AUTOMATED PERFORMANCE-BASED DESIGN TECHNIQUE FOR AN EFFICIENT LTE PDSCH IMPLEMENTATION USING SDSOC TOOL

By

Mohammed Ahmed Abd Elhamid Eladawy

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
2019

AUTOMATED PERFORMANCE-BASED DESIGN TECHNIQUE FOR AN EFFICIENT LTE PDSCH IMPLEMENTATION USING SDSOC TOOL

By

Mohammed Ahmed Abd Elhamid Eladawy

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. Ahmed Hussein Mohamed	**Dr. Hassan Mostafa Hassan**
Professor of Electronics	Associate Professor of Electronics
A. H. Khalil ………………………………	………………………………
Department of Electronics	Department of Electronics
Faculty of Engineering, Cairo University	Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2019

AUTOMATED PERFORMANCE-BASED DESIGN TECHNIQUE FOR AN EFFICIENT LTE PDSCH IMPLEMENTATION USING SDSOC TOOL

By

Mohammed Ahmed Abd Elhamid Eladawy

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

A. H. Khalil
Prof. Ahmed Hussein Mohamed **Thesis Main Advisor**
Electronics Professor, Faculty of Engineering, Cairo University

Dr. Hassan Mostafa Hassan **Advisor**
Electronics Professor, Faculty of Engineering, Cairo University

Prof. Amin Mohamed Nassar **Internal Examiner**
Electronics Professor, Faculty of Engineering, Cairo University

Dr. Amr Talaat Abdel Hamid **External Examiner**
Associate Professor, Electronics/Networks Departments, German University in Cairo

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2019

Engineer's Name: Mohammed Ahmed Abd Elhamid Eladawy
Date of Birth: 21/09/1988
Nationality: Egyptian
E-mail: eladawy1988@gmail.com
Phone: +201030701720
Address: Giza/Egypt
Registration Date: 01/10/2012
Awarding Date: …./…./2019
Degree: Master of Science
Department: Electronics and Communications Engineering

Supervisors:

Prof. Ahmed Hussein Mohamed
Dr. Hassan Mostafa Hassan

Examiners:

Prof. Ahmed Hussein (Thesis main advisor) A.H.Khalaf
Electronics Professor, Faculty of Engineering, Cairo University

Dr. Hassan Mostafa Hassan (advisor)
Electronics Professor, Faculty of Engineering, Cairo University

Prof. Amin Mohamed Nassar (Internal examiner)
Electronics Professor, Faculty of Engineering, Cairo University

Dr. Amr Talaat Abdel Hamid (External examiner)
Associate Professor, Electronics/Networks Departments,
German University in Cairo

Title of Thesis:

AUTOMATED PERFORMANCE-BASED DESIGN TECHNIQUE FOR AN EFFICIENT LTE PDSCH IMPLEMENTATION USING SDSOC TOOL

Key Words:
SDSoC; Xilinx; SoC; LTE; PDSCH; FPGA

Summary:

This thesis demonstrates the work of the typical SDSoC design flow. In addition, a proposed algorithm design flow using SDSoC is developed to introduce new automated design techniques to design SoC on a heterogeneous FPGA-CPU platform based on performance metrics constraints such as area, power and latency. Design of Physical Downlink Shared CHannel (PDSCH) in Long-Term Evolution (LTE) is presented as a case study. The objective of this thesis is to realize the implementation of the LTE PDSCH transmitter and LTE PDSCH receiver using SDSOC tool and to select a platform that meets performance metrics constraints and to select the platform that achieves the best performance.

Disclaimer

I hereby 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 acknowledged all sources used and have cited them in the references Section.

Name: Date:

Signature:

Acknowledgments

I would like to express my appreciation to my thesis advisor Dr. Hassan Mustafa for his support and guidance. He has been a constant source of inspiration and has provided consistent support and valuable suggestion throughout this project without which this work would not have been possible.

I would like to express my heartfelt gratitude to my family for their encouragement and support without which I would not have come so far.

Finally, I would like to thank all my friends for their invaluable support and cooperation.

Dedication

“I'm not interested in how things were, or how we ended up where we are now. What interests me is what we are now and what we will be.”

