterbi Decoder

Viterbi decoder is used because of its ability for error detection and correction. The decoder either requests re-transmission, or starts correcting the received bit stream according to its type. The constraint length in Wi-Fi (order of generator polynomial + 1) equals to 7. The decoder implementation consists of six major blocks as shown in Figure 4.8.

1. BMU: Branch Metric Unit is responsible for calculating the hamming distance of each branch in the trellis.
2. PMU: Path Metric Unit consists of Add Compare Select Unit (ACSU) responsible for calculating the hamming distance of the whole path, comparing paths together, and selecting the desired one.
3. TBU: Trace Back Unit is responsible for storing the correct paths in order to trace back the survivor path ultimately to deduce the original bits.
4. Metric Memory: Responsible for storing the path metric of each branch.
5. Viterbi Controller: Responsible for controlling all blocks.

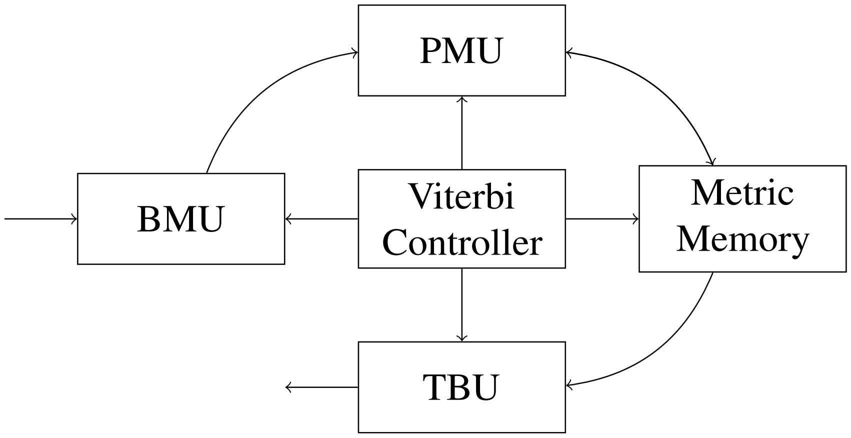


Figure 4.8: Viterbi block diagram [50]

The flow chart of the decoder algorithm illustrated in Figure 4.9 shows that the BMU first starts to calculate the branch metric. In order to calculate the path metric, the ACS unit starts its operation on each branch. The path information is stored in the metric memory, then checking is done on the reference model (the trellis). If the stages are over, then tracing back is initiated to decode the data. If not, the operation is repeated till the end of trellis stages. The algorithm can be explained briefly in the following steps:

1. Receive one input code word (2 bits or 3 bits corresponding to the coding rate).
2. Calculate the branch metric.
3. Read the previous path metric for all states from the metric memory.
4. Add the branch metric to the path metric for the old state.
5. Compare the sum for paths arriving at the new state (there are only two paths).
6. Select the path with the smallest value which is called the survivor path. If both path metrics are equal, then choose anyone.
7. Write the survivor path in the survivor memory to be used in the trace back process.
8. Write the new path metric in the metric memory unit.
9. Begin the trace back process when the sliding window reaches its end.

The trellis in Figure 4.10 shows the possible transitions for an encoder input sequence ‘1 0 1 1 1 0 0’. Viterbi algorithm uses the received version of the encoded bit sequence to find the most likely path through the trellis diagram representing the encoder state machine. Once the most likely path is determined, the encoder data bits that led to follow this path are implied, and these are the output. Viterbi algorithm is the optimum algorithm, since it minimizes the probability of error. However, the main drawback of using Viterbi decoders is the large cost in terms of chip area.

Figure 4.11 illustrates the conventional decoding process. Consider the received sequence containing errors to be decoded is ‘11 11 00 10 01 11 11’. The output of the decoder in such case will be ‘11 01 00 10 01 10 11’. After finishing the first two steps in

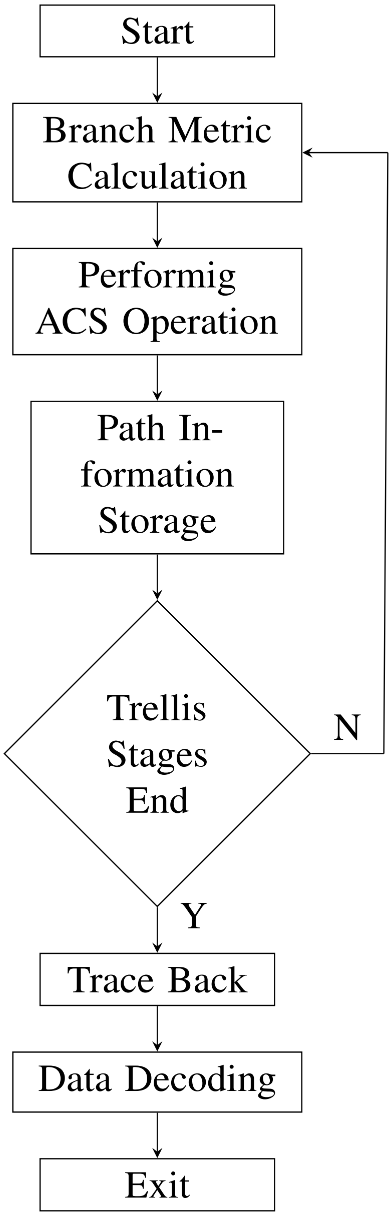


Figure 4.9: Viterbi decoder algorithm flow chart

the algorithm, the path metrics are calculated to reach each state in the trellis. Ultimately, the survivor unit traces back the optimum path which will always start from state zero.

The BMU receives the code signals, calculates the distances with all possible branch metrics, then generates the output distances. Values generated are depending on the value of ACS segment as shown in Figure 4.12.

The distance for each branch is the number of error bits between the received code word and the output of the branch. Distances are stored in 2 bits. In case of coding rate is 1/2, the error can be in 0, 1, or 2 bits which is represented in 2 bits. Similarly, the error in case of coding rate is 1/3, the error can be in 0, 1, 2, or 3 bits which is represented in 2 bits as well. The hamming distance in each case is calculated using the combinational logic shown in Figures 4.13 and 4.14. Distance calculations in both cases of coding rates are listed in Tables 4.2 and 4.3.

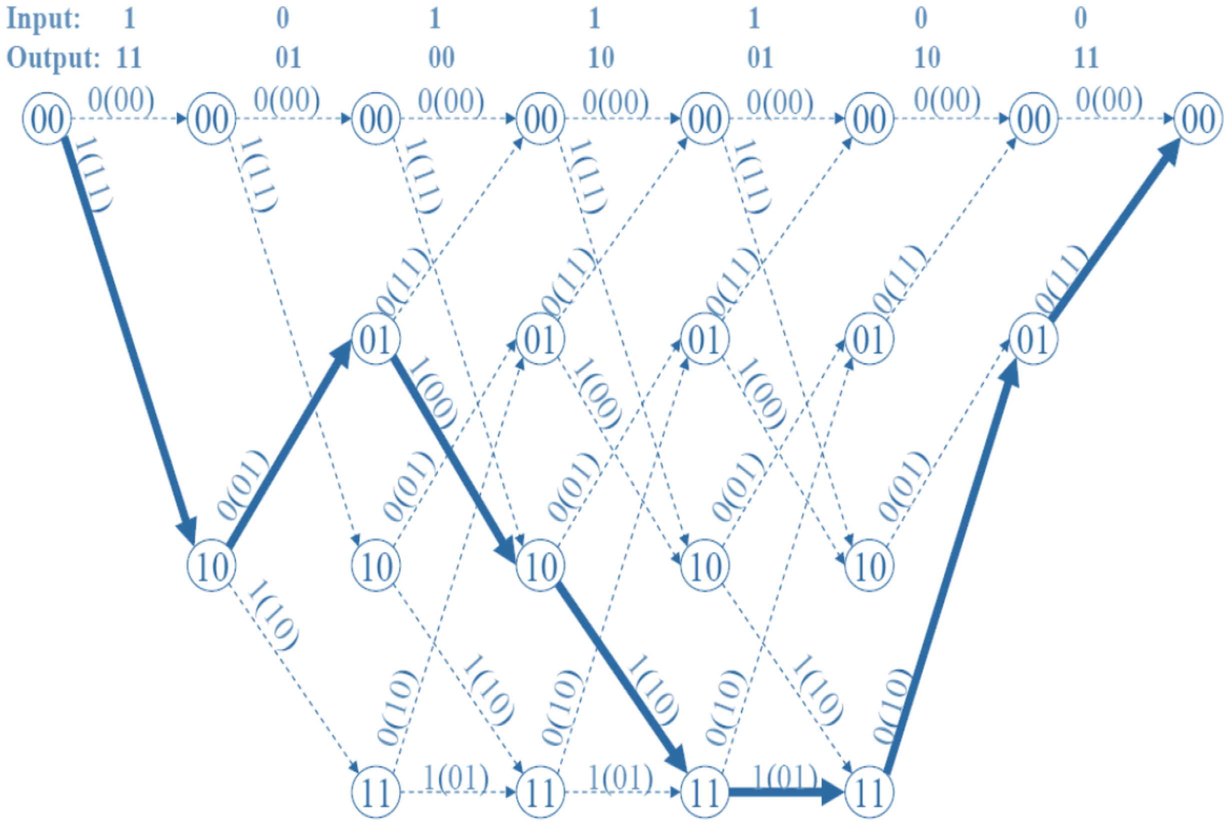


Figure 4.10: Convolutional encoder terellis example

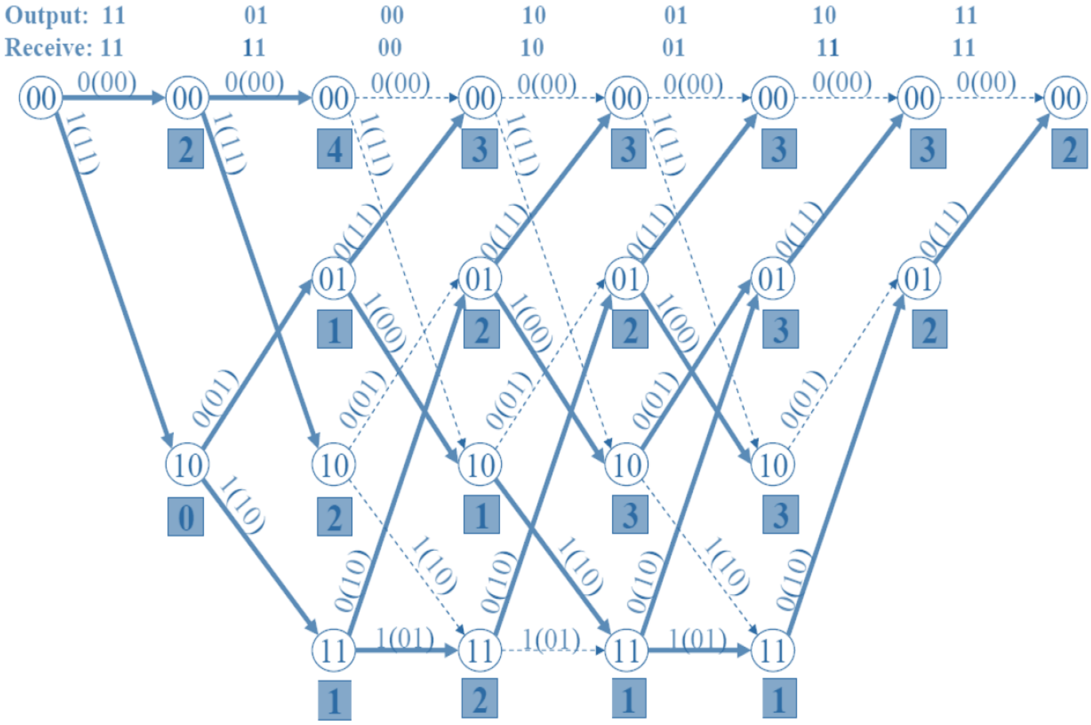


Figure 4.11: Trellis diagram for error decoding

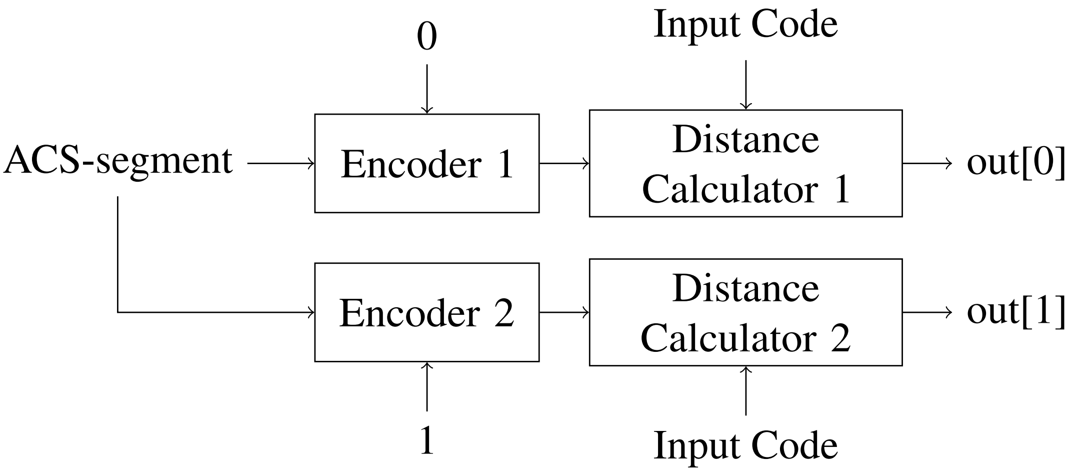


Figure 4.12: BMU block diagram

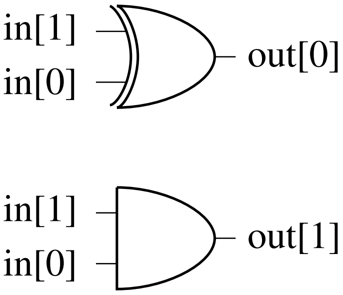


Figure 4.13: Distance calculator for coding rate 1/2

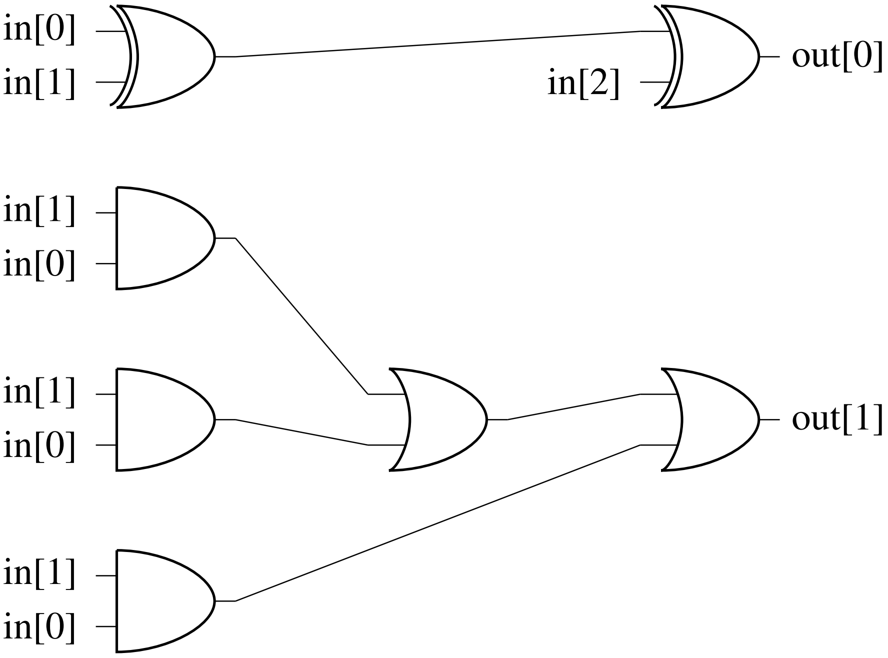


Figure 4.14: Distance calculator for coding rate 1/3

Table 4.2: Hamming Distance in case of coding rate 1/2 [36]

Input Bits	Calculated Distance in Bits
00	00
01	01
10	01
11	10

Table 4.3: Hamming Distance in case of coding rate 1/3 [36]

Input Bits	Calculated Distance in Bits
000	00
001	01
010	01
011	10
100	01
101	10
110	10
111	11

The ACS unit adds the path metric to the distance, then compares the new path metric. Finally, the chosen path metric is stored in the metric memory as the survivor path. Two BRAMs are needed to save the calculated consecutive metric values where each has its own index.

4.3.5 Descrambler

Descrambler implementation is performed as shown in Figure 4.15.

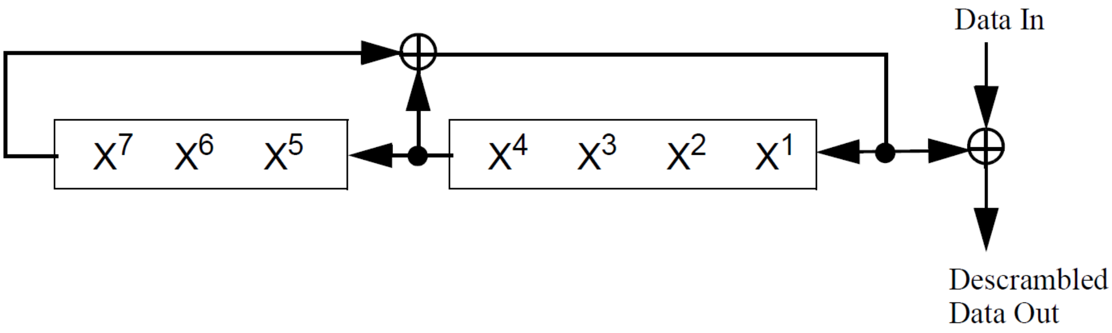


Figure 4.15: Wi-Fi descrambler [36]

4.3.6 Chain Utilization

Table 4.4 lists the utilized area for all Wi-Fi receiver chain blocks in terms of LUTs, BRAMs, and DSPs.

Table 4.4: Wi-Fi Receiver Chain Area Utilization

Chain Block	LUTs	BRAMs	DSPs
Packet Divider	26	0	0
FFT	2079	0.5	6
Demapper	330	0.5	0
Deinterleaver	367	2	5
Decoder	803	1.5	0
Descrambler	139	0	0

4.4 2G Receiver

Figure 4.16 shows the implemented blocks in the 2G receiver chain.

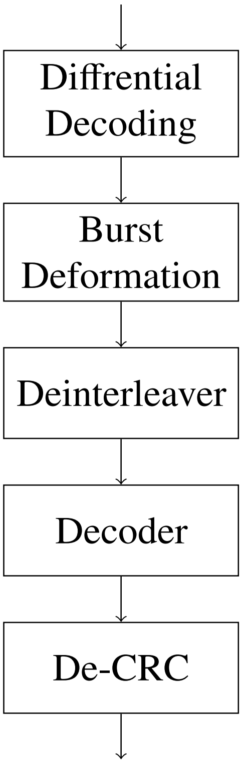


Figure 4.16: 2G receiver chain block diagram

4.4.1 Differential Decoding

Differential decoding is responsible for retrieving the data to its original form. Implementation is done through xoring the old bit with the new signed bit.

4.4.2 Burst Deformation

Burst deformation extracts the data bits from the burst, then equalizes the channel effect using the training sequence. Information about the channel is extracted from the steal flag bits. If the steal bits are zeros, the channel type is Traffic Channel (TCH). If the steal bits are ones, the channel type is Fast Associated Control Channel (FACCH).

4.4.3 Deinterleaver

The deinterleaver concatenates the data segments extracted from the burst to reconstruct the frame. Then, it re-arranges the bits in its correct order by writing them column by column and reading them row by row in 8×57 matrix.

4.4.4 Viterbi Decoder

Same Viterbi decoder used in Wi-Fi chain is used here as well. However, the constraint length in 2G is 5 and it decodes only the first 378 bits. The remaining 78 bits are buffered to the next block.

4.4.5 De-CRC

De-CRC and bit reordering perform the following steps as illustrated in Figure 4.17:

1. Detaches the tail bits from the received sequence if they are all zeros.
2. Reorders the remaining bits to their original order.
3. Removes the CRC parity bits.
4. Checks if the CRC remainder is equal to 0 for error detection.

4.4.6 Chain Utilization

Table 4.5 lists the utilized area for all GSM receiver chain blocks in terms of LUTs, BRAMs, and DSPs.

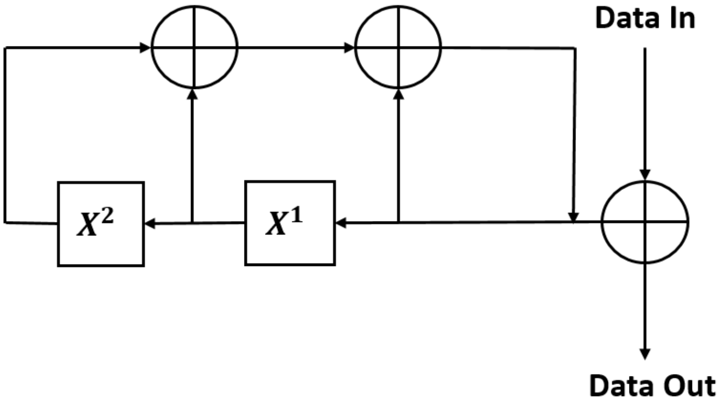


Figure 4.17: 2G de-CRC

Table 4.5: 2G Receiver Chain Area Utilization

Chain Block	LUTs	BRAMs	DSPs
Differential Decoder	3	0	0
Burst Deformation	77	0.5	0
Deinterleaver	165	0.5	0
Decoder	665	1.5	0
De-CRC	155	0	0

4.5 3G Receiver

Figure 4.18 shows the implemented blocks in the 3G receiver chain.

4.5.1 Code Division Multiplexing

BPSK demapper is used to retrieve the data bits. Descrambling operation is deployed by multiplying the received bit stream data from the demapper by the same scrambling code used at the transmitter. Despreading is performed by multiplying the descrambled data periodically by the same spreading code used at the transmitter. The spreading code used to generate the despreaded data is “(1,1,-1,1)” [47].

4.5.2 Deinterleaver

The deinterleaver block consists of four main blocks: radio frame segmentation, second deinterleaver, radio frame concatenation, and first deinterleaver. The four blocks should re-arrange the received bits to repeal the effect of the interleaver at the transmitter as

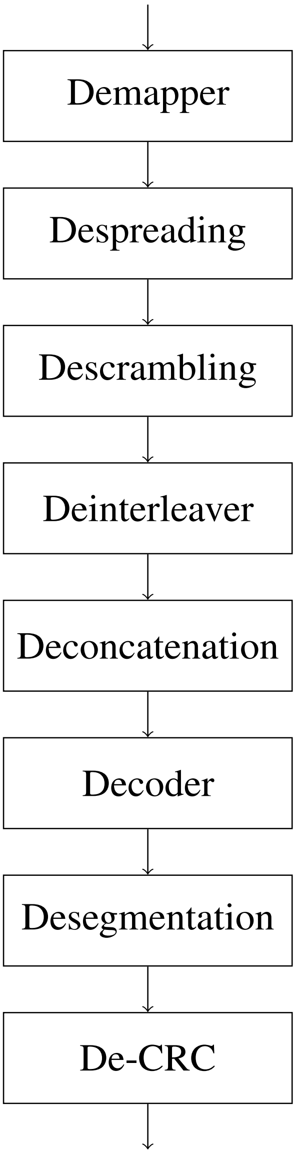


Figure 4.18: 3G receiver chain block diagram

shown in Figure 4.19. Implementation is done using BRAMs whose address is controlled by specific logic.

4.5.3 Deconcatenation

Deconcatenation of bit sequence takes place, if the block size is larger than 504×2 bits. The code blocks after deconcatenation are equally sized. Implementation is done using the same FSM used in the segmentation block at the transmitter.

4.5.4 Viterbi Decoder

Same Viterbi decoder used in Wi-Fi receiver is used here. Since the constraint length in 3G is equal to 9, the size of the trace-back memory and the metric memory will be larger than the one used in Wi-Fi and 2G.

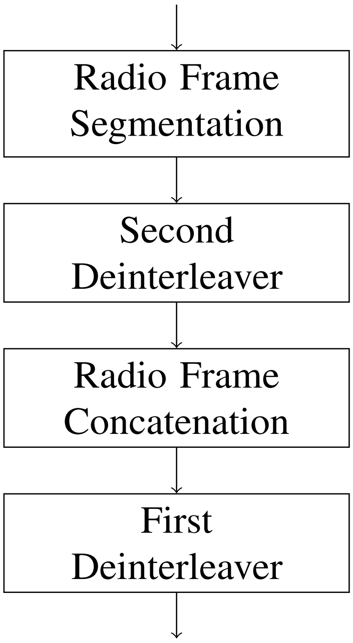


Figure 4.19: 3G deinterleaver block diagram

4.5.5 Desegmentation

Desegmentation has the same design and implementation of the concatenation block explained in transmitter.

4.5.6 De-CRC

De-CRC is provided for error checking. The entire received data block is used to calculate the CRC parity bits. CRC bits are being punctured from the received bits, then CRC parity bits are generated using the same generator polynomial equations used at the transmitter. Finally, a comparison is being done between the generated bits and the received bits to decide out if the data was correct or erroneous. Figure 4.20 shows the used generator polynomial.

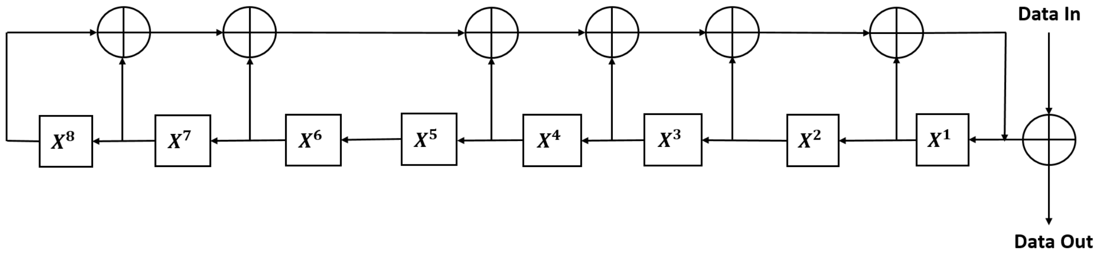


Figure 4.20: 3G de-CRC

4.5.7 Chain Utilization

Table 4.6 lists the utilized area for all UMTS receiver chain blocks in terms of LUTs, BRAMs, and DSPs.

Table 4.6: 3G Receiver Chain Area Utilization

Chain Block	LUTs	BRAMs	DSPs
Demapper	45	0	0
Despreading and Descrambling	78	0	0
Deinterleaver	636	2.5	0
Deconcatenation	424	0.5	1
Decoder	2202	13.5	0
Desegmentation	145	0.5	0
De-CRC	27	0	0

4.6 LTE Receiver

Figure 4.21 shows the implemented blocks in the LTE receiver chain.

4.6.1 OFDM Section

Inverse SC-FDMA differs from the one used at the transmitter side in the arrangement of the internal blocks. The received data first enters the 128-point FFT, then passes through 14-point IDFT as illustrated in Figure 4.22. Implementation is performed using a controller similar to the one used at the transmitter. Unlike all other receivers, the LTE receiver is soft decision based. Hard decision is done at the decoder which converts the symbols to bits again. This implies that all receiver blocks from the demapper to the decoder are soft decision (symbol domain). QPSK demapper with soft in-soft out is used to calculate the log likelihood probability of the input symbols. The resultant demapped data is associated with the probability of its correctness.

4.6.2 Descrambler

Descrambler multiplies the symbols by “-1” if the polynomial output is “1”, and passes the data unchanged if the polynomial output is “0”. Multiplication with “-1” is performed by calculating the two’s complement.

4.6.3 Code Block Deconcatenation

Deconcatenation is the same as the segmentation block at the transmitter.

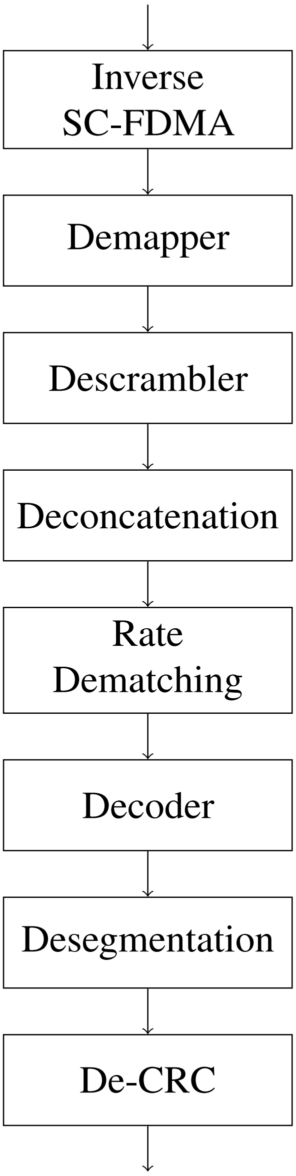


Figure 4.21: LTE receiver chain block diagram

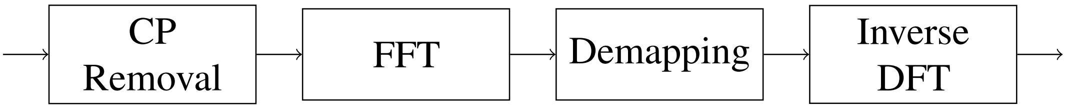


Figure 4.22: OFDM section block diagram

4.6.4 Rate Dematching

Rate dematching block receives the data bits according to the pre-specified MAC rate. Bit deselection block removes any additional bits that were padded to match the required rate. Bit decollection block converts the bits from serial to three parallel paths. Finally, each path is deinterleaved by switching the read and write functions of the interleaver defined at the transmitter.

4.6.5 Turbo Decoder

Turbo decoder is the most complicated block in all implemented chains. It is responsible for error detection and correction. As shown in Figure 4.23, implementation is done using two soft in-soft out Viterbi decoders, two interleavers, two deinterleavers, and a hard decision block. The design is based on an iterative algorithm to increase the accuracy of the bit correction. The symbols “S1, P1, and P2” represent the systematic bit, the output of upper encoder, and the output of lower encoder respectively.

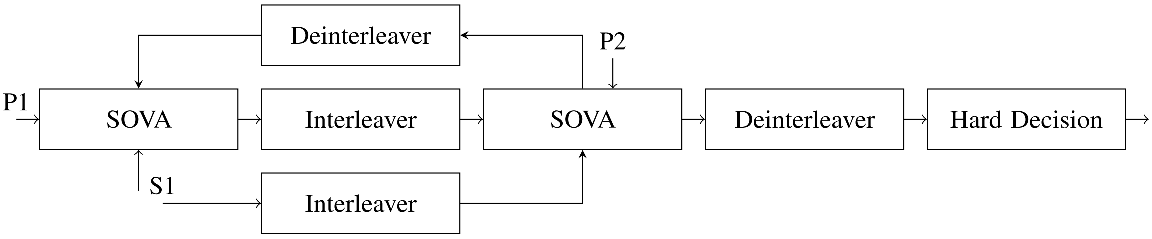


Figure 4.23: Turbo decoder block diagram [51]

4.6.6 Desegmentation

Desegmentation has the same design and implementation as the concatenation block explained in the transmitter. However, it contains an internal de-CRC block to repeal the effect of the CRC.

4.6.7 De-CRC

De-CRC has the generator polynomial shown in Figure 4.24.

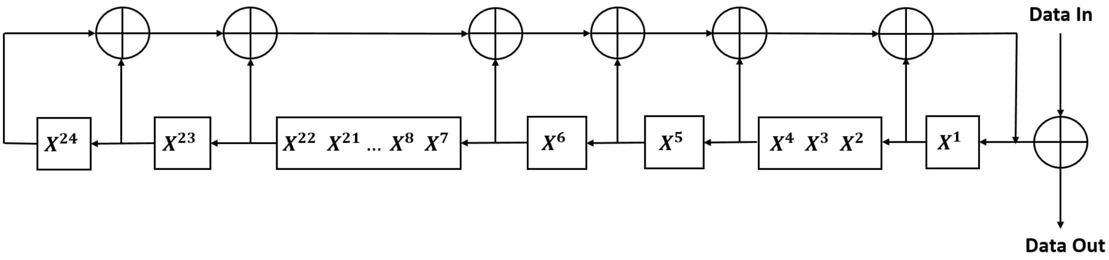


Figure 4.24: LTE de-CRC

4.6.8 Chain Utilization

Table 4.7 lists the utilized area for all LTE receiver chain blocks in terms of LUTs, BRAMs, and DSPs.

Table 4.7: LTE Receiver Chain Area Utilization

Chain Block	LUTs	BRAMs	DSPs
Inverse SC-FDMA	2942	0	15
Demapper	83	3	0
Descrambler	476	4.5	0
Deconcatenation	447	1.5	0
Rate Dematching	585	3	0
Decoder	2655	8	6
Desegmentation	177	2	0
De-CRC	87	0	0

4.7 Summary

The following chapter lists all implemented blocks in the SDR receiver. The system includes the physical layer implementation of the five receivers: Bluetooth, Wi-Fi, 2G, 3G, and LTE.

Chapter 5: FPGA Prototyping

5.1 Test Environment

FPGA prototyping is performed using Zynq-7000 SoC ZC702 Evaluation Kit. Figure 5.1 shows the two main parts of the chip [23]:

1. **Processing System (PS)**:
   The static part of the chip including the processor that is responsible for interfacing with peripherals through AXI Bus Protocol, and executing user programs [31].

2. **Programmable Logic (PL)**:
   The reconfigurable part of the chip that has sufficient resources for deploying the communication systems. The available resources are: 53200 LUTs, 220 DSPs, and 140 BRAMs.

The kit does not only offer DPR technology, but also provides bunch of useful peripherals in the testing process. Figure 5.1 shows some peripherals that are used in the DPR flow and testing as well. The used peripherals are:

1. **SDIO**: Used in interfacing the SD card with the Double Data Rate (DDR) memory with the aid of the processor. The input data files and the partial bit stream files used in the DPR flow are stored in the SD card.

2. **JTAG**: Used in programming the FPGA through the PC.

3. **UART**: Used in interfacing the FPGA with the PC while debugging the design.

4. **DDR CTRL**: Used in interfacing the processor with the DDR memory.

Each communication standard specifies the transmitter's legitimate frequency, in order to modulate its symbols on. Each chain has its own major input frequency that is used to generate the required distributed clocks in order to mimic the standards. The processor can offer up to 4 different clocks. The major operating clocks derived by the processor to feed the communication chains are:

1. **200MHz**: Used for AXI Bus, 2G, Wi-Fi, and LTE chains.

2. **100MHz**: Used for Internal Configuration Access Port (ICAP).

3. **55.5MHz**: Used for Bluetooth chain.

4. **15.3MHz**: Used for 3G chain.

As mentioned earlier, the input data is stored in the SD card and then transferred to the DDR within the run-time. The data flow in the PL side, shown in Figure 5.2, is conducted using the following steps:

1. the input data is transferred from the DDR memory to be stored in the Direct Memory Access (DMA) that adjusts the rate according to the clock of each wireless communication system.

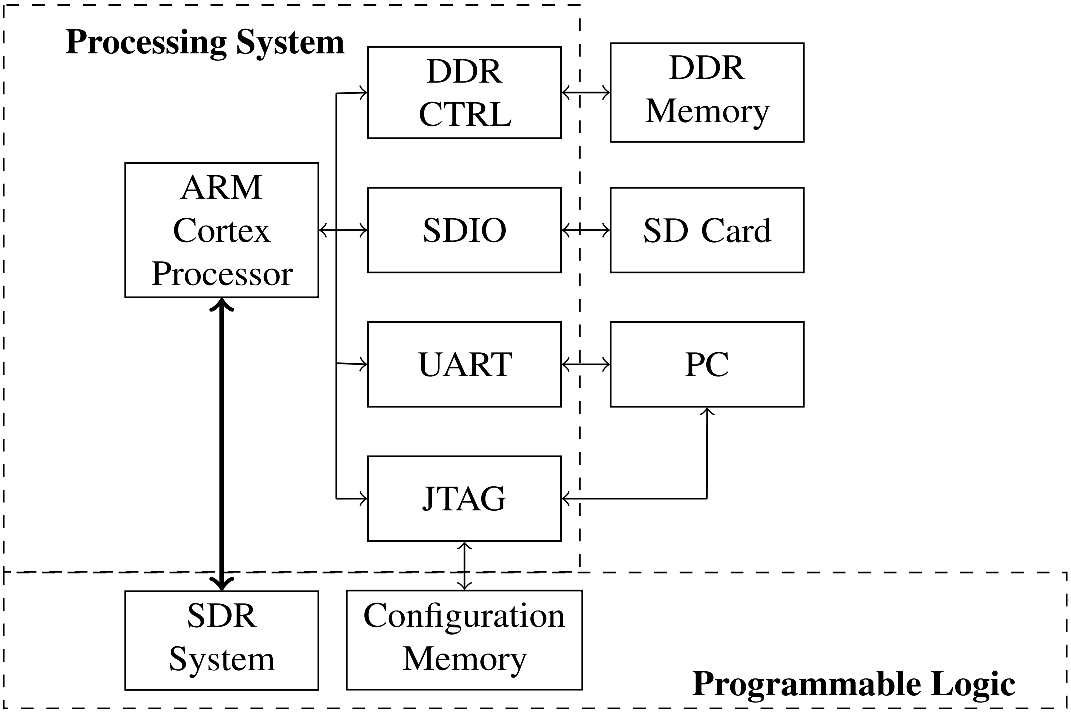


Figure 5.1: Zynq-7000 SoC ZC702 evaluation kit peripherals [27]

2. an intermediate block “Input Interface” is used to adjust the data input rate and system reset.

3. data is transferred through the system.

4. another intermediate block “Output Interface” is used to adjust output data rate for the DMA.

5. the output data is stored in a second DMA to be finally transferred to the DDR for verification.

Testing the five chains is performed through running a C-code program that uses Xilinx APIs to apply the input data for each system and extract the results. Figure 5.3 shows the program flow chart. The states are listed below:

- **Hardware Initialization**: Starts by resetting the system, then the processor sends the required parameters for DMAs, input interfaces, and Xilinx PRC [34]. The hardware initialization step includes the sub-steps mentioned in Figure 5.4.

- **Reconfiguring Chosen System**: The processor triggers the PRC with the chosen system to be loaded.

- **Testing System**: The test environment flow discussed in Figure 5.2 is launched.

- **Asking For Input**: The user has the option to whether program the FPGA with the partial bit files, or test the system.

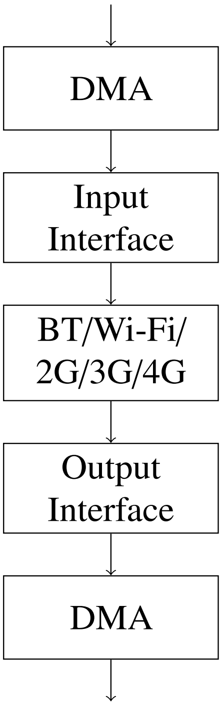


Figure 5.2: Test environment on the PL side

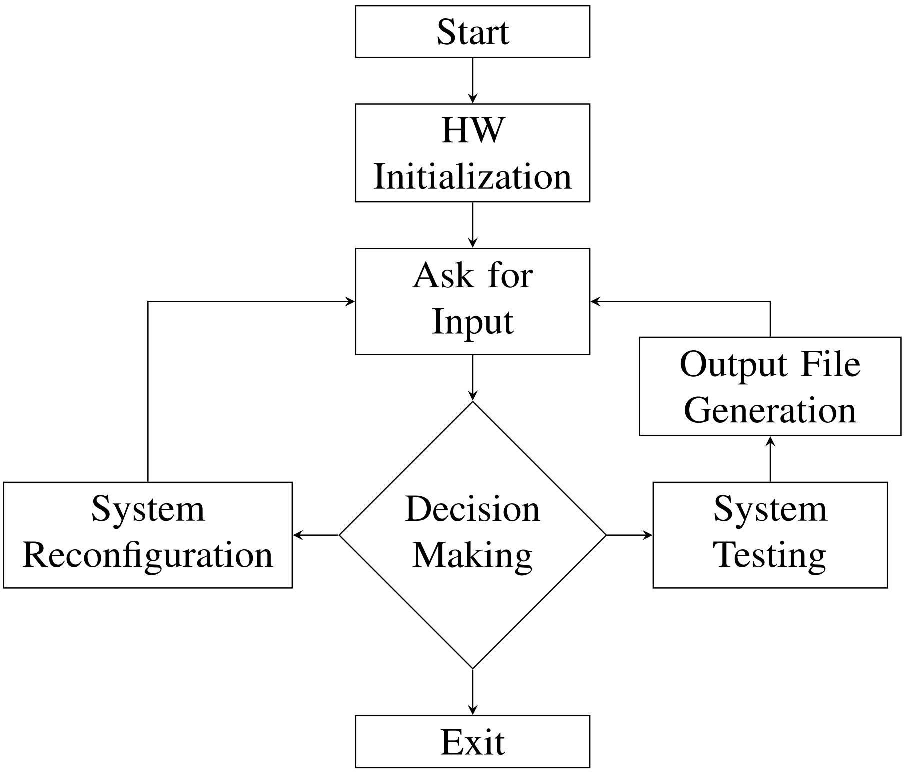


Figure 5.3: C-code flow chart

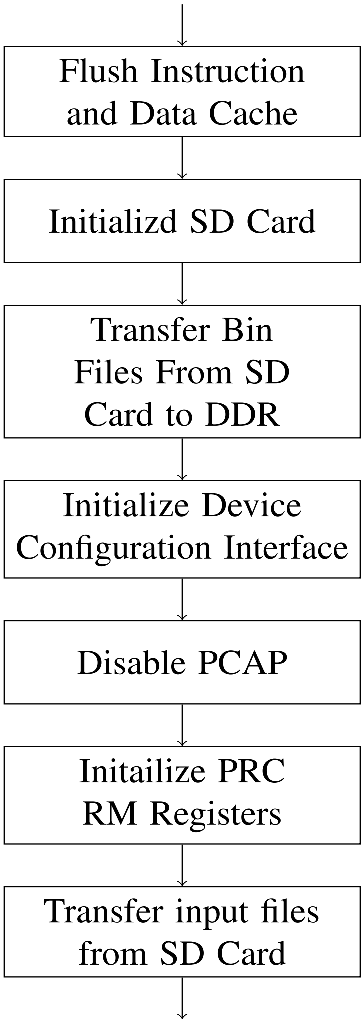

Flush Instruction and Data Cache
Initializd SD Card
Transfer Bin Files From SD Card to DDR
Initialize Device Configuration Interface
Disable PCAP
Initailize PRC RM Registers
Transfer input files from SD Card

Figure 5.4: Hardware initialization steps

5.2 Block Design Implementation

In order to deploy the test environment mentioned earlier to serve the DPR flow, the block design shown in Figure 5.5 is established. The implemented design consists of the following building blocks:

1. **DMA**: The memories used to transfer the system input data from the DDR memory to the communication systems.
2. **Interfaces**: The interfaces used to modify the input/output rate transferred from the DMAs to be sampled correctly by the communication systems.
3. **ARM processor**: The ARM is used to generate the main clock frequencies for all communication systems and synchronize between all other blocks.
4. **PRC**: Xilinx Partial Reconfiguration Controller IP is used to control the ICAP for the DPR flow [52].

5. **ILA**: Xilinx Interactive Logic Analyzer is used to capture the values of specific signals in the design within the run-time. This is performed by setting debugging probes on the required signals and storing the values in memories.

6. **Reset systems**: Since each communication system has its own frequency of operation, synchronized reset blocks for each main clock are used to reset each system.

7. **Communication systems**: The communication systems black boxes that have the same I/O ports of the synthesized design in each chain.

8. **Cross bars**: Xilinx offers AXI4 interface used for IP communication.

5.3 DPR Flow Steps

In this section, Xilinx DPR flow design steps are introduced through implementing a multi-standard SDR system using Vivado and SDK (Version: 2015.2) tools [54].

1. **Step 1: Creating top level black box has the same I/O ports for all RMs**
   The black box module is a wrapper module where input and output ports are defined without performing any logic. The top level module of each Reconfigurable Module (RM) must have the same module name and same I/O ports. This module is connected to the rest of IPs used in the static design. The internal implementation of the black box module is modified later according to which communication system is deployed.

2. **Step 2: Synthesize static and reconfigurable modules separately**
   In this step, the static design of DPR system is synthesized as a black box. It is expected to see a critical warning saying that the tool could not resolve a non-primitive black box cell. Each RM is synthesized in a separate project using the following synthesis options:

   (a) "-BUFG = 0", since the Reconfigurable Partition (RP) shouldn't contain any buffers.

   (b) "-mode out_of_context" to ensure that this synthesized block will be a part of a bigger design which is the static design in case of DPR.

   This step is applied on each RM that will occupy the RP. In case of SDR, this step is applied 5 times (2G, 3G, LTE, Wi-Fi, and Bluetooth) on each partition.

3. **Step 3: Top level block design creation**
   After creating the black box for each partition, blocks are imported in the top level project and are connected to the DMAs and interfaces mentioned earlier. Connections are shown in Figure 5.5.

4. **Step 4: PRC configuration**
   The PRC needs to be configured to store the information about the number of RPs and RMs in each partition. As illustrated in Figure 5.6, the number of virtual sockets represents the number of partitions. The number of RMs is determined by the number of chains that share this partition.