Dr. Ahmed Khaled Tawfik

Table of Contents

ACKNOWLEDGMENTS I

DEDICATION IV

TABLE OF CONTENTS V

LIST OF TABLES VII

LIST OF FIGURES VIII

NOMENCLATURE X

LIST OF PUBLICATIONS XII

ABSTRACT XIII

CHAPTER 1 : INTRODUCTION 1

1.1. LONG TERM EVALUATION (LTE) 2
1.2. FIELD PROGRAMMABLE GATE ARRAY (FPGA) 3
1.3. SDSOC 3
1.4. ORGANIZATION OF THE THESIS 4

CHAPTER 2 : FPGA STRUCTURE AND LTE ARCHITECTURE 5

2.1. FPGA STRUCTURE 5
  2.1.1. Programmable input and output block unit 6
  2.1.2. Configurable logic block 7
  2.1.3. Digital clock management module 8
  2.1.4. Embedded block RAM 9
  2.1.5. Routing resource 9
  2.1.6. Embedded functional unit 10
2.2. LTE ARCHITECTURE 11
  2.2.1. LTE frame structure 12
  2.2.2. LTE PDSCH transmitter and receiver model 13
    2.2.2.1. CRC addition and CRC removing 13
    2.2.2.2. Segmentation and de-segmentation 15
    2.2.2.3. Channel coding and channel de-coding 19
    2.2.2.4. Interleaver and de-interleaver 24
    2.2.2.5. Rate matching and rate de-matching 24
    2.2.2.6. Scrambler and de-scrambler 27
    2.2.2.7. Modulator and de-modulator 28
    2.2.2.8. Resource element mapper and resource element de-mapper 32

CHAPTER 3 : SDSOC 34

3.1. SDSOC PRAGMA SPECIFICATION 34
  3.1.1. Function optimization 35
  3.1.2. Loop optimizations 35
  3.1.3. Array optimizations 36
  3.1.4. Interface management 37

3.2. TYPICAL HW/SW CO-DESIGN DESIGN FLOW USING SDSOC DESIGN FLOW 38
3.3. PROPOSED AUTOMATED PERFORMANCE-BASED DESIGN TECHNIQUE 41

CHAPTER 4 : IMPLEMENTION AND RESLULTS 47

4.1. IMPLEMENTATION OF TURBO ENCODER USING SDSCOC TOOL 47
  4.1.1. Turbo encoder block diagram 47
  4.1.2. Turbo encoder implementation 48
  4.1.3. Configurable Embedded FPGA Platform 49
  4.1.4. Results and Comparative Studies 50
    4.1.4.1. Hardware Utilization for turbo encoder implementation 50
    4.1.4.2. Dynamic power for turbo encoder implementation 51
    4.1.4.3. Hardware acceleration for turbo encoder implementation 52
    4.1.4.4. Figure of Merit for turbo encoder implementation 53
4.2. IMPLEMENTATION OF THE LTE PDSCH TRANSMITTER AND RECEIVER USING SDSOC TOOL 54
  4.2.1. LTE PDSCH transmitter implementation 54
    4.2.1.1. Configurable embedded FPGA platform for LTE PDSCH transmitter 55
  4.2.2. LTE PDSCH receiver implementation 56
    4.2.2.1. Configurable embedded FPGA platform for LTE PDSCH receiver 57
  4.2.3. Constraints solution 59
  4.2.4. Results and comparative studies 60
    4.2.4.1. LTE TX implementation results 60
      4.2.4.1.1. Hardware utilization for the LTE TX implementation 60
      4.2.4.1.2. Latency for the LTE TX implementation 61
      4.2.4.1.3. Dynamic Power for the LTE TX implementation 62
      4.2.4.1.4. Hardware acceleration for the LTE TX implementation 63
      4.2.4.1.5. FoM for the LTE TX implementation 64
    4.2.4.2. LTE RX implementation results 65
      4.2.4.2.1. Hardware utilization for the LTE T RX implementation 65
      4.2.4.2.2. Latency for the LTE RX implementation 66
      4.2.4.2.3. Dynamic Power for the LTE RX implementation 67
      4.2.4.2.4. Hardware acceleration for the LTE RX implementation 68
      4.2.4.2.5. FoM for the LTE RX implementation 69
    4.2.4.3. Constraints solution examples results 70