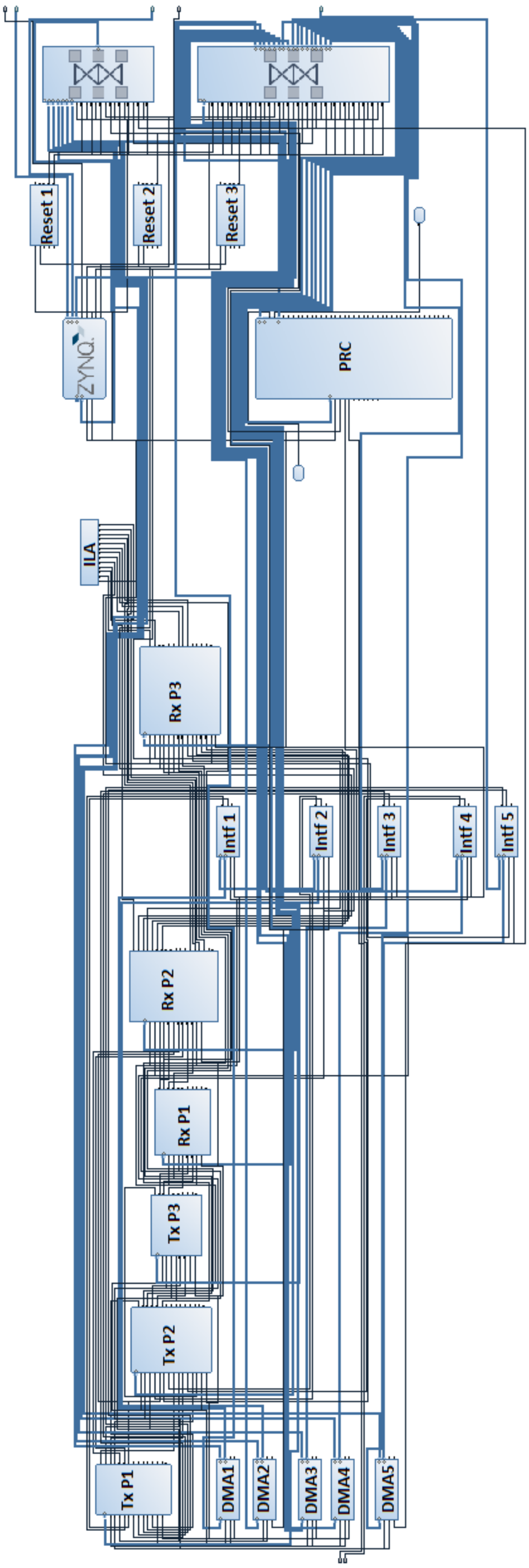
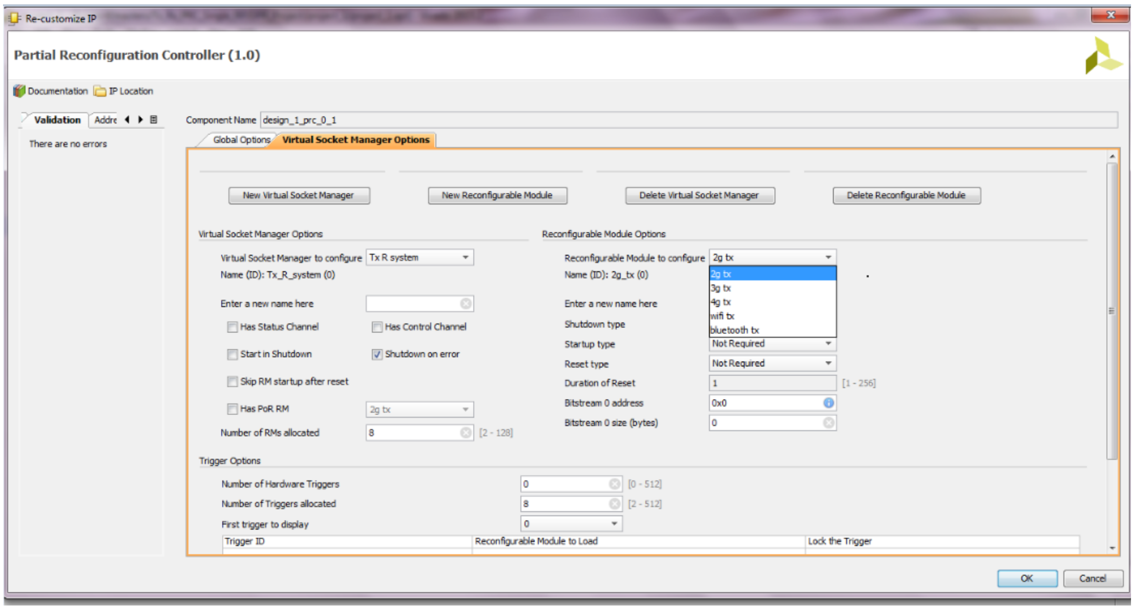


Figure 5.5: Block design connections


Re-customize IP
Documentation
IP Location
Validation
Addre
Global Options
Virtual Socket Manager Options
New Virtual Socket Manager
New Reconfigurable Module
Delete Virtual Socket Manager
Delete Reconfigurable Module
OK
Cancel

Partial Reconfiguration Controller (1.0)

There are no errors

Component Name design_1_prc_0_1

Virtual Socket Manager Options

Virtual Socket Manager to configure Tx R system
Name (ID): Tx_R_system (0)
Enter a new name here
☐ Has Status Channel
☐ Has Control Channel
☐ Start in Shutdown
☑ Shutdown on error
☐ Skip RM startup after reset
☐ Has PoR RM 2g tx
Number of RMs allocated 8 [2 - 128]

Reconfigurable Module Options

Reconfigurable Module to configure 2g tx
Name (ID): 2g_tx (0)
2g tx
3g tx
4g tx
wifi tx
bluetooth tx
Enter a new name here
Shutdown type
Startup type Not Required
Reset type Not Required
Duration of Reset 1 [1 - 256]
Bitstream 0 address 0x0
Bitstream 0 size (bytes) 0

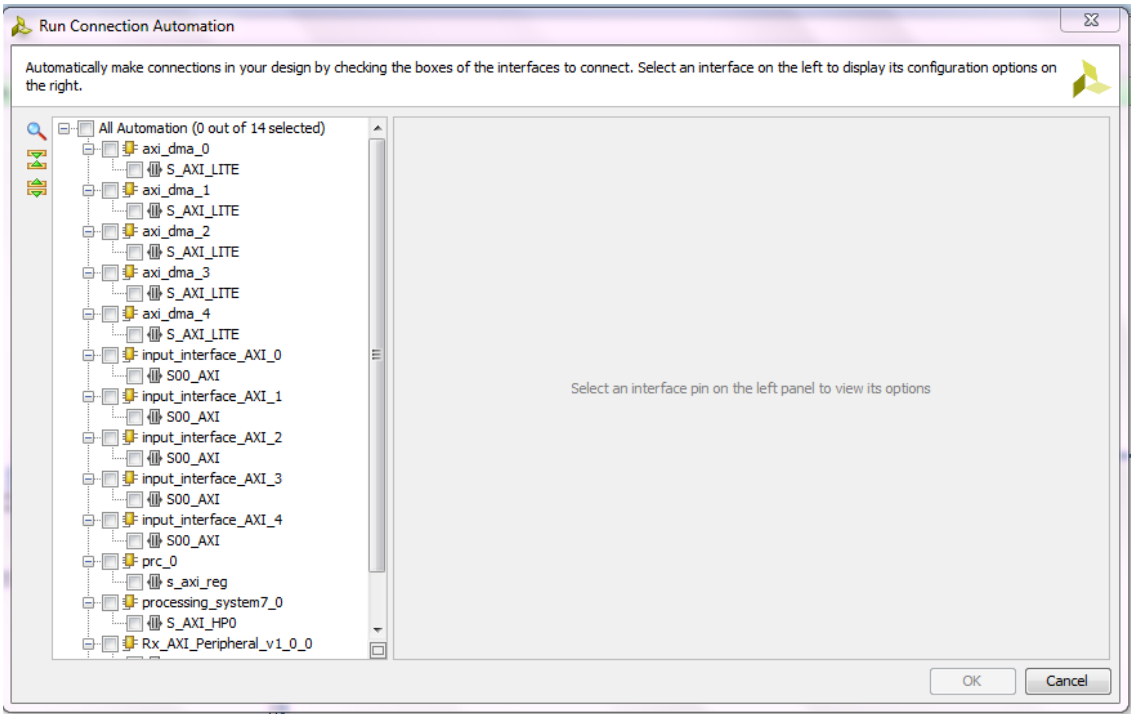
Trigger Options

Number of Hardware Triggers 0 [0 - 512]
Number of Triggers allocated 8 [2 - 512]
First trigger to display 0

Trigger ID	Reconfigurable Module to Load	Lock the Trigger

Figure 5.6: PRC configuration

5. **Step 5: Running connection automation**
   Since every transceiver is operating on a specific generated clock by the ARM processor, connection automation phase is not a simple process. Figure 5.7 shows the options provided by Xilinx Vivado for connecting each block in the whole design to the appropriate clock domain. The rules of the clock distribution network mentioned in Section 5.1 are used here as well.



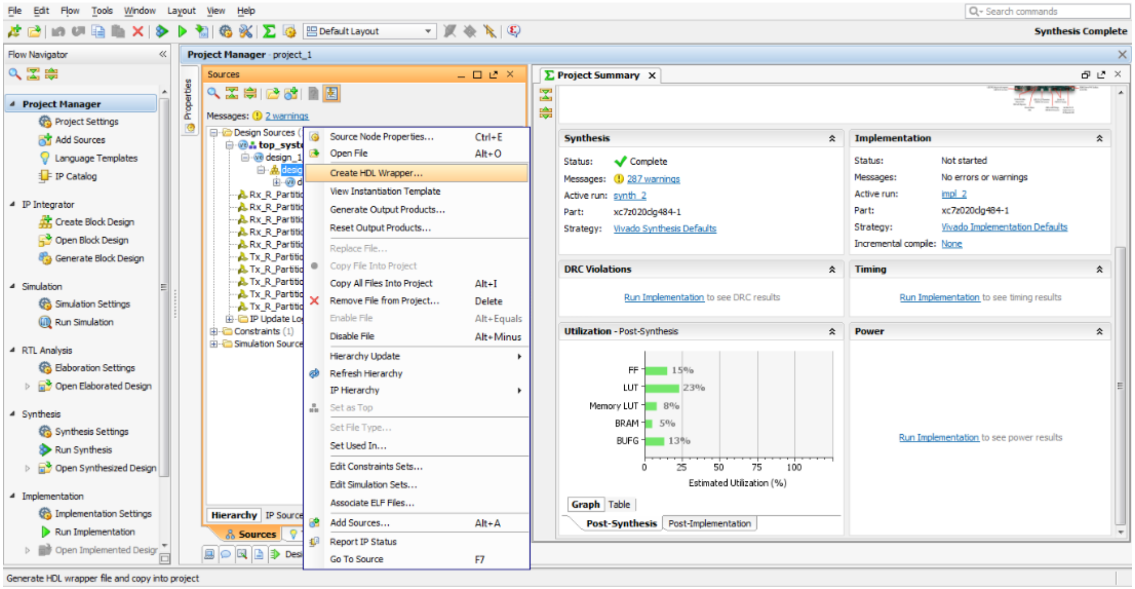
Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

☐ All Automation (0 out of 14 selected)
  ☐ axi_dma_0
    ☐ S_AXI_LITE
  ☐ axi_dma_1
    ☐ S_AXI_LITE
  ☐ axi_dma_2
    ☐ S_AXI_LITE
  ☐ axi_dma_3
    ☐ S_AXI_LITE
  ☐ axi_dma_4
    ☐ S_AXI_LITE
  ☐ input_interface_AXI_0
    ☐ S00_AXI
  ☐ input_interface_AXI_1
    ☐ S00_AXI
  ☐ input_interface_AXI_2
    ☐ S00_AXI
  ☐ input_interface_AXI_3
    ☐ S00_AXI
  ☐ input_interface_AXI_4
    ☐ S00_AXI
  ☐ prc_0
    ☐ s_axi_reg
  ☐ processing_system7_0
    ☐ S_AXI_HP0
  ☐ Rx_AXI_Peripheral_v1_0_0

Select an interface pin on the left panel to view its options

Figure 5.7: Connection automation options

6. **Step 6: Synthesizing the top level**
   After connecting all blocks in the block design, an HDL top level wrapper must be created in order to synthesis the design. As shown in Figure 5.8, an HDL wrapper is created, then synthesis command is executed.



Synthesis Complete

Messages: 2 warnings

Synthesis
Status: Complete
Messages: 287 warnings
Active run: synth_2
Part: xc7z020clg484-1
Strategy: Vivado Synthesis Defaults

Implementation
Status: Not started
Messages: No errors or warnings
Active run: impl_2
Part: xc7z020clg484-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

Run Implementation to see DRC results

Run Implementation to see timing results

Run Implementation to see power results

Generate HDL wrapper file and copy into project

Figure 5.8: HDL wrapper creation

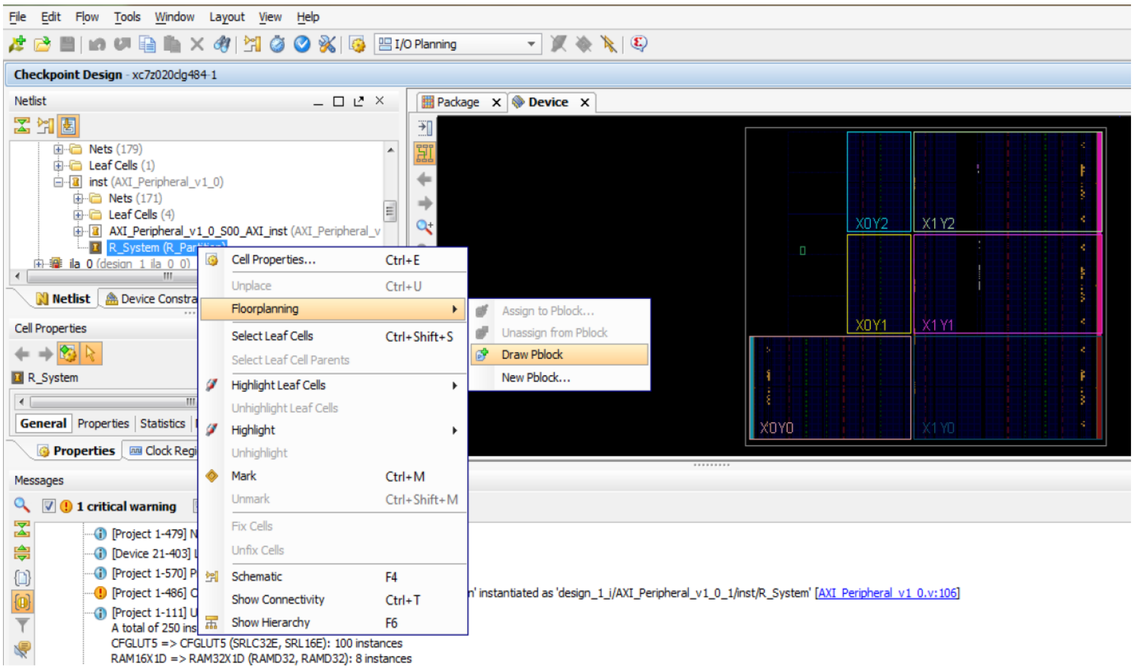
7. **Step 7: Create physical constraints (Pblocks) defining the reconfigurable regions**
   The static design check point generated in Step 2 is opened for the RP floor-planning as shown in Figure 5.9. The Pblock is selected for each partition in a way where the available resources cover the resources needed by each RM. Placement of partitions must be in a specific distribution in order to bypass the placer and router rules.

8. **Step 8: Setting HD RECONFIGURABLE property on each black box**
   This step ensures that each black box in the design will be a reconfigurable partition.

9. **Step 9: Setting RESET_AFTER_RECONFIG property on each RP**
   Each clock region in the FPGA has its own reset pin. In order to ensure that configuration of the new image is performed safely without any trails from the old one, using the reset after reconfiguration feature is recommended.

10. **Step 10: Setting SNAPPING_MODE property on each RP**
    Since the RP must be multiples of the size of the Reconfigurable Frame (RF) defined in [55]. Snapping mode feature is required in order to avoid errors in the consequent steps.

11. **Step 11: Implementing the full design in context**
    After selecting the floor plan of each RP, the synthesized design check point of any standard such as LTE is loaded. The Pblock properties must be checked in order to guarantee that all required resources are available as shown in Figure 5.10. Finally,



```
[Project 1-479] N
[Device 21-403] L
[Project 1-570] P
[Project 1-486] C                                        n' instantiated as 'design_1_i/AXI_Peripheral_v1_0_1/inst/R_System' [AXI_Peripheral_v1_0.v:106]
[Project 1-111] U
A total of 250 ins
CFGLUT5 => CFGLUT5 (SRLC32E, SRL16E): 100 instances
RAM16X1D => RAM32X1D (RAMD32, RAMD32): 8 instances
```

Figure 5.9: Pblock selection

implement, place, and route the design then, save it in a new design check point. The implemented design is used in the generation of bit stream files.

12. **Step 12: Removing RMs from design and saving design checkpoint**
    The LTE block shall be removed from the Pblock to make it black box again using "update_design" command in order to implement other standards.

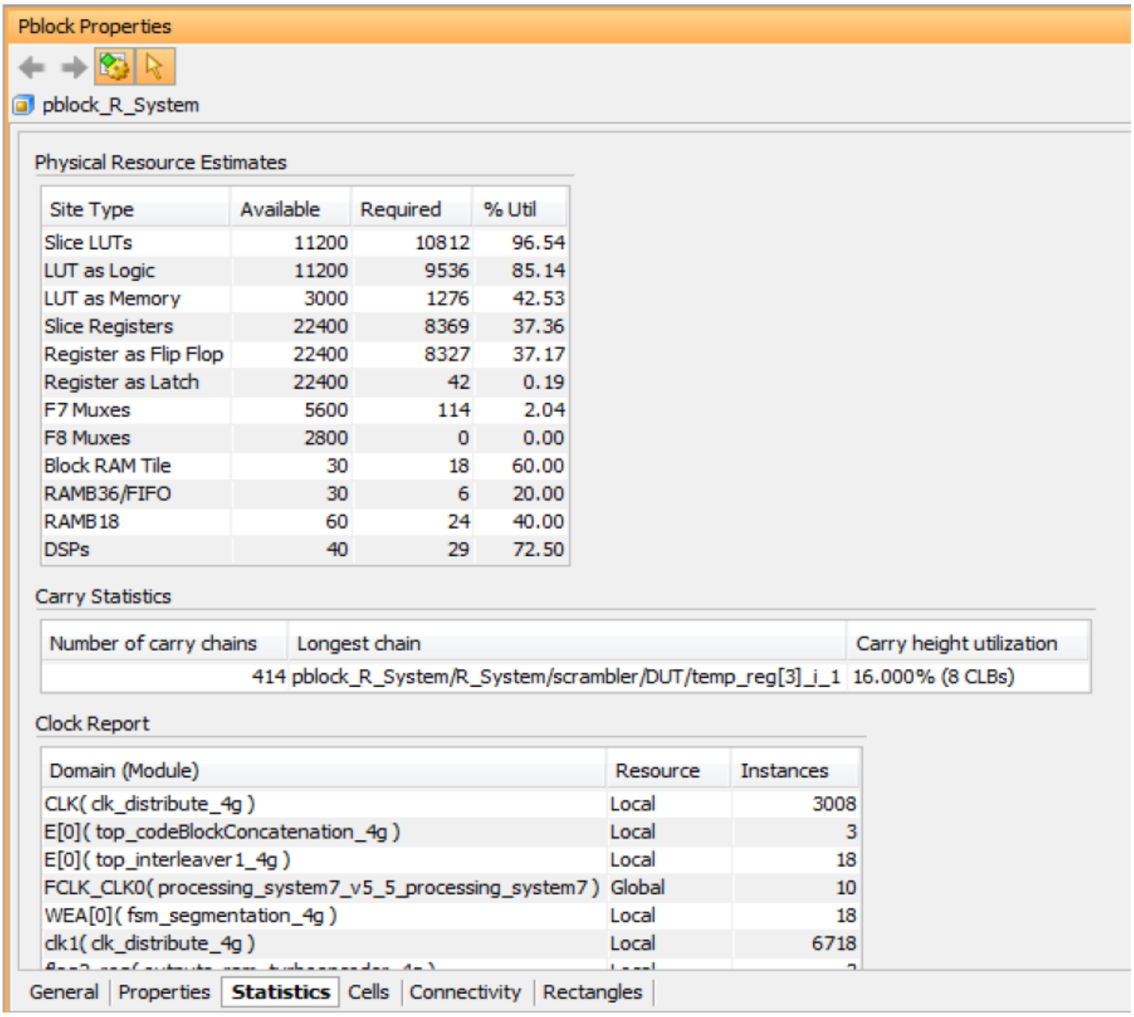
13. **Step 13: Locking static placement and routing**
    This step is used to save the placement and routing of the static parts in the design. Since the consequent steps will change the implementation of the RPs.

14. **Step 14: Repeating Steps 7,8, and 9 till all RMs are implemented**
    The cell must be updated to be a black box again. Then, the remaining chains are implemented.

15. **Step 15: Running pr_verify utility on all configurations**
    This step verifies that all implemented design check points are compatible with Step 5, 8, and 9. This step must pass in order to complete the flow.

16. **Step 16: Creating bit streams for each configuration**
    Full and partial bit stream files are created for each standard in order to program the FPGA. The partial bit files extension is (.bin) if loading is performed through the SD-card, and (.bit) if loading is performed using the JTAG cable.

17. **Step 17: Exporting hardware for software preparation**
    Since the software code is loaded bare metal without using an operating system, the implemented hardware must be transformed into memory addresses where the launcher can execute. Xilinx Vivado assigns each hardware block a mapping to a



Physical Resource Estimates

Site Type	Available	Required	% Util
Slice LUTs	11200	10812	96.54
LUT as Logic	11200	9536	85.14
LUT as Memory	3000	1276	42.53
Slice Registers	22400	8369	37.36
Register as Flip Flop	22400	8327	37.17
Register as Latch	22400	42	0.19
F7 Muxes	5600	114	2.04
F8 Muxes	2800	0	0.00
Block RAM Tile	30	18	60.00
RAMB36/FIFO	30	6	20.00
RAMB18	60	24	40.00
DSPs	40	29	72.50

Carry Statistics

Number of carry chains	Longest chain	Carry height utilization
414	pblock_R_System/R_System/scrambler/DUT/temp_reg[3]_i_1	16.000% (8 CLBs)

Clock Report

Domain (Module)	Resource	Instances
CLK(clk_distribute_4g)	Local	3008
E[0](top_codeBlockConcatenation_4g)	Local	3
E[0](top_interleaver1_4g)	Local	18
FCLK_CLK0(processing_system7_v5_5_processing_system7)	Global	10
WEA[0](fsm_segmentation_4g)	Local	18
clk1(clk_distribute_4g)	Local	6718

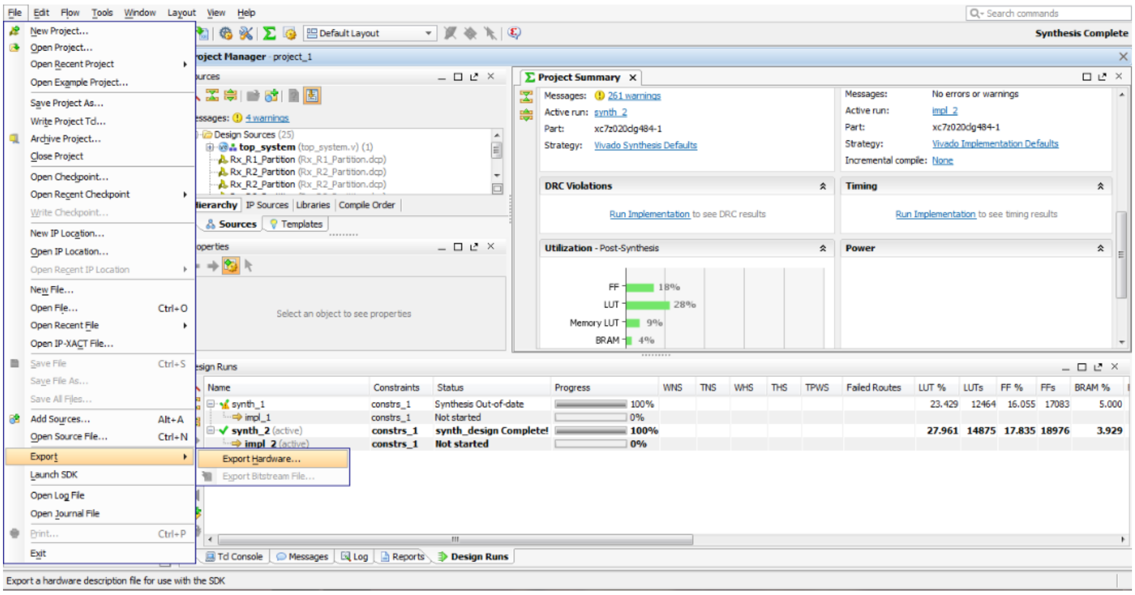
Figure 5.10: Pblock properties

certain memory address. Exporting the hardware step shown in Figure 5.11 dumps all these addresses into header files that will be included in the main function later.

18. **Step 18: Launching Xilinx SDK**
Xilinx SDK is used to launch the software program mentioned earlier to test the system. First a new project must be created with type *FatFileSystem*. Then, the reconfiguration algorithm is included in the project source files to be executed. Project creation is performed as illustrated in Figure 5.12. The C-program must include the mapped addresses of the whole design including the PRC. The PRC addresses are extracted from a text file created by Xilinx Vivado as mentioned in [34].

19. **Step 19: Setting up the UART terminal connection**
Interaction between the board and the PC is performed using a UART cable. The UART terminal must be configured as serial connection with baud rate equals to 125000 bits per second.

20. **Step 20: Running the software program on the processor**
Ultimately, the GDB debugger is used to launch the ELF file generated by the SDK



Messages: 261 warnings
Active run: synth_2
Part: xc7z020clg484-1
Strategy: Vivado Synthesis Defaults

Messages: No errors or warnings
Active run: impl_2
Part: xc7z020clg484-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

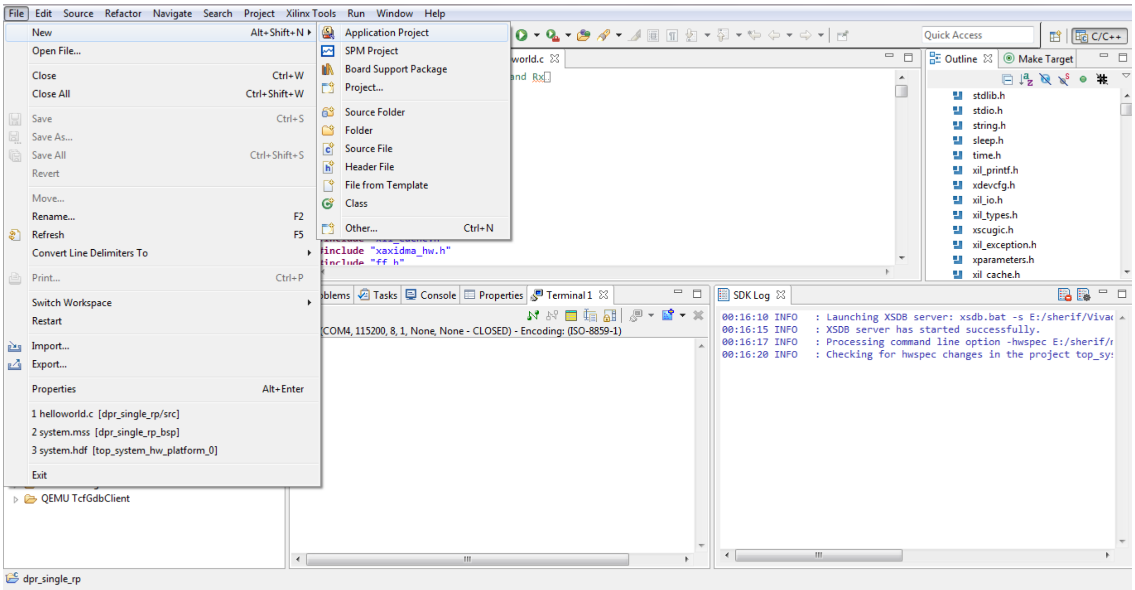
Run Implementation to see DRC results

Run Implementation to see timing results

Name	Constraints	Status	Progress	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT %	LUTs	FF %	FFs	BRAM %
synth_1	constrs_1	Synthesis Out-of-date	100%							23.429	12464	16.055	17083	5.000
impl_1	constrs_1	Not started	0%											
synth_2 (active)	**constrs_1**	**synth_design Complete!**	**100%**							**27.961**	**14875**	**17.835**	**18976**	**3.929**
impl_2 (active)	**constrs_1**	**Not started**	**0%**											

Export a hardware description file for use with the SDK

Figure 5.11: Exporting Hardware



```
#include "xaxidma_hw.h"
```

COM4, 115200, 8, 1, None, None - CLOSED) - Encoding: (ISO-8859-1)

```
00:16:10 INFO  : Launching XSDB server: xsdb.bat -s E:/sherif/Vivac
00:16:15 INFO  : XSDB server has started successfully.
00:16:17 INFO  : Processing command line option -hwspec E:/sherif/r
00:16:20 INFO  : Checking for hwspec changes in the project top_sy:
```

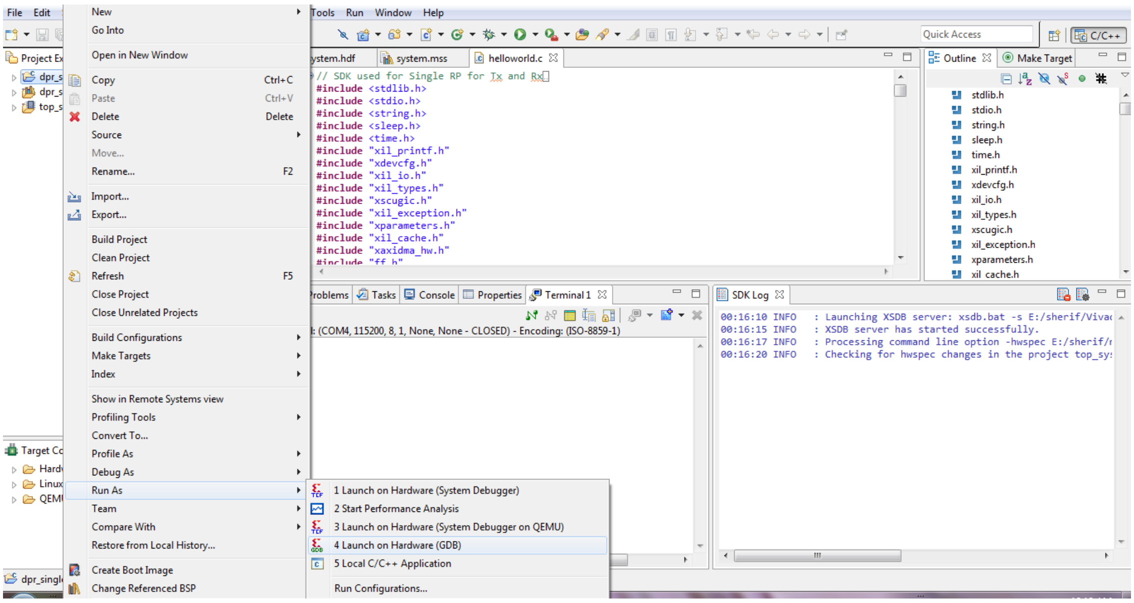
Figure 5.12: SDK project creation

compiler on the ARM processor. The menu containing user options is displayed on the UART terminal. Steps for launching the GDB debugger are provided in Figure 5.13.

5.4 DPR Proposed Approaches

DPR system migration, while keeping the same test environment, is performed through replacing the five systems (2G, 3G, LTE, Wi-Fi, BT) by one block which will have:

1. Multiplexer to pass the input data according the chosen system.



```
// SDK used for Single RP for Tx and Rx
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sleep.h>
#include <time.h>
#include "xil_printf.h"
#include "xdevcfg.h"
#include "xil_io.h"
#include "xil_types.h"
#include "xscugic.h"
#include "xil_exception.h"
#include "xparameters.h"
#include "xil_cache.h"
#include "xaxidma_hw.h"
#include "ff.h"
```

```
l: (COM4, 115200, 8, 1, None, None - CLOSED) - Encoding: (ISO-8859-1)
```

```
00:16:10 INFO   : Launching XSDB server: xsdb.bat -s E:/sherif/Vivac
00:16:15 INFO   : XSDB server has started successfully.
00:16:17 INFO   : Processing command line option -hwspec E:/sherif/r
00:16:20 INFO   : Checking for hwspec changes in the project top_sys
```

Figure 5.13: Software program execution

2. The physical layer implementation of the communication standard transceiver.

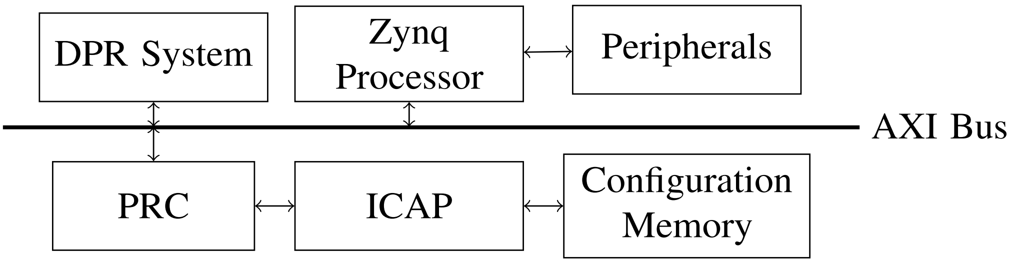


Figure 5.14: DPR overall system

According to the DPR techniques discussed in [52, 53], Xilinx PRC is used due to its high throughput that is close to the ideal throughput of 400 MB/sec using ICAP to communicate with the FPGA configuration memory. Figure 5.14 shows an overview of the overall system.

The three metrics that measure the effectiveness of deploying the SDR system using the DPR flow are: area, power, and switching time. The switching time is the bottle neck of the technique since the system should switch rapidly between chains in order to achieve performance similar to the case of no DPR.

The size of the bit stream file is dependent on the allocated design area. The switching time of the RP is dependent on the size of the partial bit file. Thus, area optimization is one of the challenges to achieve small switching time. System optimization is achieved through modifying the RTL code to reduce the utilized area. Reduction is achieved by the synthesis techniques mentioned in Section 3.1 [40].

Since the maximum clock frequency of ICAP is 100 MHz, and the width of the data bus is 32-bits, the ideal switching time is calculated using the following equation [23]:

$$T = \frac{Partial\ Bit\ File\ Size\ in\ Bits}{Bus\ Width \times Max\ Clock\ Frequency} \tag{5.1}$$

5.4.1 Single Partition Approach

In real life, wireless transmitter and receiver communicate remotely. The simplest approach in partitioning is choosing a single RP for the transmitter and another one for the receiver as shown in Figure 5.15.

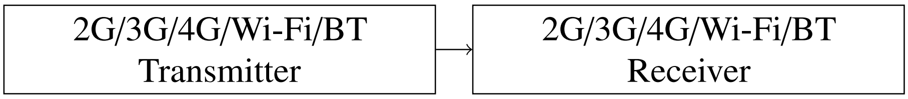


Figure 5.15: Single partition approach block diagram

Since the LTE is the most complex transceiver chain, it has the largest allocated area. Therefore, the size of the two partitions is selected to fit the LTE transceiver. Figure 5.16 shows the floor planning of the allocated RPs on the FPGA.

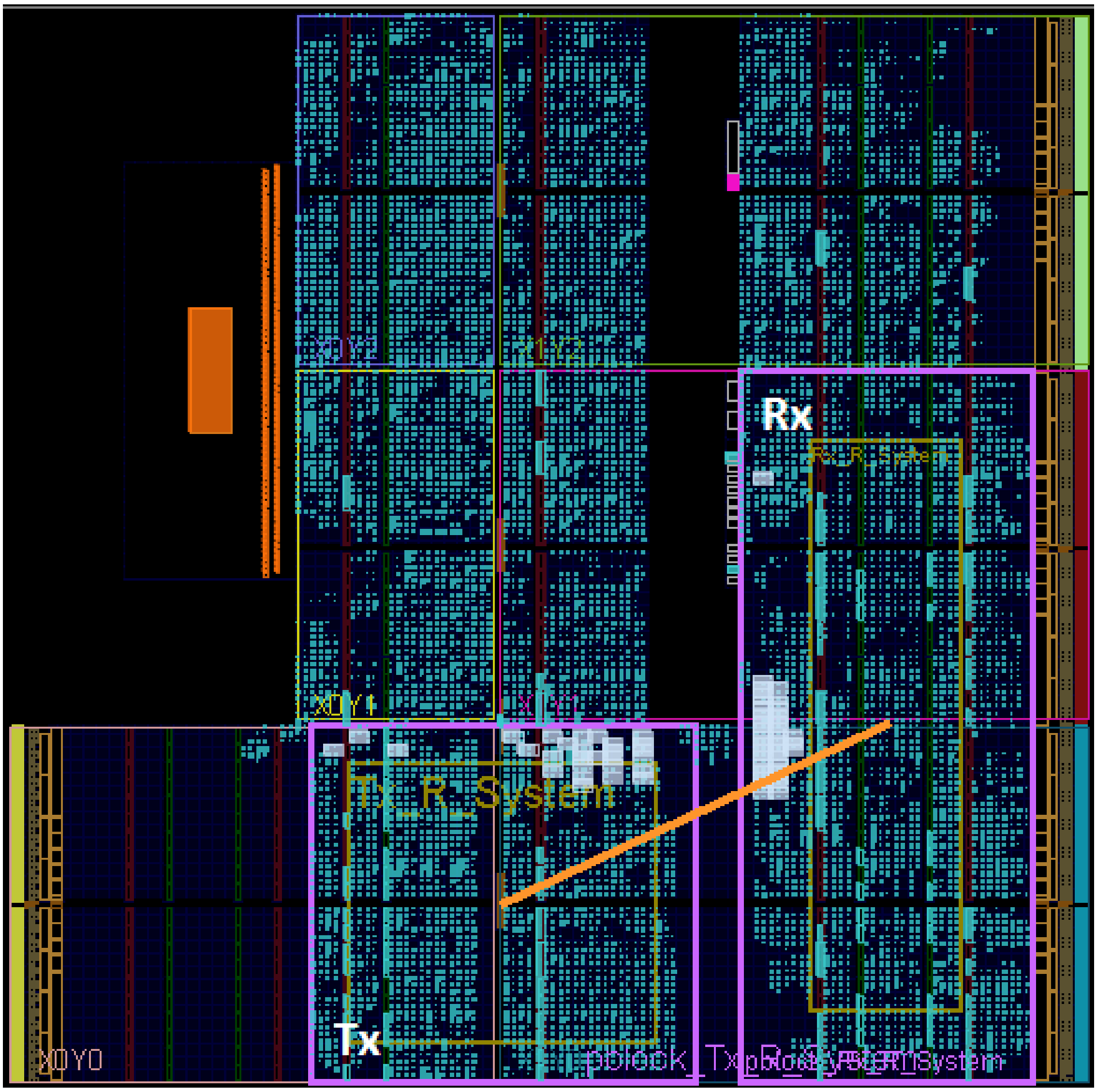


Figure 5.16: Single partition approach floor plan

5.4.2 Multi-Partition Approach

Although the single partition approach is the best fit for LTE, simple chains such as 2G and Bluetooth are constrained with the large allocated unwanted area. This leads to increase the power consumption and the switching time [55].

In order to solve this dilemma, another approach is proposed. The new technique suggests splitting both transmitter and receiver partitions into smaller partitions, in order to fit the area of the small chains. Meanwhile, there are some constrains should be taken in consideration in order to minimize the wasted area leading to the increase of switching time and power consumption:

1. Sizes of the partitions must be relatively multiples of the minimum RF size. As mentioned in [30] the minimum RF area is (400 LUTs, 10 DSPs, or 10 BRAMs).
2. The difference between areas of chosen blocks in each chain to be merged together must be small.
3. Since the FPGA resources (BRAMs and DSPs) are distributed as shown in Figure 5.17, partitions are placed in a certain way not only to fit the required area, but also to obey the rules of placement and routing.
4. Partitions must not be placed vertically in the same clock region. It is mandatory to reset the whole partition after reconfiguration. Every clock region has its own reset pin.
5. Finally, as the number of partitions increase, the PRC overhead time becomes much more significant.

A MATLAB code is developed to satisfy the constraints mentioned earlier. The algorithm main aim is to find which blocks shall be merged together in each chain in order to obtain the minimum power consumption and switching time. The procedures shown in Figure 5.18 are being taken:

1. Calculate the weighted sum of all blocks in LTE chain that could be merged together. Iterations are done on the LTE chain specifically, since it has the largest number of blocks. The weighted sum is calculated by the following equation [30]:

$$\sum A_{LUTs} = N_{LUTs} + 40 \times N_{BRAMs} + 40 \times N_{DSPs} \tag{5.2}$$

2. Compare the weighted sum of each block in the LTE chain with the weighted sum of 3G blocks, then choose the blocks that will be merged together in each chain based on two aspects: the difference in the area must be the optimum; and the area of the largest partition should be nearly multiples of the minimum RP area (400 LUTs, 10 BRAMs, or 10 DSPs).
3. The merged LTE blocks are excluded and the operation is repeated again on Wi-Fi blocks, then 2G, and finally the Bluetooth.
4. Finally, the remaining LTE blocks are merged together.

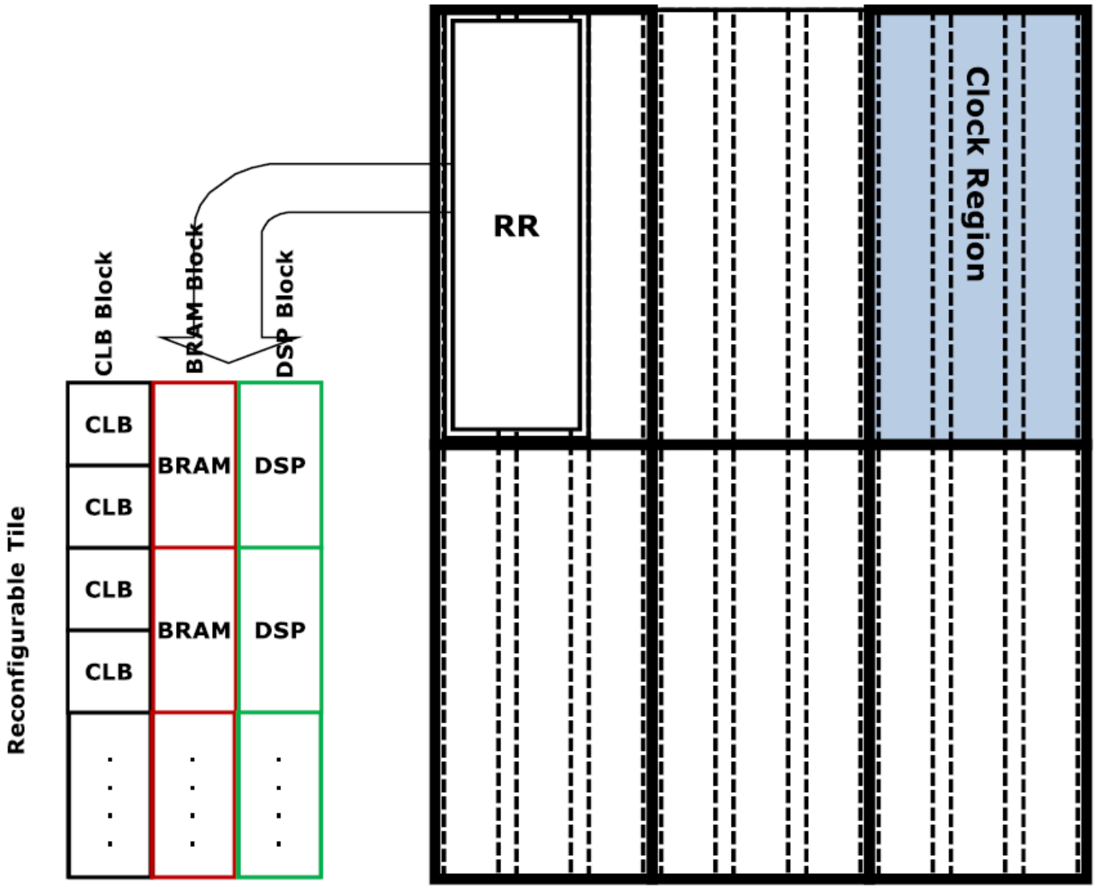


Figure 5.17: Virtex-7 RP physical constraints

It worth to mention that the transmitters and receivers in all chains except the LTE are considered as two large non-dividable blocks while using the algorithm in order to minimize the overhead of the PRC as much as possible.

The algorithm suggests dividing the LTE transmitter and receiver each into 3 sub-blocks. As shown in Figure 5.19, the whole 2G and Bluetooth transmitters are nominated to be merged with LTE CRC, segmentation, and encoder. Meanwhile the Wi-Fi and 3G transmitters shall be merged with the rest of the blocks in the second partition, except the SC-FDMA which is left alone in the third partition.

At the receiver side, the 2G and Bluetooth are merged with the LTE rate dematching. The 3G and Wi-Fi are nominated to be merged with LTE decoder, desegmentation, and de-CRC. The rest are suggested to occupy the third partition alone.

Figure 5.20 shows the floor planing of the 6 partitions on the FPGA. Placement of all RPs is done in such shape in order to obey the placer rules and easily route the design.

5.5 Simulation Results

5.5.1 Fixation Error

Wi-Fi and LTE chains are OFDM based. They are implemented using the FFT block that represents a source of errors due to the fixed point representation. There is a trade off

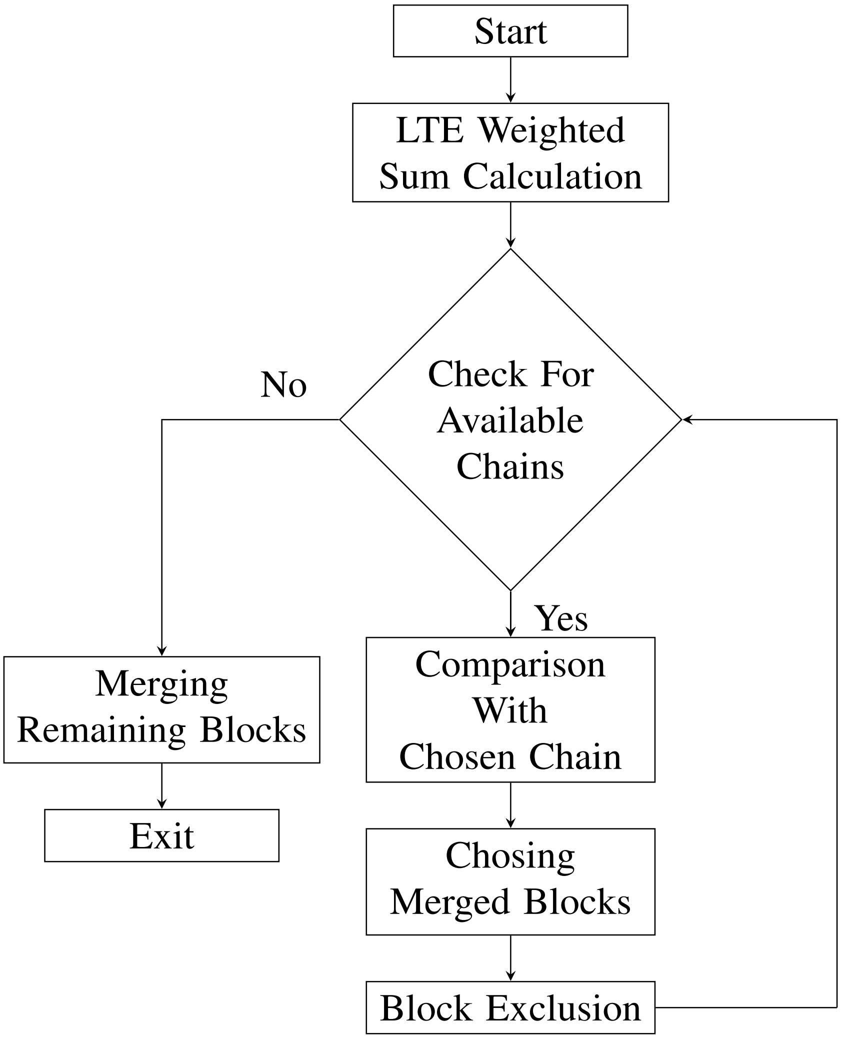

Start
LTE Weighted Sum Calculation
Check For Available Chains
No
Yes
Merging Remaining Blocks
Exit
Comparison With Chosen Chain
Chosing Merged Blocks
Block Exclusion

Figure 5.18: Partitioning algorithm flow chart

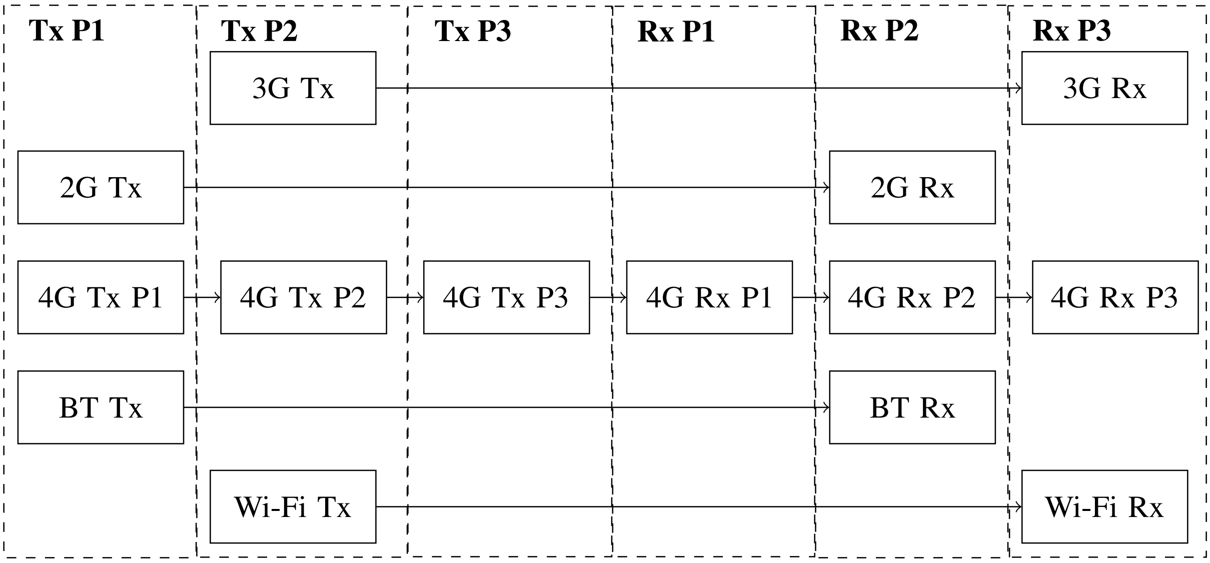


Figure 5.19: Multi-partition approach block diagram

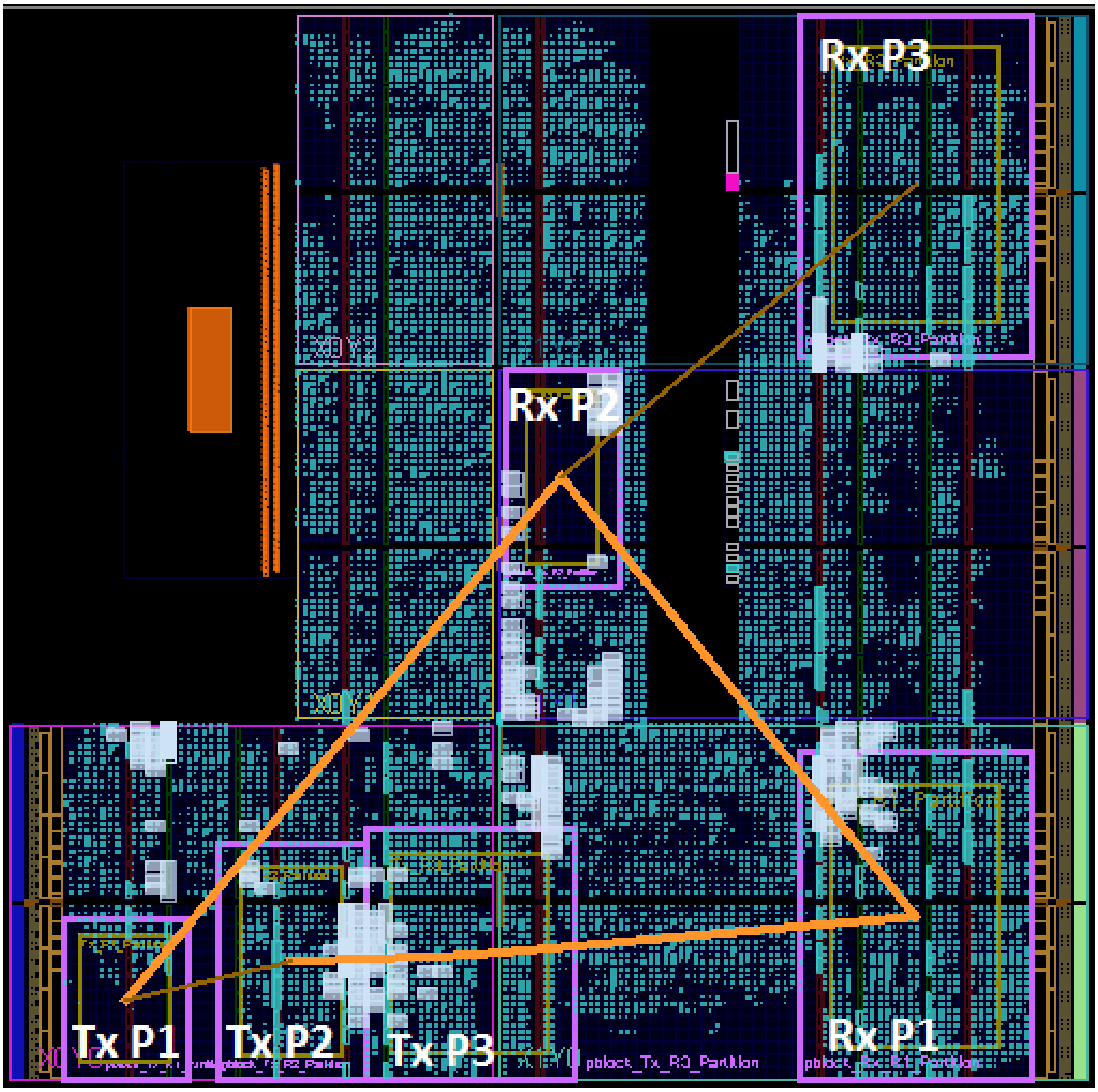


Figure 5.20: Multi-partition approach floor plan

between the system accuracy and the size of memories storing the symbols.

The IFFT block in Wi-Fi chain does complex mathematical operations which result in some errors due to the fixed point representation of the mapped symbols. The chosen bit representation shown in Figure 5.21 is 3 bits for the decimal part and 9 bits for the fraction. The demapper is able to fix the accumulated errors produced from the IFFT and FFT. It worth to mention that the decoder only gets rid of the noise errors. System average error calculations are performed using the following equation:

$$Error = \frac{|\sum_{i=1}^{N} O_{float}(i) - O_{fixed}(i)|}{N} \tag{5.3}$$

The variables O_{float} and O_{fixed} are the system output symbols in case of floating and fixed point respectively. The calculated receiver error is -64.3 dB. Meanwhile, the whole transceiver accumelated error is -58.6 dB.

The situation is much more complicated for LTE chain. The SC-FDMA consists of two complex blocks: DFT and IFFT that lead to produce error accumulation larger than

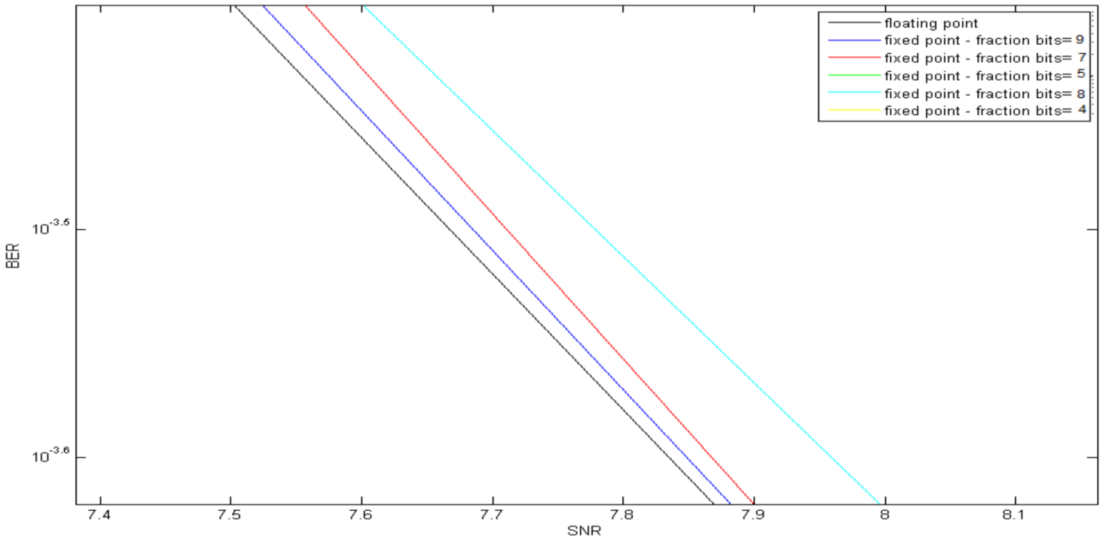


Figure 5.21: Wi-Fi 16-QAM fraction part fixation

the Wi-Fi chain. The DFT output symbols are represented in large decimal numbers. In order to overcome this issue, the number of bits representing the symbol is increased to be 14 bits, where 5 bits are for the decimal part and 9 bits for the fraction. The calculated receiver error is -12.07 dB and the whole transceiver accumulated error is -9.83 dB.

5.5.2 Area and Power Measuremnet

The tables shown below, summarize the utilized area for all transmitter and receiver chains in terms of LUTs, BRAMs, and DSPs.

Table 5.1: Transmitter Chains Area Utilization

Transmitter Chain	LUTs	BRAMs	DSPs
Bluetooth	940	4	0
Wi-Fi	2557	3	7
GSM	637	2	0
UMTS	1734	3.5	2
LTE	5850	12	16

Table 5.2: Receiver Chains Area Utilization

Receiver Chain	LUTs	BRAMs	DSPs
Bluetooth	635	3	0
Wi-Fi	3749	4.5	11
GSM	965	2.5	0
UMTS	3563	17	1
LTE	7659	22	21

In order to measure the effectiveness of the proposed technique, a comparison is performed between the case of no DPR, single RP, and multi-RPs with respect to power, area, and switching time. In order to be able to express the occupied area of each approach in a single number rather than comparing the number of LUTs, BRAMs, and DSPs; Equation 5.2 shall be used here as well.

Figure 5.22 shows that the system total area is reduced by 10.19% in case of single RP. The multi-RP approach decreases the power consumption for 2G and Bluetooth by 98.58%, 3G and Wi-Fi by 80.59%, and LTE by 50.81% compared to the case of no DPR. The increase in the area of LTE is due to partitioning the design on multiple RPs.

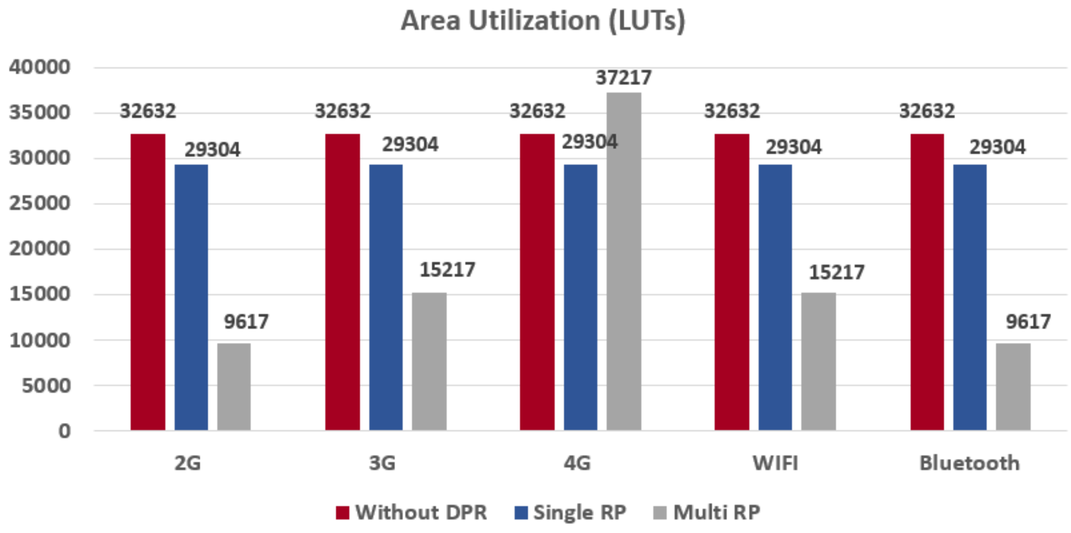


Figure 5.22: Area utilization (LUTs)

The sizes of the partial bit files of the transmitter and the receiver are 523KB and 837KB respectively. Measuring the switching time is done in the C-code with the aid of timers for time calculation. The actual switching time for all chains calculated from Equation 5.2 is 3.49 ms, which is close to the theoretical value 3.47 ms calculated from adding the calculated switching time of both transmitter and receiver.

The sizes of partial bit files in case of multi-RP are 248, 324, and 306 KBs for the transmitter and 407, 252, and 390 KBs for the receiver. Table 5.3 compares the theoretical calculated switching time with the measured actual time for all transceiver chains.

Table 5.3: Switching Time

Standard	Theoretical Time (ms)	Actual Time (ms)
2G and Bluetooth	1.27	1.29
3G and Wi-Fi	1.82	1.83
LTE	4.93	4.95

A sacrifice is done with the switching time of LTE chain to save the rest of the chains. Figure 5.23 shows that the multi-partition approach decreases the switching time in all chains except the LTE compared with the switching time in case of the single RP. The reasons behind the large switching time in case of LTE are:

1. Configuring 6 partitions serially rather than 2, accumulates the configuration time of each partition while calculating the total time.

2. Increasing the number of partitions makes the PRC overhead significant.

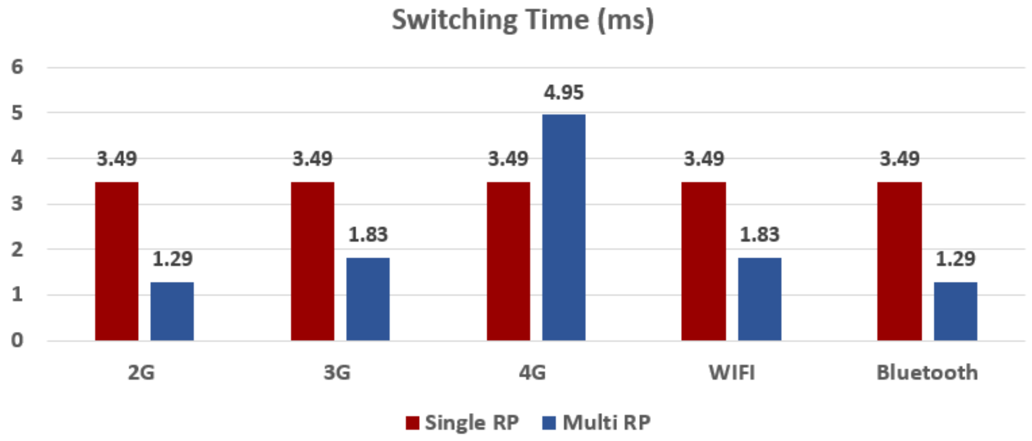


Figure 5.23: Swithcing time (ms)

Estimated power calculations listed in Figure 5.24 show that the single partition approach decreases the power by 76.71% compared with the case of no DPR. The multi-partiton approach decreases the power consumption for 2G and Bluetooth by 95.43%, 3G and Wi-Fi by 79.69%, and LTE by 59.09% compared to the case of no DPR.

Table 5.4 shows the percentages of decrease/increase in system utilized area and power consumption using the multi-partition approach compared to the case of no DPR.

5.6 Summary

The following chapter illustrates the established test environment to verify the DPR technique. The DPR flow steps are provided. An illustration of two DPR approaches is

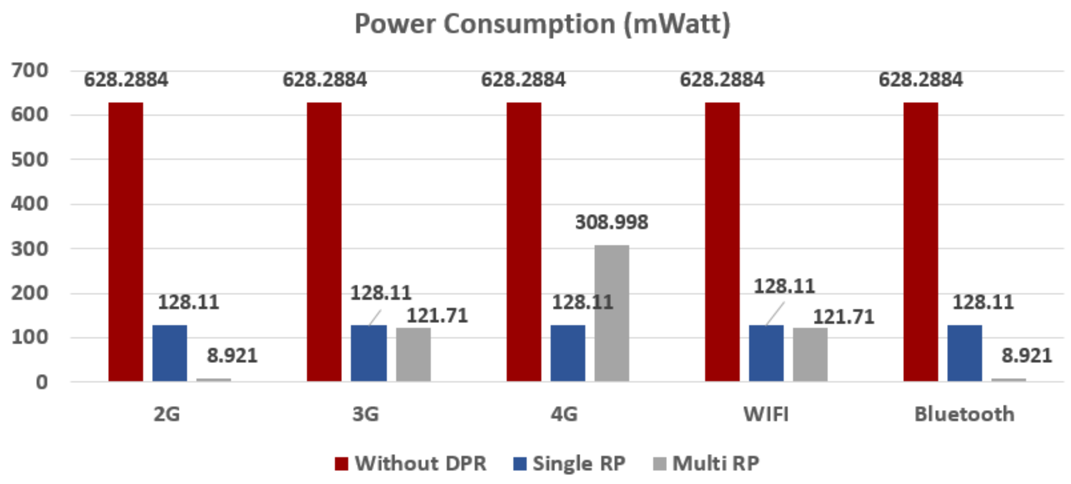


Figure 5.24: Average power (mW)

Table 5.4: Area and Power Comparison

Standard	Area	Power
2G and Bluetooth	70.52% ⇊	95.43% ⇊
3G and Wi-Fi	53.36% ⇊	79.69% ⇊
LTE	12.65% ⇈	59.09% ⇊

provided. Ultimately, the chapter shows simulation results of both approches compared with the current implemented transceivers in mobile phones.

Chapter 6: Conclusion And Proposed Future Work

6.1 Conclusion

In this research work, five communicattion standards: Bluetooth, Wi-Fi, 2G, 3G, and LTE are being deployed. The five transceivers are fully implemented in order to prove the concept of SDR. The DPR technique is used to switch between different communication standards. A test environment is deployed to verify the corrrectness of the system while using DPR. Two partioning approaches are being developed to minimize area, power, and switching time.

DPR technique shows its effectiveness in saving the allocated area and the power consumption compared to the current transceivers in the mobile devices with a reasonable switching time overhead. The single partition approach reduces the system total area and power by 10.19% and 76.71% respectively compared with case of no DPR.

The multi partition approach proved its ability to decrease the utilized area and power consumption of 2G and Bluetooth chains by 70.52% and 95.43% with respect to no DPR. The 3G and Wi-Fi area and power decreased by 53.36% and 79.69%. The area of the LTE increased by 12.65% but its power decreased by 59.09%.

6.2 Proposed Future Work

Further actions can be taken as future work including the following:

1. Implementing the data link and MAC layers to achieve system completeness.
2. Sending the data through the air. Since the transmitter and receiver should be communicating through air such that the whole communication system is complete. This requires two FPGAs and two USRPs for the transmitter and receiver implementation.
3. Hardware verification of DPR flow is a challenging area. Developing a generic automated hardware environment for any system using DPR flow is promising.
4. High Level Synthesis (HLS) flow is used to generate RTL codes of hardware designs from C/C++/MATLAB level. Integration of DPR flow with the HLS flow shall enhance and fasten the design process. Xilinx introduces a new tool that is responsible for the HLS flow called SDSOC.

References

[1] E. MARTIN and I. GUSTAFSSON, "Reconfigurable Analog to Digital Converters for Low Power Wireless Applications," Doctoral Thesis, KTH, School of Information and Communication Technology (ICT), Electronic, Computer and Software Systems, May 2008.

[2] A. Sadek, H. Mostafa, and A. Nassar, "Dynamic Channel Coding Reconfiguration in Software Defined Radio," International Conference on Microelectronics (ICM 2015), Casablanca, Morocco, pp. 13-16, Dec. 2015.

[3] J. Delahaye, G. Gogniat, C. Ronald, and P. Bomel, "Software radio and dynamic reconfiguration on a DSP/FPGA platform," 3rd Karlsruhe Workshop on Software Radios, France, pp. 1-9, June 2004.

[4] M. Hentati, A. Nafkha, P. Leray, J. F.Nezan, and M. Abid, "Software Defined Radio Equipment: What's the Best Design Approach to Reduce Power Consumption and Increase Reconfigurability?," International Journal of Computer Applications, vol. 45, no. 14, pp. 26-31, May 2012.

[5] E. Grayver and P. Dafesh, "Multi-modulation programmable transceiver system with turbo coding," in Proc. of the IEEE Aerospace Conference, Big Sky, MT, USA, pp. 1484-1493, March 2005.

[6] E. J. McDonal, "Runtime FPGA Partial Reconfiguration," IEEE Communications Magazine, vol. 33, no. 5, pp. 26-38, May 1995.

[7] S. Shreejith, B. Banarjee, K. Vipin and S. Fahmy, "Dynamic cognitive radios on the Xilinx Zynq hybrid FPGA," in International Conference on Cognitive Radio Oriented Wireless Networks, vol. 156, pp. 427-437, Oct. 2015.

[8] A. Sadek, H. Mostafa, A. Nassar, and Y. Ismail "Towards the implementation of Multi-band Multi-standard Software-Defined Radio using Dynamic Partial Reconfiguration," International Journal of Communication system, pp. 1-12, June 2017.

[9] R. Kumar, R. C. Joshi, and K. S. Raju, "A FPGA partial reconfiguration design approach for RASIP SDR," *IEEE India Conference (INDICON 2009)*, Gujarat, India, pp. 1-4, Dec. 2009.

[10] J. Delahaye, J. Palicot, and C. Moy, "Partial Reconfiguration of FPGAs for Dynamical Reconfiguration of a Software Radio Platform," Mobile and Wireless Communications Summit, Budapest, Hungary, pp. 1-5, July 2007.

[11] W. Lie and W. Feng-yan, “Dynamic partial reconfiguration in FPGAs,” Third International Symposium on Intelligent Information Technology Application, Shanghai, China, pp. 253-257, Nov. 2009.

[12] X. Di, S. Fazhuang, D. Zhantao, and H. Wei, “A Design Flow for FPGA Partial Dynamic Reconfiguration,” Second International Conference on Instrumentation, Measurement, Computer, Communication and Control, Harbin, China, pp. 119-123, Dec. 2012.

[13] T. Ulversoy, “Software Defined Radio: Challenges and Opportunities,” *IEEE Communications Surveys & Tutorials*, Vol. 12 , no. 4, pp. 31-124, May 2010.

[14] J. Mitola and G. Q. Maguire, “Cognitive radio: making software radios more personal,” in IEEE Personal Communications, vol. 6, no. 4, pp. 13-18, Aug 1999.

[15] A. Sadek, H. Mostafa, and A. Nassar “On the Use of Dynamic Partial Reconfiguration for Multi-band/Multi-standard Software Defined Radio,” *IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2015)*, Cairo, Egypt, pp. 498-499, Dec. 2015.

[16] M. L. Ferreira, A. Barahimi, and J. C.Ferreira “Dynamically Reconfigurable LTE-compliant OFDM Modulator for Downlink Transmission,” Design of Circuits and Integrated Systems (DCIS) Granada, Spain, pp. 1-6, Nov. 2016.

[17] M. Tran, E. Casseau, and M. Gautier, “Demo abstract: FPGA-based implementation of a flexible FFT dedicated to LTE standard,” Design and Architectures for Signal and Image Processing (DASIP), Rennes, France, pp. 241-242, Oct. 2016.

[18] K. A. Arun Kumar, “FPGA implementation of QAM modems using PR for reconfigurable wireless radios,” International Conference on Microelectronics, Communications and Renewable Energy, India, pp. 1-6, June 2013.

[19] K. A.Kumar “A low power implementation of PSK modems in FPGA with reconfigurable filter and digital NCO using PR for SDR and CR applications,” International Conference on Green Technologies (ICGT), Trivandrum, India, pp. 192-197, Dec. 2012.

[20] G. Sklivanitis, A. Gannon, and S. N. Batalama, “Addressing next-generation wireless challenges with commercial software-defined radio platforms,” IEEE Communication Magazine, vol. 54, no. 1, pp. 9-35, Jan. 2016.

[21] X. Wu, J. Palicot, and P. Leray, “Cognitive Radio Management Benefiting From Flexible Reconfiguration,” Fourth Conference on Telecommunications and Remote Sensing, pp. 18-21, Sep. 2015.

[22] E. Grayver, “Implementing Software Defined Radio,” Springer Science and Business Media, vol. 34, no. 1, pp. 59-67, Nov. 2013.

[23] Xilinx Inc., “ZC702 Evaluation Board reference manual UG585,” (v1.12.1) Dec. 2017.

[24] V. Kannan, T. A. Abbasi, M. U. Abbasi, and S. Ahmed, “Novel FPGA Based Hardware Realization Of Arbitrary Functions,” Journal of Circuits System and Computerst, vol. 16, no. 1, pp. 895-909, Dec. 2007.

[25] Xilinx Inc., “MicroBlaze Processor Reference Guide UG081,” April 2008.

[26] V. Kanhiroth, C. Parikh, C. Trefftz, and A. Rahman, “Embedded processors on FPGA: Hard-core vs Soft-core,” Masters Thesis, Grand Valley State University, May 2017.

[27] Xilinx Inc., “ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC UG850,” (v1.12.1) Jan. 2018.

[28] Xilinx Inc., “7 Series FPGAs Configurable Logic Block UG474,” (v1.8) Sep. 2017.

[29] D. Koch, J. Torresen, C. Beckhoff, D. Ziener, C. Dennl, V. Breuer, J. Teich, M. Feilen, and W. Stechele, “Partial Reconfiguration on FPGAs: Architectures, Tools and Applications,” in ARCS 2012, Muenchen, Germany, pp. 1-12, Feb. 2012.

[30] Xilinx Inc., “Partial Reconfiguration User Guide UG909,” (v2017.1) April 2017.

[31] Xilinx Inc., “AXI Reference Guide UG761,” (v13.1) March 2011.

[32] Xilinx Inc., “AXI DMA LogiCORE IP Product Guide PG021,” April 2018.

[33] Xilinx Inc., “AXI HWICAP LogiCORE IP Product Guide PG134,” Oct. 2016.

[34] Xilinx Inc., “Partial Reconfiguration Controller PG193,” April 2016.

[35] Bluetooth SIG Inc., “Specification of the Bluetooth System,” (v2.0) Vol. 2, Nov. 2004.

[36] IEEE Computer Society, “IEEE Std 802.11™-2012,” March 2012.

[37] IEEE Computer Society, “IEEE Std 1800™-2012,” Feb. 2013.

[38] IEEE Computer Society, “IEEE Std 1364™-2005,” April 2006.

[39] IEEE Computer Society, “IEEE Std 1076™-2008,” Jan. 2009.

[40] Xilinx Inc., “Synthesis UG901,” (v2016.3) Oct. 2016.

[41] IEEE Computer Society, “IEEE Std 1735™-2014,” Dec. 2014.

[42] 3GPP, “Multiplexing and multiple access on the radio path TS 145 002,” (v9.3.0) April 2010.

[43] 3GPP, “MChannel coding TS 145 003,” (v10.0.0) April 2011.

[44] 3GPP, “Modulation TS 145 004,” (v9.0.0) Feb. 2010.

[45] 3GPP, “Physical channels and mapping of transport channels onto physical channels (FDD) TS 25.211,” (v12.1.0) Dec. 2014.

[46] 3GPP, “Multiplexing and channel coding (FDD) TS 25.212,” (v12.1.0) Dec. 2014.

[47] 3GPP, “Spreading and modulation (FDD) TS 25.213,” (v12.0.0) Sep. 2014.

[48] 3GPP, “Multiplexing and channel coding TS 36.211,” (v12.5.0) Dec. 2014.

[49] 3GPP, “Physical channels and modulation TS 36.212,” (v12.5.0) June 2015.

[50] K. Cholan, “Design and Implementation of Low Power High Speed Viterbi Decoder,” International Conference on Communication Technology and System Design, vol. 30, no. 3, pp. 61-68, Jan. 2012.

[51] M. Raymond and C. Arun, “Design and VLSI Implementation of a High Throughput Turbo Decoder,” International Journal of Computer Applications, vol. 22, no. 3, pp. 34-35, May 2011.

[52] A. K. ELdin, A. Mohamed, A. Nagy, Y. Gamal, A. Shalash, Y. Ismail, and H. Mostafa, “Design Guidelines for the High-Speed Dynamic Partial Reconfiguration Based Software Defined Radio Implementations on Xilinx Zynq FPGA,” *IEEE International Symposium on Circuits and Systems (ISCAS 2017)*, Baltimore, USA, pp. 1-4, May 2017.

[53] R. Tessier, K. Pocek, and A. DeHon, “Reconfigurable Computing Architectures,” in Proceedings of IEEE, vol. 103, no. 3, pp. 332-354, March 2015.

[54] Xilinx Inc., “Partial Reconfiguration Tutorial UG947,” April 2017.

[55] A. K. ELdin, S. Hosny, K. Mohamed, M. Gamal, A. Hussein, E. Elnader, A. Shalash, A. M. Obeid, Y. Ismail, and H. Mostafa, “A Reconfigurable Hardware Platform Implementation for Software Defined Radio using Dynamic Partial Reconfiguration on Xilinx Zynq FPGA,” *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS 2017)*, Boston, MA, USA, pp. 1540-1543, Aug. 2017.

Appendix A: Partioning Algorithm

The MATLAB code listed below implements the portioning algorithm used to define which blocks shall be merged together between all chains.

```
clc;
clear all;

number_of_chains = 5;
% LTE
chain_A = [83     0    0    83;  % CRC
           474    0.5  0   674;  % SEGMENTATION
           320    1.5  3  2120;  % ENCODER
           347    3    0  1547;  % RATE MATCHING
           88     2    0   888;  % CONCAT
           383    4.5  0  2183;  % DESCRAMBLER
           51     0.5  0   251;  % DEMAPPER
           2584   0   13  7784]; % SCFDMA
% WIFI
chain_B = [107    0    0   107;  % SCRMABLER
           45     0    0    45;  % ENCODER
           0      0    0     0;  % PUNCTURE
           151    2    0   951;  % INTERLEAVER
           64     0.5  0   264;  % MAPPER
           1929   0.5  6  4529]; % IFFT
% 3G
chain_C = [24     0    0    24;  % CRC
           407    0.5  1  1007;  % SEGMENTATION
           56     0    0    56;  % ENCODER
           83     0.5  0   283;  % CONCAT
           820    2.5  1  2220;  % INTERLEAVER
           79     0    0    79;  % SPREADING & SCRAMBLING
           6      0    0     6]; % MAPPER
%Bluetooth
chain_D = [91     0.5  0   291;  % HEADER & PAYLOAD SEGMENTATION
           127    0    0   127;  % HEC
           127    0.5  0   327;  % CRC
           7      0    0     7;  % H-WHITENING
           7      0    0     7;  % P-WHITENING
           43     0    0    43;  % H-ENCODER
           169    1    0   569;  % P-ENCODER
           78     1    0   478;  % H-MAPPER
           78     1    0   478]; % P-MAPPER
% 2G
chain_E = [200    1    0   600;  % CRC + BIT ORDERING
           55     0    0    55;  % ENCODER
```

```
               161   0.5   0   361;  % INTERLEAVER
               114   0.5   0   314;  % BRUST FORMATION
               2      0    0     2]; % MAPPER

number_of_blocks_for_chain_A = 8;
number_of_blocks_for_chain_B = 6;
number_of_blocks_for_chain_C = 7;
number_of_blocks_for_chain_D = 9;
number_of_blocks_for_chain_E = 5;

A_chosen = get_min_partition_per_chain (number_of_blocks_for_chain_A ,chain_A);
B_chosen = get_min_partition_per_chain (number_of_blocks_for_chain_B ,chain_B);
C_chosen = get_min_partition_per_chain (number_of_blocks_for_chain_C ,chain_C);
D_chosen = get_min_partition_per_chain (number_of_blocks_for_chain_D ,chain_D);
E_chosen = get_min_partition_per_chain (number_of_blocks_for_chain_E ,chain_E);

for f1=1:1:size(A_chosen,1)
    B_min_matrix = iterate_on_other_chains(A_chosen,f1,B_chosen);
    C_min_matrix = iterate_on_other_chains(A_chosen,f1,C_chosen);
    D_min_matrix = iterate_on_other_chains(A_chosen,f1,D_chosen);
    E_min_matrix = iterate_on_other_chains(A_chosen,f1,E_chosen);
    BCDE_min_matrix (f1,1:4) = [B_min_matrix C_min_matrix D_min_matrix E_min_matrix];
    min_matrix (f1,1) = f1;
    min_matrix (f1,2) = sum(BCDE_min_matrix(f1,:));
    sorted_min_matrix = sort(min_matrix);
    sorted_A_chosen (f1,:) = A_chosen(sorted_min_matrix(f1,1),:);
end

for k1=1:1:size(sorted_A_chosen,1)
    if(size(B_chosen,1) >0)
        [B_chosen] = get_min_resource_and_exclude_merged(sorted_A_chosen,B_chosen,k1,'B');
    end
    if(size(C_chosen,1) >0)
```

```
        [C_chosen] = get_min_resource_and_exclude_merged(sorted_A_chosen,C_chosen,k1,'C');
    end
    if(size(D_chosen,1) >0)
        [D_chosen] = get_min_resource_and_exclude_merged(sorted_A_chosen,D_chosen,k1,'D');
    end
    if(size(E_chosen,1) >0)
        [E_chosen] = get_min_resource_and_exclude_merged(sorted_A_chosen,E_chosen,k1,'E');
    end
end

function [chosen_min_diff]=iterate_on_other_chains(A_chosen, A_chosen_index, chosen_matrix)
    for f2=1:1:size(chosen_matrix,1)
        chosen_diff_matrix(f2,1:3) = abs(A_chosen(A_chosen_index,1:3) - chosen_matrix(f2,1:3));
        chosen_diff_matrix(f2,4) = (chosen_diff_matrix(f2,1))+(40*chosen_diff_matrix(f2,2))+(40*chosen_diff_matrix(f2,3));
    end
    [min_value, min_row] = min(chosen_diff_matrix(:,4));
    chosen_min_diff(1,1) = chosen_diff_matrix(min_row,4);
end

function [chosen_matrix]=get_min_resource_and_exclude_merged(sorted_A_chosen, chosen_matrix,k1,chain_name)
        for k2=1:1:size(chosen_matrix,1)
            diff_matrix(k2,1:3) = abs(sorted_A_chosen(k1,1:3) - chosen_matrix(k2,1:3));
            diff_matrix(k2,4) = (diff_matrix(k2,1))+(40*diff_matrix(k2,2))+(40*diff_matrix(k2,3));
            diff_matrix(k2,5:6) = chosen_matrix(k2,5:6);
        end
        [min_value, min_row] = min(diff_matrix(:,4));
        for k3=1:1:size(chosen_matrix,1)
            AB_min_diff(k3,1) = diff_matrix(min_row,4);
            AB_min_diff(k3,2:3) = sorted_A_chosen(k1,5:6);
            AB_min_diff(k3,4:5) = diff_matrix(k3,5:6);
            AB_min_str(k3,1:2) = num2str(AB_min_diff(k3,2));
            AB_min_str(k3,3:4) = num2str(AB_min_diff(k3,4));
            for s1=1:1:4
```

```
                AB_min_matrix(k3,s1) = str2num(AB_min_str(k3,s1));
            end
            AB_min_matrix(k3,5) = AB_min_diff(k3,3);
            AB_min_matrix(k3,6) = AB_min_diff(k3,5);
        end
        b_remaining_str = num2str(diff_matrix(min_row,5));
        b_remaining = str2num(b_remaining_str(:,1));
        b_remaining(:,2) = str2num(b_remaining_str(:,2));

        v1 = 1; v2 = 1;
        vn = size(chosen_matrix,1);
        while (v1 <= vn)
            if( (b_remaining(1,1) == AB_min_matrix(v1,3)) || ...
                (b_remaining(1,2) == AB_min_matrix(v1,4)) || ...
                ((b_remaining(1,2) < AB_min_matrix(v1,2))&&(AB_min_matrix(v1,6) == 1)))
                chosen_matrix(v2,:) = [];
            else
                v2 = v2 + 1;
            end
            v1 = v1 + 1;
        end
        if(sorted_A_chosen(k1,6) == 1 && diff_matrix(min_row,6) == 1)
            format_specifier = 'All merged blocks of A%d will be merged with all merged blocks of %s%d';
        elseif (sorted_A_chosen(k1,6) == 1 && diff_matrix(min_row,6) == 0)
            format_specifier = 'All merged blocks of A%d will be merged with %s%d';
        elseif (sorted_A_chosen(k1,6) == 0 && diff_matrix(min_row,6) == 1)
            format_specifier = 'A%d will be merged with all merged blocks of %s%d';
        else
            format_specifier = 'A%d will be merged with %s%d';
        end
        out = sprintf(format_specifier,sorted_A_chosen(k1,5),chain_name,diff_matrix(min_row,5))
end

```

```
function [chosen_matrix]=get_min_partition_per_chain(number_of_blocks,input_matrix)
    len = size(input_matrix,1);
    sum_index=len+1;
    chosen_index = 1;
    if (number_of_blocks == 1)
        sum_matrix (:,5) = [11];
    elseif (number_of_blocks == 2)
        sum_matrix (:,5) = [11 22];
    elseif (number_of_blocks == 3)
        sum_matrix (:,5) = [11 22 33];
    elseif (number_of_blocks == 4)
        sum_matrix (:,5) = [11 22 33 44];
    elseif (number_of_blocks == 5)
        sum_matrix (:,5) = [11 22 33 44 55];
    elseif (number_of_blocks == 6)
        sum_matrix (:,5) = [11 22 33 44 55 66];
    elseif (number_of_blocks == 7)
        sum_matrix (:,5) = [11 22 33 44 55 66 77];
    else
        sum_matrix (:,5) = [11 22 33 44 55 66 77 88];
    end
    for m1=1:1:len
        sum_matrix(m1,1:4) = input_matrix(m1,:);
    end

    for n1=1:1:len-1
        for n2=n1:1:len-1
            group_matrix(1,:) = sum(input_matrix(n1:n2,:),1);
            for n3=(n2+1):1:len
                sum_matrix(sum_index,1:4) = group_matrix(1,:) + input_matrix(n3,:);
                sum_matrix(sum_index,5) = str2double(strcat(num2str(n1),num2str(n3)));
                if(n3 == n2+1)
                    sum_matrix(sum_index,6) = 1;
                else
                    sum_matrix(sum_index,6) = 0;
                end
                sum_index = sum_index + 1;
            end
        end
    end

    for l1=1:1:size(sum_matrix,1)
```

```
        if(mod(sum_matrix(l1,4),400) == 2)
            chosen_matrix(chosen_index,:) = sum_matrix(l1,:);
            chosen_index = chosen_index + 1;
        end
    end
end
```

Appendix B: FPGA Prototyping Code

The TCL commands listed below summarizes all the commands used in Xilinx Vivado console in order to optimize, place, route, and generate bit stream files for each chain.

```
###############################
#Open synthesized project
###############################
set_param general.maxThreads 8
cd E:/sherif/masters/Tx_Rx_PRC_Single_RP/DPR_Project

read_checkpoint -cell design_1_i/design_1_i/Tx_AXI_Peripheral_v1_0_0/inst/Tx_R_System Synth/reconfig_modules/4g_tx/Tx_R_Partition.dcp
read_checkpoint -cell design_1_i/design_1_i/Rx_AXI_Peripheral_v1_0_0/inst/Rx_R_System Synth/reconfig_modules/4g_rx/Rx_R_Partition.dcp
###############################
# Do floor_planning
###############################
read_xdc ./xcd_File.xdc
###############################
#RESET_AFTER_CONFIG
###############################
set_property HD.RECONFIGURABLE 1 [get_cells design_1_i/design_1_i/Tx_AXI_Peripheral_v1_0_0/inst/Tx_R_System]
set_property HD.RECONFIGURABLE true [get_cells design_1_i/design_1_i/Tx_AXI_Peripheral_v1_0_0/inst/Tx_R_System]
set_property HD.RECONFIGURABLE 1 [get_cells design_1_i/design_1_i/Rx_AXI_Peripheral_v1_0_0/inst/Rx_R_System]
set_property HD.RECONFIGURABLE true [get_cells design_1_i/design_1_i/Rx_AXI_Peripheral_v1_0_0/inst/Rx_R_System]
###############################
#SNAPPING Mode
###############################
set_property RESET_AFTER_RECONFIG 1 [get_pblocks pblock_Tx_R_System]
set_property SNAPPING_MODE ON [get_pblocks pblock_Tx_R_System]
set_property RESET_AFTER_RECONFIG 1 [get_pblocks pblock_Rx_R_System]
set_property SNAPPING_MODE ON [get_pblocks pblock_Rx_R_System]
write_checkpoint -force Checkpoint/R_Partition.dcp
###############################
#4G Tx/Rx
###############################
```

```
opt_design
place_design
route_design
write_checkpoint −force Implement/4g/top_route_design.dcp
write_debug_probes −force ./Implement/4g/debug_4g_nets.ltx
update_design −cell design_1_i/design_1_i/Tx_AXI_Peripheral_v1_0_0/inst/Tx_R_System −black_box
update_design −cell design_1_i/design_1_i/Rx_AXI_Peripheral_v1_0_0/inst/Rx_R_System −black_box
lock_design −level routing
write_checkpoint −force Checkpoint/static_route_design.dcp
##############################
#3G Tx/Rx
##############################
read_checkpoint −cell design_1_i/design_1_i/Tx_AXI_Peripheral_v1_0_0/inst/Tx_R_System Synth/reconfig_modules/3g_tx/Tx_R_Partition.dcp
read_checkpoint −cell design_1_i/design_1_i/Rx_AXI_Peripheral_v1_0_0/inst/Rx_R_System Synth/reconfig_modules/3g_rx/Rx_R_Partition.dcp
opt_design
place_design
route_design
write_checkpoint −force Implement/3g/top_route_design.dcp
write_debug_probes −force ./Implement/3g/debug_3g_nets.ltx
update_design −cell design_1_i/design_1_i/Tx_AXI_Peripheral_v1_0_0/inst/Tx_R_System −black_box
update_design −cell design_1_i/design_1_i/Rx_AXI_Peripheral_v1_0_0/inst/Rx_R_System −black_box
#2G Tx/Rx
read_checkpoint −cell design_1_i/design_1_i/Tx_AXI_Peripheral_v1_0_0/inst/Tx_R_System Synth/reconfig_modules/2g_tx/Tx_R_Partition.dcp
read_checkpoint −cell design_1_i/design_1_i/Rx_AXI_Peripheral_v1_0_0/inst/Rx_R_System Synth/reconfig_modules/2g_rx/Rx_R_Partition.dcp
opt_design
place_design
route_design
write_checkpoint −force Implement/2g/top_route_design.dcp
write_debug_probes −force ./Implement/2g/debug_2g_nets.ltx
update_design −cell design_1_i/design_1_i/Tx_AXI_Peripheral_v1_0_0/inst/Tx_R_System −black_box
update_design −cell design_1_i/design_1_i/Rx_AXI_Peripheral_v1_0_0/inst/Rx_R_System −black_box
##############################
```

```
#WIFI Tx/Rx
###############################
read_checkpoint -cell design_1_i/design_1_i/Tx_AXI_Peripheral_v1_0_0/inst/Tx_R_System Synth/reconfig_modules/wifi_tx/Tx_R_Partition.dcp
read_checkpoint -cell design_1_i/design_1_i/Rx_AXI_Peripheral_v1_0_0/inst/Rx_R_System Synth/reconfig_modules/wifi_rx/Rx_R_Partition.dcp
opt_design
place_design
route_design
write_checkpoint -force Implement/wifi/top_route_design.dcp
write_debug_probes -force ./Implement/wifi/debug_wifi_nets.ltx
update_design -cell design_1_i/design_1_i/Tx_AXI_Peripheral_v1_0_0/inst/Tx_R_System -black_box
update_design -cell design_1_i/design_1_i/Rx_AXI_Peripheral_v1_0_0/inst/Rx_R_System -black_box
###############################
#Bluetooth Tx/Rx
###############################
read_checkpoint -cell design_1_i/design_1_i/Tx_AXI_Peripheral_v1_0_0/inst/Tx_R_System Synth/reconfig_modules/bluetooth_tx/Tx_R_Partition.dcp
read_checkpoint -cell design_1_i/design_1_i/Rx_AXI_Peripheral_v1_0_0/inst/Rx_R_System Synth/reconfig_modules/bluetooth_rx/Rx_R_Partition.dcp
opt_design
place_design
route_design
write_checkpoint -force Implement/bluetooth/top_route_design.dcp
write_debug_probes -force ./Implement/bluetooth/debug_bluetooth_nets.ltx
close_project
###############################
# VERIFY
###############################
pr_verify -initial Implement/wifi/top_route_design.dcp -additional {Implement/2g/top_route_design.dcp Implement/3g/top_route_design.dcp Implement/4g/top_route_design.dcp Implement/bt/top_route_design.dcp}
###############################
# GENERATE_BIT_STREAM
###############################
```

```
cd E:/sherif/masters/Tx_Rx_Multi_RP_Final/DPR_Project
#############################
# 4G Tx/Rx
#############################
set_property SEVERITY {Warning} [get_drc_checks LUTLP-1]
open_checkpoint Implement/4g/top_route_design.dcp
write_bitstream -file Bitstreams/config_4g.bit -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_4g_pblock_Tx_R1_Partition_partial.bit” Bitstreams/Tx_R1_4g.bin -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_4g_pblock_Tx_R2_Partition_partial.bit” Bitstreams/Tx_R2_4g.bin -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_4g_pblock_Tx_R3_Partition_partial.bit” Bitstreams/Tx_R3_4g.bin -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_4g_pblock_Rx_R1_Partition_partial.bit” Bitstreams/Rx_R1_4g.bin -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_4g_pblock_Rx_R2_Partition_partial.bit” Bitstreams/Rx_R2_4g.bin -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_4g_pblock_Rx_R3_Partition_partial.bit” Bitstreams/Rx_R3_4g.bin -force
close_project
#############################
# 3G Tx/Rx
#############################
open_checkpoint Implement/3g/top_route_design.dcp
write_bitstream -file Bitstreams/config_3g.bit -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_3g_pblock_Tx_R3_Partition_partial.bit” Bitstreams/Tx_R2_3g.bin -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_3g_pblock_Rx_R1_Partition_partial.bit” Bitstreams/Rx_R3_3g.bin -force
close_project
```

```
##############################
# 2G Tx/Rx
##############################
open_checkpoint Implement/2g/top_route_design.dcp
write_bitstream -file Bitstreams/config_2g.bit -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_2g_pblock_Tx_R1_Partition_partial.bit” Bitstreams/Tx_R1_2g.bin -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_2g_pblock_Rx_R2_Partition_partial.bit” Bitstreams/Rx_R2_2g.bin -force
close_project
##############################
# WIFI Tx/Rx
##############################
open_checkpoint Implement/wifi/top_route_design.dcp
write_bitstream -file Bitstreams/config_wifi.bit -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_wifi_pblock_Tx_R3_Partition_partial.bit” Bitstreams/Tx_R2_wifi.bin -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_wifi_pblock_Rx_R1_Partition_partial.bit” Bitstreams/Rx_R3_wifi.bin -force
close_project
##############################
# Bluetooth Tx/Rx
##############################
open_checkpoint Implement/bt/top_route_design.dcp
write_bitstream -file Bitstreams/config_blue.bit -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_blue_pblock_Tx_R1_Partition_partial.bit” Bitstreams/Tx_R1_blue.bin -force
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit ”up 0 Bitstreams/config_blue_pblock_Rx_R2_Partition_partial.bit” Bitstreams/Rx_R2_blue.bin -force
close_project
```

Appendix C: Reconfiguration Algorithm

Listed below the C-code used to implement the program running on Xilinx SDK. The program transfers the data across the design registers whose addresses are mapped using Xilinx Vivado.

The code consists of two major functions: *DPR_Int*, and *main*. Where *DPR_Int* is responsible for initializing the registers values. Meanwhile, *main* is responsible for displaying a menu containing options to the user. It reconfigures the design according to the selected option.

```
//==========================================
// DPR Initialization
//==========================================
int DPR_Int()
{
int Status;
//======================================
// Flush and disable Data and instruction Cache
//======================================
Xil_DCacheDisable();
Xil_ICacheDisable();
//======================================
// Initialize SD controller and transfer partials to DDR
//======================================
while (SD_Init() != XST_SUCCESS);
SD_TransferPartial(”Tx_R1_2g.bin”        ,Tx_P1_2G_ADDR   ,(
    Tx_P1_BITFILE_LEN));
SD_TransferPartial(”Tx_R1_4g.bin”        ,Tx_P1_4G_ADDR   ,(
    Tx_P1_BITFILE_LEN));
SD_TransferPartial(”Tx_R1_blue.bin”      ,Tx_P1_BLUE_ADDR ,(
    Tx_P1_BITFILE_LEN));
SD_TransferPartial(”Tx_R2_3g.bin”        ,Tx_P2_3G_ADDR   ,(
    Tx_P2_BITFILE_LEN));
SD_TransferPartial(”Tx_R2_4g.bin”        ,Tx_P2_4G_ADDR   ,(
    Tx_P2_BITFILE_LEN));
SD_TransferPartial(”Tx_R2_wifi.bin”      ,Tx_P2_WIFI_ADDR ,(
    Tx_P2_BITFILE_LEN));
SD_TransferPartial(”Tx_R3_4g.bin”        ,Tx_P3_4G_ADDR   ,(
    Tx_P3_BITFILE_LEN));
SD_TransferPartial(”Rx_R1_4g.bin”        ,Rx_P1_4G_ADDR   ,(
    Rx_P1_BITFILE_LEN));
SD_TransferPartial(”Rx_R2_2g.bin”        ,Rx_P2_2G_ADDR   ,(
    Rx_P2_BITFILE_LEN));
SD_TransferPartial(”Rx_R2_4g.bin”        ,Rx_P2_4G_ADDR   ,(
    Rx_P2_BITFILE_LEN));
```

```
SD_TransferPartial(”Rx_R2_blue.bin”      ,Rx_P2_BLUE_ADDR ,(Rx_P2_BITFILE_LEN));
SD_TransferPartial(”Rx_R3_3g.bin”        ,Rx_P3_3G_ADDR   ,(Rx_P3_BITFILE_LEN));
SD_TransferPartial(”Rx_R3_4g.bin”        ,Rx_P3_4G_ADDR   ,(Rx_P3_BITFILE_LEN));
SD_TransferPartial(”Rx_R3_wifi.bin”      ,Rx_P3_WIFI_ADDR ,(Rx_P3_BITFILE_LEN));
//===================================
xil_printf(”Partial Binaries transferred successfully!\r\n”);
//===================================
// Initialize Device Configuration Interface
//===================================
DcfgInstPtr = &DcfgInstance;
XDcfg_0 = XDcfg_LookupConfig(XPAR_XDCFG_0_DEVICE_ID) ;
Status =  XDcfg_CfgInitialize(DcfgInstPtr, XDcfg_0, XDcfg_0->BaseAddr);
if (Status != XST_SUCCESS) {
        return XST_FAILURE;
}
//===================================
// De-select PCAP as the configuration device as we are going to use the ICAP
//===================================
XDcfg_ClearControlRegister(DcfgInstPtr, XDCFG_CTRL_PCAP_PR_MASK | XDCFG_CTRL_PCAP_MODE_MASK);
//===================================
// Display PRC status
//===================================
print(”Putting the PRC core's System RP in Shutdown mode\n\r”);
//===================================
Xil_Out32(Tx_P1_CONTROL,0);
Xil_Out32(Tx_P2_CONTROL,0);
Xil_Out32(Tx_P3_CONTROL,0);
Xil_Out32(Rx_P1_CONTROL,0);
Xil_Out32(Rx_P2_CONTROL,0);
Xil_Out32(Rx_P3_CONTROL,0);
//===================================
print(”Waiting for the shutdown to occur\r\n”);
//===================================
while(!(Xil_In32(Tx_P1_STATUS)&0x80));
while(!(Xil_In32(Tx_P2_STATUS)&0x80));
while(!(Xil_In32(Tx_P3_STATUS)&0x80));
while(!(Xil_In32(Rx_P1_STATUS)&0x80));
while(!(Xil_In32(Rx_P2_STATUS)&0x80));
while(!(Xil_In32(Rx_P3_STATUS)&0x80));
print(”System RP is shutdown\r\n”);
//=====================================
print(”Initializing RM bitstream address \r\n”);
//=====================================
Xil_Out32(Tx_P1_BS_ADDRESS0,Tx_P1_2G_ADDR);
Xil_Out32(Tx_P1_BS_ADDRESS1,Tx_P1_4G_ADDR);
Xil_Out32(Tx_P1_BS_ADDRESS2,Tx_P1_BLUE_ADDR);
Xil_Out32(Tx_P2_BS_ADDRESS0,Tx_P2_3G_ADDR);
Xil_Out32(Tx_P2_BS_ADDRESS1,Tx_P2_4G_ADDR);
Xil_Out32(Tx_P2_BS_ADDRESS2,Tx_P2_WIFI_ADDR);
Xil_Out32(Tx_P3_BS_ADDRESS0,Tx_P3_4G_ADDR);
Xil_Out32(Rx_P1_BS_ADDRESS0,Rx_P1_4G_ADDR);
Xil_Out32(Rx_P2_BS_ADDRESS0,Rx_P2_2G_ADDR);
Xil_Out32(Rx_P2_BS_ADDRESS1,Rx_P2_4G_ADDR);
Xil_Out32(Rx_P2_BS_ADDRESS2,Rx_P2_BLUE_ADDR);
Xil_Out32(Rx_P3_BS_ADDRESS0,Rx_P3_3G_ADDR);
Xil_Out32(Rx_P3_BS_ADDRESS1,Rx_P3_4G_ADDR);
Xil_Out32(Rx_P3_BS_ADDRESS2,Rx_P3_WIFI_ADDR);
//=====================================
print(”Initializing RM size registers \r\n”);
//=====================================
Xil_Out32(Tx_P1_BS_SIZE0,Tx_P1_BITFILE_LEN);
Xil_Out32(Tx_P1_BS_SIZE1,Tx_P1_BITFILE_LEN);
Xil_Out32(Tx_P1_BS_SIZE2,Tx_P1_BITFILE_LEN);
Xil_Out32(Tx_P2_BS_SIZE0,Tx_P2_BITFILE_LEN);
Xil_Out32(Tx_P2_BS_SIZE1,Tx_P2_BITFILE_LEN);
Xil_Out32(Tx_P2_BS_SIZE2,Tx_P2_BITFILE_LEN);
Xil_Out32(Tx_P3_BS_SIZE0,Tx_P3_BITFILE_LEN);
Xil_Out32(Rx_P1_BS_SIZE0,Rx_P1_BITFILE_LEN);
Xil_Out32(Rx_P2_BS_SIZE0,Rx_P2_BITFILE_LEN);
Xil_Out32(Rx_P2_BS_SIZE1,Rx_P2_BITFILE_LEN);
Xil_Out32(Rx_P2_BS_SIZE2,Rx_P2_BITFILE_LEN);
Xil_Out32(Rx_P3_BS_SIZE0,Rx_P3_BITFILE_LEN);
Xil_Out32(Rx_P3_BS_SIZE1,Rx_P3_BITFILE_LEN);
Xil_Out32(Rx_P3_BS_SIZE2,Rx_P3_BITFILE_LEN);
//=====================================
print(”Initializing RM trigger ID registers \r\n”);
//=====================================
Xil_Out32(Tx_P1_TRIGGER0,0);
Xil_Out32(Tx_P1_TRIGGER1,1);
Xil_Out32(Tx_P1_TRIGGER2,2);
Xil_Out32(Tx_P2_TRIGGER0,0);
Xil_Out32(Tx_P2_TRIGGER1,1);
Xil_Out32(Tx_P2_TRIGGER2,2);
```

```
Xil_Out32(Tx_P3_TRIGGER0,0);
Xil_Out32(Rx_P1_TRIGGER0,0);
Xil_Out32(Rx_P2_TRIGGER0,0);
Xil_Out32(Rx_P2_TRIGGER1,1);
Xil_Out32(Rx_P2_TRIGGER2,2);
Xil_Out32(Rx_P3_TRIGGER0,0);
Xil_Out32(Rx_P3_TRIGGER1,1);
Xil_Out32(Rx_P3_TRIGGER2,2);
//====================================
print(”Initializing RM address registers \r\n”);
//====================================
Xil_Out32(Tx_P1_RM_BS_INDEX0,0);
Xil_Out32(Tx_P1_RM_BS_INDEX1,1);
Xil_Out32(Tx_P1_RM_BS_INDEX2,2);
Xil_Out32(Tx_P2_RM_BS_INDEX0,0);
Xil_Out32(Tx_P2_RM_BS_INDEX1,1);
Xil_Out32(Tx_P2_RM_BS_INDEX2,2);
Xil_Out32(Tx_P3_RM_BS_INDEX0,0);
Xil_Out32(Rx_P1_RM_BS_INDEX0,0);
Xil_Out32(Rx_P2_RM_BS_INDEX0,0);
Xil_Out32(Rx_P2_RM_BS_INDEX1,1);
Xil_Out32(Rx_P2_RM_BS_INDEX2,2);
Xil_Out32(Tx_P3_RM_BS_INDEX0,0);
Xil_Out32(Rx_P3_RM_BS_INDEX1,1);
Xil_Out32(Rx_P3_RM_BS_INDEX1,2);
//====================================
print(”Initializing RM control registers \r\n”);
//====================================
Xil_Out32(Tx_P1_RM_CONTROL0,0);
Xil_Out32(Tx_P1_RM_CONTROL1,0);
Xil_Out32(Tx_P1_RM_CONTROL2,0);
Xil_Out32(Tx_P2_RM_CONTROL0,0);
Xil_Out32(Tx_P2_RM_CONTROL1,0);
Xil_Out32(Tx_P2_RM_CONTROL2,0);
Xil_Out32(Tx_P3_RM_CONTROL0,0);
Xil_Out32(Rx_P1_RM_CONTROL0,0);
Xil_Out32(Rx_P2_RM_CONTROL0,0);
Xil_Out32(Rx_P2_RM_CONTROL1,0);
Xil_Out32(Rx_P2_RM_CONTROL2,0);
Xil_Out32(Rx_P3_RM_CONTROL0,0);
Xil_Out32(Rx_P3_RM_CONTROL1,0);
Xil_Out32(Rx_P3_RM_CONTROL2,0);
//====================================
print(”Putting the PRC core’s System RP in Restart with
    Status mode\n\r”);
//====================================
Xil_Out32(Tx_P1_CONTROL,2);
Xil_Out32(Tx_P2_CONTROL,2);
Xil_Out32(Tx_P3_CONTROL,2);
Xil_Out32(Rx_P1_CONTROL,2);
Xil_Out32(Rx_P2_CONTROL,2);
Xil_Out32(Rx_P3_CONTROL,2);
//===================================
xil_printf(”Reading the Math Tx_P1 status=%x\n\r”,Xil_In32
    (Tx_P1_STATUS));
xil_printf(”Reading the Math Tx_P2 status=%x\n\r”,Xil_In32
    (Tx_P2_STATUS));
xil_printf(”Reading the Math Tx_P3 status=%x\n\r”,Xil_In32
    (Tx_P3_STATUS));
xil_printf(”Reading the Math Rx_P1 status=%x\n\r”,Xil_In32
    (Rx_P1_STATUS));
xil_printf(”Reading the Math Rx_P2 status=%x\n\r”,Xil_In32
    (Rx_P2_STATUS));
xil_printf(”Reading the Math Rx_P3 status=%x\n\r”,Xil_In32
    (Rx_P3_STATUS));
//===================================
// Loading 2G Data_in from SD Card to DDR and input Array
//===================================
Status = read_files(”twogdata.txt”, INPUT_DATA_SIZE_2G,
    input_2G);
if (Status != XST_SUCCESS)
{
        xil_printf(” Test 2G failed \r\n”);
        return XST_FAILURE;
}
xil_printf(”2G file Successfully Loaded  \r\n”);
//===================================
// Loading 3G Data_in from SD Card to DDR and input Array
//===================================
 Status = read_files(”thrgdata.txt”, INPUT_DATA_SIZE_3G,
     input_3G);
if (Status != XST_SUCCESS)
{
        xil_printf(” Test 3G failed \r\n”);
        return XST_FAILURE;
}
xil_printf(”3G file Successfully Loaded  \r\n”);
//===================================
// Loading 4G Data_in from SD Card to DDR and input Array
//===================================
Status = read_files(”forgdata.txt”, INPUT_DATA_SIZE_4G,
    input_4G);
```

```
if (Status != XST_SUCCESS)
{
        xil_printf(” Test 4G failed \r\n”);
        return XST_FAILURE;
}

xil_printf(”4G file Successfully Loaded  \r\n”);
//====================================
// Loading WIFI Data_in from SD Card to DDR and input Array
//====================================
Status = read_files(”wifidata.txt”, INPUT_DATA_SIZE_WIFI, input_wifi);
if (Status != XST_SUCCESS)
{
        xil_printf(” Test WIFI failed \r\n”);
        return XST_FAILURE;
}
xil_printf(”WiFi file Successfully Loaded  \r\n”);
//====================================
// Loading BLUETOOTH Data_in from SD Card to DDR and input Array
//====================================
Status = read_files(”bluedata.txt”, INPUT_DATA_SIZE_BLUETOOTH, input_BLUETOOTH);
if (Status != XST_SUCCESS)
{
        xil_printf(” Test BLUETOOTH failed \r\n”);
        return XST_FAILURE;
}
xil_printf(”BLUETOOTH file Successfully Loaded  \r\n”);
//====================================
return XST_SUCCESS;
}
//========================================
// Main Function
//========================================
int main()
{
//====================================
if(Init_Timer_ARM() != XST_SUCCESS)
{
xil_printf(”Error in Init_Timer_ARM!\n\r”);
return 0;
}
//====================================
if(DPR_Int() != XST_SUCCESS)
{
xil_printf("Error in DPR_Int!\n\r");
return 0;
}
//================================
while(1)
{
int loading_done_Tx_P1,loading_done_Tx_P2,loading_done_Tx_P3;
int Tx_P1_Status,Tx_P2_Status,Tx_P3_Status;
int loading_done_Rx_P1,loading_done_Rx_P2,loading_done_Rx_P3;
int Rx_P1_Status,Rx_P2_Status,Rx_P3_Status;
//==================================================
int Exit = 0;
int OptionNext = 0;     // start-up default
int choose_system = 0;  //0 = 2G , 1 = 3G , 2 = 4G , 3 = WIFI , 4 = BLUETOOTH
XTime tStart, tEnd;
//==================================================
while(Exit != 1)
{
//==============================================
do
{
print("    1: 2G\n\r");
print("    2: 3G\n\r");
print("    3: 4G\n\r");
print("    4: WIFI\n\r");
print("    5: BLUETOOTH\n\r");
print("    6: Test Chain\n\r");
print("    7: Exit\n\r");
print("> ");
OptionNext = inbyte();
if (isalpha(OptionNext))
        OptionNext = toupper(OptionNext);
xil_printf("%c\n\r", OptionNext);
} while (!isdigit(OptionNext));
//==============================================
switch (OptionNext)
{
//==========================================
case '1':        // 2G
        //======================================
        choose_system = 0;
        xil_printf(”Generating software trigger for 2G reconfiguration\r\n”);
        XTime_GetTime(&tStart);
        //==========================================
        Tx_P1_Status=Xil_In32(Tx_P1_SW_TRIGGER);
        Rx_P2_Status=Xil_In32(Rx_P2_SW_TRIGGER);
        //==========================================
        if(!(Tx_P1_STATUS&0x8000)) { Xil_Out32(Tx_P1_SW_TRIGGER,0); }
        if(!(Rx_P2_STATUS&0x8000)) { Xil_Out32(Rx_P2_SW_TRIGGER,0); }
        //==========================================
        loading_done_Tx_P1 = 0;
        loading_done_Rx_P2 = 0;
        //==========================================
        while((!loading_done_Tx_P1)||(!loading_done_Rx_P2))
        {
                //======================================
                Tx_P1_Status=Xil_In32(Tx_P1_STATUS)&0x07;
                Rx_P2_Status=Xil_In32(Rx_P2_STATUS)&0x07;
                //======================================
                switch(Tx_P1_Status) {
                        case 7 : /*print(”RM loaded\r\n”)*/; loading_done_Tx_P1=1; break;
                        case 6 : print(”RM is being reset\r\n”); break;
                        case 5 : print(”Software start-up step\r\n”); break;
                        case 4 : /*print(”Loading new RM\r\n”)*/; break;
                        case 2 : print(”Software shutdown\r\n”); break;
                        case 1 : print(”Hardware shutdown\r\n”); break;
                }
                //======================================
                switch(Rx_P2_Status) {
                        case 7 : /*print(”RM loaded\r\n”)*/; loading_done_Rx_P2=1; break;
                        case 6 : print(”RM is being reset\r\n”); break;
                        case 5 : print(”Software start-up step\r\n”); break;
                        case 4 : /*print(”Loading new RM\r\n”)*/; break;
```

```
                    case 2 : print("Software shutdown\r\n"); break;
                    case 1 : print("Hardware shutdown\r\n"); break;
                }
                //========================================
        }
        //============================================
        XTime_GetTime(&tEnd);
        xil_printf("2G Reconfiguration Completed!\n\r");
        printf("Reconfiguration took %.2f ms.\n",1.0 * (tEnd - tStart) / (COUNTS_PER_SECOND/1000));
        //============================================
break;
//================================================
case '2':       // 3G
        //============================================
        choose_system = 1;
        xil_printf("Generating software trigger for 3G reconfiguration\r\n");
        XTime_GetTime(&tStart);
        //============================================
        Tx_P2_Status=Xil_In32(Tx_P2_SW_TRIGGER);
        Rx_P3_Status=Xil_In32(Rx_P3_SW_TRIGGER);
        //============================================
        if(!(Tx_P2_STATUS&0x8000)) { Xil_Out32(Tx_P2_SW_TRIGGER,0); }
        if(!(Rx_P3_STATUS&0x8000)) { Xil_Out32(Rx_P3_SW_TRIGGER,0); }
        //============================================
        loading_done_Tx_P2 = 0;
        loading_done_Rx_P3 = 0;
        //============================================
        while((!loading_done_Tx_P2)||(!loading_done_Rx_P3))
        {
                //========================================
                Tx_P2_Status=Xil_In32(Tx_P2_STATUS)&0x07;
                Rx_P3_Status=Xil_In32(Rx_P3_STATUS)&0x07;
                //========================================
                switch(Tx_P2_Status) {
                    case 7 : /*print("RM loaded\r\n")*/; loading_done_Tx_P2=1; break;
                    case 6 : print("RM is being reset\r\n"); break;
                    case 5 : print("Software start-up
                            step\r\n”); break;
                        case 4 : /*print(”Loading new RM\r\n”)*/; break;
                        case 2 : print(”Software shutdown\r\n”); break;
                        case 1 : print(”Hardware shutdown\r\n”); break;
                }
                //========================================
                switch(Rx_P3_Status) {
                        case 7 : /*print(”RM loaded\r\n”)*/; loading_done_Rx_P3=1; break;
                        case 6 : print(”RM is being reset\r\n”); break;
                        case 5 : print(”Software start-up step\r\n”); break;
                        case 4 : /*print(”Loading new RM\r\n”)*/; break;
                        case 2 : print(”Software shutdown\r\n”); break;
                        case 1 : print(”Hardware shutdown\r\n”); break;
                }
                //========================================
        }
        XTime_GetTime(&tEnd);
        xil_printf(”3G Reconfiguration Completed!\n\r”);
        printf(”Reconfiguration took %.2f ms.\n”,1.0 * (tEnd - tStart) / (COUNTS_PER_SECOND/1000));
        //============================================
break;
//============================================
case ’3’:       // 4G
        //============================================
        choose_system = 2;
        xil_printf(”Generating software trigger for 4G reconfiguration\r\n”);
        XTime_GetTime(&tStart);
        //============================================
        Tx_P1_Status=Xil_In32(Tx_P1_SW_TRIGGER);
        Tx_P2_Status=Xil_In32(Tx_P2_SW_TRIGGER);
        Tx_P3_Status=Xil_In32(Tx_P3_SW_TRIGGER);
        //============================================
        Rx_P1_Status=Xil_In32(Rx_P1_SW_TRIGGER);
        Rx_P2_Status=Xil_In32(Rx_P2_SW_TRIGGER);
        Rx_P3_Status=Xil_In32(Rx_P3_SW_TRIGGER);
```

```
	//==========================================
	if(!(Tx_P1_STATUS&0x8000)) { Xil_Out32(Tx_P1_SW_TRIGGER,0); }
	if(!(Tx_P2_STATUS&0x8000)) { Xil_Out32(Tx_P2_SW_TRIGGER,0); }
	if(!(Tx_P3_STATUS&0x8000)) { Xil_Out32(Tx_P3_SW_TRIGGER,0); }
	//==========================================
	if(!(Rx_P1_STATUS&0x8000)) { Xil_Out32(Rx_P1_SW_TRIGGER,0); }
	if(!(Rx_P2_STATUS&0x8000)) { Xil_Out32(Rx_P2_SW_TRIGGER,0); }
	if(!(Rx_P3_STATUS&0x8000)) { Xil_Out32(Rx_P3_SW_TRIGGER,0); }
	//==========================================
	loading_done_Tx_P1 = 0;
	loading_done_Tx_P2 = 0;
	loading_done_Tx_P3 = 0;
	//==========================================
	loading_done_Rx_P1 = 0;
	loading_done_Rx_P2 = 0;
	loading_done_Rx_P3 = 0;
	//==========================================
	while((!loading_done_Tx_P1)||(!loading_done_Tx_P2)||(!loading_done_Tx_P3)||
			(!loading_done_Rx_P1)||(!loading_done_Rx_P2)||(!loading_done_Rx_P3))
	{
		//==========================================
		Tx_P1_Status=Xil_In32(Tx_P1_STATUS)&0x07;
		Tx_P2_Status=Xil_In32(Tx_P2_STATUS)&0x07;
		Tx_P3_Status=Xil_In32(Tx_P3_STATUS)&0x07;
		//==========================================
		Rx_P1_Status=Xil_In32(Rx_P1_STATUS)&0x07;
		Rx_P2_Status=Xil_In32(Rx_P2_STATUS)&0x07;
		Rx_P3_Status=Xil_In32(Rx_P3_STATUS)&0x07;
		//==========================================
		switch(Tx_P1_Status) {
			case 7 : /*print(”RM loaded\r\n”)*/; loading_done_Tx_P1=1; break;
			case 6 : print(”RM is being reset\r\n”); break;
			case 5 : print(”Software start-up step\r\n”); break;
			case 4 : /*print(”Loading new RM\r
\n");*/ break;
            case 2 : print("Software shutdown\r\n"); break;
            case 1 : print("Hardware shutdown\r\n"); break;
        }
        //========================================
        switch(Tx_P2_Status) {
            case 7 : /*print("RM loaded\r\n");*/ loading_done_Tx_P2=1; break;
            case 6 : print("RM is being reset\r\n"); break;
            case 5 : print("Software start-up step\r\n"); break;
            case 4 : /*print("Loading new RM\r\n");*/ break;
            case 2 : print("Software shutdown\r\n"); break;
            case 1 : print("Hardware shutdown\r\n"); break;
        }
        //========================================
        switch(Tx_P3_Status) {
            case 7 : /*print("RM loaded\r\n");*/ loading_done_Tx_P3=1; break;
            case 6 : print("RM is being reset\r\n"); break;
            case 5 : print("Software start-up step\r\n"); break;
            case 4 : /*print("Loading new RM\r\n");*/ break;
            case 2 : print("Software shutdown\r\n"); break;
            case 1 : /*print("Hardware shutdown\r\n");*/ break;
        }
        //========================================
        switch(Rx_P1_Status) {
            case 7 : /*print("RM loaded\r\n");*/ loading_done_Rx_P1=1; break;
            case 6 : print("RM is being reset\r\n"); break;
            case 5 : print("Software start-up step\r\n"); break;
            case 4 : /*print("Loading new RM\r\n");*/ break;
            case 2 : print("Software shutdown\r\n"); break;
            case 1 : /*print("Hardware shutdown\r\n");*/ break;
        }
        //======================================
        switch(Rx_P2_Status) {
            case 7 : /*print("RM loaded\r\n");*/ loading_done_Rx_P2=1; break;
            case 6 : print("RM is being reset\r\n"); break;
            case 5 : print("Software start-up step\r\n"); break;
            case 4 : /*print("Loading new RM\r\n");*/ break;
            case 2 : print("Software shutdown\r\n"); break;
            case 1 : print("Hardware shutdown\r\n"); break;
        }
        //======================================
        switch(Rx_P3_Status) {
            case 7 : /*print("RM loaded\r\n");*/ loading_done_Rx_P3=1; break;
            case 6 : print("RM is being reset\r\n"); break;
            case 5 : print("Software start-up step\r\n"); break;
            case 4 : /*print("Loading new RM\r\n");*/ break;
            case 2 : print("Software shutdown\r\n"); break;
            case 1 : print("Hardware shutdown\r\n"); break;
        }
        //======================================
    }
    XTime_GetTime(&tEnd);
    xil_printf("4G Reconfiguration Completed!\n\r");
    printf("Reconfiguration took %.2f ms.\n",1.0 * (tEnd - tStart) / (COUNTS_PER_SECOND/1000));
    //==========================================
break;
//==============================================
case '4':       // WIFI
    //==========================================
    choose_system = 3;
    xil_printf("Generating software trigger for WIFI reconfiguration\r\n");
    XTime_GetTime(&tStart);
    //==========================================
    Tx_P2_Status=Xil_In32(Tx_P2_SW_TRIGGER);
    Rx_P3_Status=Xil_In32(Rx_P3_SW_TRIGGER);
    //==========================================
    if(!(Tx_P2_STATUS&0x8000)) { Xil_Out32(Tx_P2_SW_TRIGGER,0); }
    if(!(Rx_P3_STATUS&0x8000)) { Xil_Out32(Rx_P3_SW_TRIGGER,0); }
    //==========================================
    loading_done_Tx_P2 = 0;
    loading_done_Rx_P3 = 0;
    //==========================================
    while((!loading_done_Tx_P2)||(!loading_done_Rx_P3))
    {
        //==================================
        Tx_P2_Status=Xil_In32(Tx_P2_STATUS)&0x07;
        Rx_P3_Status=Xil_In32(Rx_P3_STATUS)&0x07;
        //==================================
        switch(Tx_P2_Status) {
            case 7 : /*print("RM loaded\r\n")*/; loading_done_Tx_P2=1; break;
            case 6 : print("RM is being reset\r\n"); break;
            case 5 : print("Software start-up step\r\n"); break;
            case 4 : /*print("Loading new RM\r\n")*/; break;
            case 2 : print("Software shutdown\r\n"); break;
            case 1 : print("Hardware shutdown\r\n"); break;
        }
        //==================================
        switch(Rx_P3_Status) {
            case 7 : /*print("RM loaded\r\n")*/; loading_done_Rx_P3=1; break;
            case 6 : print("RM is being reset\r\n"); break;
            case 5 : print("Software start-up step\r\n"); break;
            case 4 : /*print("Loading new RM\r
                            \n")*/; break;
            case 2 : print("Software shutdown\r\n"); break;
            case 1 : print("Hardware shutdown\r\n"); break;
        }
        //=======================================
    }
    XTime_GetTime(&tEnd);
    xil_printf("WiFi Reconfiguration Completed!\n\r");
    printf("Reconfiguration took %.2f ms.\n",1.0 * (tEnd - tStart) / (COUNTS_PER_SECOND/1000));
    //=======================================
break;
//=======================================
case '5':               // BLUETOOTH
    //=======================================
    choose_system = 4;
    xil_printf("Generating software trigger for BLUETOOTH reconfiguration\r\n");
    XTime_GetTime(&tStart);
    //=======================================
    Tx_P1_Status=Xil_In32(Tx_P1_SW_TRIGGER);
    Rx_P2_Status=Xil_In32(Rx_P2_SW_TRIGGER);
    //=======================================
    if(!(Tx_P1_STATUS&0x8000)) { Xil_Out32(Tx_P1_SW_TRIGGER,0); }
    if(!(Rx_P2_STATUS&0x8000)) { Xil_Out32(Rx_P2_SW_TRIGGER,0); }
    //=======================================
    loading_done_Tx_P1 = 0;
    loading_done_Rx_P2 = 0;
    //=======================================
    while((!loading_done_Tx_P1)||(!loading_done_Rx_P2))
    {
        //=======================================
        Tx_P1_Status=Xil_In32(Tx_P1_STATUS)&0x07;
        Rx_P2_Status=Xil_In32(Rx_P2_STATUS)&0x07;
        //=======================================
        switch(Tx_P1_Status) {
            case 7 : /*print("RM loaded\r\n")*/; loading_done_Tx_P1=1; break;
            case 6 : print("RM is being reset\r\n"); break;
            case 5 : print("Software start-up
                           step\r\n"); break;
                        case 4 : /*print("Loading new RM\r
                           \n")*/; break;
                        case 2 : print("Software shutdown\
                           r\n"); break;
                        case 1 : print("Hardware shutdown\
                           r\n"); break;
                }
                //======================================
                switch(Rx_P2_Status) {
                        case 7 : /*print("RM loaded\r\n")
                           */; loading_done_Rx_P2=1; break;
                        case 6 : print("RM is being reset\
                           r\n"); break;
                        case 5 : print("Software start-up
                           step\r\n"); break;
                        case 4 : /*print("Loading new RM\r
                           \n")*/; break;
                        case 2 : print("Software shutdown\
                           r\n"); break;
                        case 1 : print("Hardware shutdown\
                           r\n"); break;
                }
                //======================================
        }
        //==========================================
        XTime_GetTime(&tEnd);
        xil_printf("BLUETOOTH Reconfiguration Completed!\n
           \r");
        printf("Reconfiguration took %.2f ms.\n",1.0 * (
           tEnd - tStart) / (COUNTS_PER_SECOND/1000));
        //==========================================
break;
//==============================================
case '6':       // Test Chain
        //==========================================
        Xil_Out32(
           XPAR_TX_R1_AXI_PERIPHERAL_V1_0_0_BASEADDR,
           choose_system);      //Option in HDL code
        Xil_Out32(
           XPAR_TX_R2_AXI_PERIPHERAL_V1_0_0_BASEADDR,
           choose_system);      //Option in HDL code
        Xil_Out32(
           XPAR_TX_R3_AXI_PERIPHERAL_V1_0_0_BASEADDR,
           choose_system);      //Option in HDL code
        Xil_Out32(
           XPAR_RX_R1_AXI_PERIPHERAL_V1_0_0_BASEADDR,
           choose_system);     //Option in HDL code
        Xil_Out32(
           XPAR_RX_R2_AXI_PERIPHERAL_V1_0_0_BASEADDR,
           choose_system);     //Option in HDL code
        Xil_Out32(
           XPAR_RX_R3_AXI_PERIPHERAL_V1_0_0_BASEADDR,
           choose_system);     //Option in HDL code
        //==========================================
        Xil_Out32(
           XPAR_TX_R1_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);
                                       //Reset in HDL
           code
        Xil_Out32(
           XPAR_TX_R2_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);
                                       //Reset in HDL
           code
        Xil_Out32(
           XPAR_TX_R3_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);
                                       //Reset in HDL
           code
        Xil_Out32(
           XPAR_RX_R1_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);
                                       //Reset in HDL
           code
        Xil_Out32(
           XPAR_RX_R2_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);
                                       //Reset in HDL
           code
        Xil_Out32(
           XPAR_RX_R3_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);
                                       //Reset in HDL
           code
        //==========================================
        usleep(1);
        //==========================================
        Xil_Out32(
           XPAR_TX_R1_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,0);
                                       //Reset in HDL
           code
        Xil_Out32(
           XPAR_TX_R2_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,0);
                                       //Reset in HDL
           code
        Xil_Out32(
           XPAR_TX_R3_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,0);
```

```
                                                        //Reset in HDL code
        Xil_Out32(XPAR_RX_R1_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,0);  //Reset in HDL code
        Xil_Out32(XPAR_RX_R2_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,0);  //Reset in HDL code
        Xil_Out32(XPAR_RX_R3_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,0);  //Reset in HDL code
        //==============================================
        usleep(1);
        //==============================================
        Xil_Out32(XPAR_TX_R1_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);  //Reset in HDL code
        Xil_Out32(XPAR_TX_R2_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);  //Reset in HDL code
        Xil_Out32(XPAR_TX_R3_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);  //Reset in HDL code
        Xil_Out32(XPAR_RX_R1_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);  //Reset in HDL code
        Xil_Out32(XPAR_RX_R2_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);  //Reset in HDL code
        Xil_Out32(XPAR_RX_R3_AXI_PERIPHERAL_V1_0_0_BASEADDR+12,1);  //Reset in HDL code
        //==============================================
        usleep(1);
        //==============================================
        if(choose_system == 0)
        {
                xil_printf(”Testing 2g!\n\r”);
                INPUT_INTERFACE_AXI_mWriteReg(
                    XPAR_INPUT_INTERFACE_AXI_0_S00_AXI_BASEADDR
                    ,
                    INPUT_INTERFACE_AXI_S00_AXI_SLV_REG0_OFFSET
                    , INPUT_DATA_RATE_2G);
                setup_DMA0();
                xil_printf(”Done!\n\r”);
        }
        //========================================
        else if(choose_system == 1)
        {
                xil_printf(”Testing 3g!\n\r”);
                INPUT_INTERFACE_AXI_mWriteReg(
                    XPAR_INPUT_INTERFACE_AXI_1_S00_AXI_BASEADDR
                    ,
                    INPUT_INTERFACE_AXI_S00_AXI_SLV_REG0_OFFSET
                    , INPUT_DATA_RATE_3G);
                setup_DMA1();
                xil_printf(”Done!\n\r”);
        }
        //========================================
        else if(choose_system == 2)
        {
                xil_printf(”Testing 4g!\n\r”);
                INPUT_INTERFACE_AXI_mWriteReg(
                    XPAR_INPUT_INTERFACE_AXI_2_S00_AXI_BASEADDR
                    ,
                    INPUT_INTERFACE_AXI_S00_AXI_SLV_REG0_OFFSET
                    , INPUT_DATA_RATE_4G);
                setup_DMA2();
                xil_printf(”Done!\n\r”);
        }
        //========================================
        else if(choose_system == 3)
        {
                xil_printf(”Testing Wifi!\n\r”);
                INPUT_INTERFACE_AXI_mWriteReg(
                    XPAR_INPUT_INTERFACE_AXI_3_S00_AXI_BASEADDR
                    ,
                    INPUT_INTERFACE_AXI_S00_AXI_SLV_REG0_OFFSET
                    , INPUT_DATA_RATE_WIFI);
                setup_DMA3();
                xil_printf(”Done!\n\r”);
        }
        //========================================
        else if(choose_system == 4)
        {
                xil_printf(”Testing bluetooth!\n\r”);
                INPUT_INTERFACE_AXI_mWriteReg(
                    XPAR_INPUT_INTERFACE_AXI_4_S00_AXI_BASEADDR
                    ,
                    INPUT_INTERFACE_AXI_S00_AXI_SLV_REG0_OFFSET
                    , INPUT_DATA_RATE_BLUETOOTH);
                setup_DMA4();
                xil_printf(”Done!\n\r”);
        }
        //==========================================
        else
        {
                xil_printf(”No system!\n\r”);
        }
        //==========================================
break;
//==============================================
case '7':       // Exit
        Exit = 1;
        break;
//==============================================
default:
        break;
//==============================================
}
//==================================================
}
//======================================================
}
//==================================
}
```

الملخص

خلال السنوات القليلة الماضية تم استخدام تقنية إعادة التكوين الجزئي الديناميكي (DPR) على نطاق واسع، مما اتاح إعادة تشكيل حقل مصفوفات البوابات المنطقية (FPGA) خلال وقت التشغيل .يعتبر حقل مصفوفات البوابات المنطقية واحد من أفضل الحلول لتنفيذ الأجهزة القابلة لإعادة التكوين .مفهوم إعادة تكوين الأجهزة موجود منذ عدة عقود و مر بالعديد من مراحل التطور .باستخدام إعادة التكوين الجزئي الديناميكي، يمكن تطبيق نظام الراديو المعرف برمجيا (SDR) من أجل توفير القدرة والمساحة على نطاق واسع .على مدى السنوات القليلة الماضية، شهدت معايير الاتصالات اللاسلكية تطور كبير و سريع .السوق دائما ما يتطلب الحصول على معدل بيانات أعلى وخدمات خاصة أكثر .هذا يؤدي إلى زيادة تعقيدالتصميم، و زيادة المساحة، واستهلاك القدرة .تطبيق تقنية إعادة التكوين الجزئي الديناميكي على حقل مصفوفات البوابات المنطقية جعل من الممكن تصميم وتصنيع جميع معايير الاتصالات اللاسلكية على نفس الجهاز .تحميل كل المعايير حسب الطلب يقلل من المساحة المستخدمة واستهلاك القدرة.

نظام الراديو المعرف برمجيا هو نظام اتصال يتم تطبيق طبقته المادية ككتل برمجية .تم تصميم كتل الاتصال في أجهزة الإرسال والاستقبال اللاسلكية العادية في بيئة ثابتة لمعالجة شكل موجة معين .يمكن لنظام الراديو المعرف برمجيا معالجة العديد من أشكال الموجات نظرا لأنه يمكن تهيئته بسهولة باستخدام البرنامج .مع زيادة المرونة في إعادة تشكيل الواجهة الأمامية الرقمية أصبح من الممكن تحقيقه .واحدة من مزايا تصميم نظام الراديو المعرف برمجيا هي زيادة المرونة التي تساعد على إعادة التشكيل الديناميكي خلال وقت التشغيل .ميزة أخرى لتطبيق الراديو هي الاستخدام الفعال للموارد في ظل الظروف المختلفة .خلاصة القول هي أن مرونة الأجهزة تسمح لنظام الراديو المعرف برمجيا بتنفيذ معايير مختلفة خلال وقت التشغيل دون الحاجة إلى إيقاف تشغيل النظام .يتمثل التحدي الأساسي الذي يواجه تنفيذ نظام الراديو المعرف برمجيا في كيفية تحقيق قدرة حاسوبية كافية، خاصة في معالجة أشكال الموجات ذات معدل البيانات العالي، ضمن عوامل الحجم والوزن المقبولة، و ضمن تكاليف مقبولة للوحدة ومع استهلاك مقبول للقدرة.

يطبق هذا العمل نظام الإرسال والاستقبال للراديو المعرف برمجيا لخمسة معايير للاتصالات اللاسلكية: Bluetooth و Wi-Fi و 2G و 3G و LTE في مجموعة تقييم Zynq-7000. يتم استخدام تقنية اعادة التشكيل الجزئي الديناميكي الجديدة للتبديل بين أنظمة الاتصالات متعددة المعايير على نفس قسم حقل مصفوفات البوابات المنطقية .يجمع تطبيق الراديو المعرف برمجيا باستخدام تقنية اعادة التشكيل الجزئي الديناميكي بين مزايا أداء الأجهزة ومرونة البرامج .تم إنشاء بيئة اختبار لقياس فعالية التقنية الجديدة .قد تم تطبيق أسلوبين لتنفيذ أجهزة الإرسال والاستقبال الخمسة باستخدام تقنية اعادة التشكيل الجزئي الديناميكي .تستخدم التقنية الأولى قسما واحد قابلا لإعادة التشكيل للمرسل و المستقبل .التقنية الثانية توصي بتقسيم التصميم إلى عدة أقسام من أجل تحقيق أفضل أداء لجميع أجهزة الإرسال والاستقبال .تم إجراء مقارنة لمساحة النظام الكلية واستهلاك للقدرة في حالة تطبيق تقنية اعادة التشكيل الجزئي الديناميكي وحالة عدم استخدامه .يحقق نهج التقسيم الفردي تقليل للمساحة و القدرة بنسبة 10.19 ٪ و 76.71 ٪ على التوالي مع وقت تبديل معقول .بينما نهج متعدد التقسيم ,فهو قادر على تقليل المساحة المخصصة واستهلاك القدرة لجميع السلاسل .خفض القدرة في حالة ال G 2 و ال Bluetooth هو 95.43 ٪ ، اما بالنسبة لل G3 و ال Wi-Fi هو 79.69 ٪ ، و في حالة ال LTE هو 59.09 ٪ مقارنة مع حالة عدم استخدام التقنية.

مهندس: شريف محمد حسني عفيفي

تاريخ الميلاد: ١٩٩٣\٠٢\١٥

الجنسية: مصري

تاريخ التسجيل: ٢٠١٥/٠٣/٠١

تاريخ المنح: ٢٠١٨

القسم: هندسة الإلكترونيات والإتصالات الكهربية

الدرجة: ماجستير العلوم

المشرفون:

ا.د. أحمد حسين محمد خليل

د. حسن مصطفى حسن

الممتحنون:

أ.د أحمد حسين محمد خليل (المشرف الرئيسي)

أ.د. محمد فتحى ابواليزيد (الممتحن الداخلي)

أ.م.د. مجدي علي علي المرسي (الممتحن الخارجي)

(معهد بحوث الالكترونيات)

عنوان الرسالة:

تطبيق نظام الراديو المعرف برمجيا على اساس اعادة التكوين الجزئي الديناميكي

الكلمات الدالة:

نظام الراديو المعرف برمجيا, إعادة التكوين الجزئي الديناميكي, حقل مصفوفات البوابات المنطقية

ملخص الرسالة:

يطبق هذا العمل نظام الإرسال والاستقبال للراديو المعرف برمجيا لخمسة معايير للاتصالات اللاسلكية: Bluetooth و Wi-Fi و G2 و G3 و LTE في مجموعة تقييم Zynq-7000 . يتم استخدام تقنية اعادة التشكيل الجزئي الديناميكي الجديدة للتبديل بين أنظمة الاتصالات متعددة المعايير على نفس قسم حقل مصفوفات البوابات المنطقية. يجمع تطبيق الراديو المعرف برمجيا باستخدام تقنية اعادة التشكيل الجزئي الديناميكي بين مزايا أداء الأجهزة ومرونة البرامج. تم إنشاء بيئة اختبار لقياس فعالية التقنية الجديدة.

تطبيق نظام الراديو المعرف برمجيا على اساس اعادة التكوين الجزئي الديناميكي

اعداد

شريف محمد حسني عفيفي

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة
ماجستير العلوم
في
هندسة الالكترونيات والاتصالات الكهربية

يعتمد من لجنة الممتحنين:

أ. د. أحمد حسين محمد خليل	المشرف الرئيسى
أ. د. محمد فتحى ابواليزيد	الممتحن الداخلي
أ. م. د. مجدي علي علي المرسي (معهد بحوث الالكترونيات)	الممتحن الخارجي

كليـة الهندسـة - جامعـة القاهـرة
الجيـزة - جمهوريـة مصـر العربيـة
٢٠١٨

تطبيق نظام الراديو المعرف برمجيا على اساس اعادة التكوين الجزئي الديناميكي

اعداد
شريف محمد حسني عفيفي

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة
ماجستير العلوم
في
هندسة الالكترونيات والاتصالات الكهربية

تحت اشراف

اسم المشرف	اسم المشرف
أ.د. أحمد حسين محمد خليل	**د. حسن مصطفى حسن**
أستاذ	مدرس
قسم هندسة الالكترونيات والاتصالات الكهربية	قسم هندسة الالكترونيات والاتصالات الكهربية
كلية الهندسة - جامعة القاهرة	كلية الهندسة - جامعة القاهرة

كلية الهندسة - جامعة القاهرة
الجيزة - جمهورية مصر العربية
٢٠١٨

تطبيق نظام الراديو المعرف برمجيا على اساس اعادة التكوين الجزئي الديناميكي

إعداد

شريف محمد حسني عفيفي

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة
ماجستير العلوم
في
هندسة الالكترونيات والاتصالات الكهربية

كلية الهندسة - جامعة القاهرة
الجيزة - جمهورية مصر العربية
٢٠١٨