CONCLUSIONS 71

FUTURE WORK 73

REFERENCES 74

APPENDIX A: MODULATION LOOKUP TABLE 74

APPENDIX B: IMPLEMENT SDSOC PROJECT MANUALLY STEPS 82

APPENDIX C: SDSOC SHELL SCRIPT 87

List of Tables

Table 2.1: LTE standard different bandwidths 11
Table 2.2: Code block size for segmentations 16
Table 2.3: Turbo code interalver parameter 21
Table 2.4: Inter-column permutation matric for sub-block interleaver......25
Table 2.5: QBSK modulation lookup table 29
Table 2.6: 16QAM modulation lookup table...29
Table 2.7: Soft decision QPSK demodulation lookup table 31
Table 3.1: Mapping of C-code to RTL construct 34
Table 3.2: Three function configuration scenario example........................43
Table 3.3: Estimated performance list for selected combination 44
Table 3.4: Estimated performance list for all combination 45
Table 4.1: Turbo encoder sub-function configuration scenarios................48
Table 4.2: LTE PDSCH transmitter sub-function configuration scenario .55
Table 4.3: LTE PDSCH receiver sub-function configuration scenario......58
Table 4.4: Constraints solution examples...59
Table 4.5: Constraints solution examples results 71
Table C.1: Supported SDSoC evaluation board 88
Table C.2: Supported SDSoC operating system..88
Table C.3: synthesis frequency supported by SDSoC tool.........................88
Table C.4: Data motion operating frequency 89

List of Figures

Figure 2.1: Internal structure of the FPGA chip 6
Figure 2.2: Internal structure of IOB 7
Figure 2.3: Internal structure of slice...8
Figure 2.4: Structure of the LTE frame in FDD mode 12
Figure 2.5: LTE PDSCH transmitter and receiver model 13
Figure 2.6: Role of CRC block in transmitter and receiver........................14
Figure 2.7:Code block segmentation process 15
Figure 2.8: Code block De-segmentation 19
Figure 2.9: Turbo encoder block diagram 19
Figure 2.10: Turbo decoder block diagram 23
Figure 2.11: Rate matching block diagram 24
Figure 2.12: Rate de-matching block diagram 26
Figure 2.13: Constellation diagrams of modulation schemes 29
Figure 2.14: LTE resource grid 33
Figure 3.1: Memory interface management..37
Figure 3.2: FIFO Interface management 38
Figure 3.3: Typical HW/SW co-design flow using SDSoC 38
Figure 3.4: SDSoC design flow 39
Figure 3.5: Embedded FPGA platform 40
Figure 3.6: Performance-based design technique flow chart 42
Figure 4.1: Turbo encoder block diagram 47
Figure 4.2: Configurable embedded FPGA platform for turbo encoder 49
Figure 4.3: Hardware utilization for turbo encoder implementation..........50
Figure 4.4: Dynamic power for turbo encoder implementation 51
Figure 4.5: Hardware acceleration for turbo encoder implementation.......52
Figure 4.6: FoM for turbo encoder implementation 53
Figure 4.7: Configurable embedded FPGA platform for the LTE PDSCH transmitter 56
Figure 4.8: Configurable embedded FPGA platform for the LTE PDSCH receiver..57
Figure 4.9: Hardware utilization for the LTE TX implementation 61
Figure 4.10: Latency for the LTE TX implementation 62
Figure 4.11: Dynamic power for the LTE TX implementation 63
Figure 4.12: Hardware acceleration for the LTE TX implementation 64
Figure 4.13: FoM for the LTE TX implementation 65
Figure 4.14: Hardware utilization for the LTE RX implementation..........66
Figure 4.15: latency for the LTE RX implementation 67
Figure 4.16: Dynamic power for the LTE RX implementation 68
Figure 4.17: Hardware acceleration for the LTE T RX implementation ..69

Figure 4.18: FoM for the LTE RX implementation 70
Figure B.1: Project naming and platform selection and OS selection........82
Figure B.2; Empty project Selection 83
Figure B.3: project window in estimated performance mode 84
Figure B.4: Estimated performance results 84
Figure B.5: Window project in non-estimated performance mode 85
Figure B.6: SDOSC directory structure...85

Nomenclature

3GPP	3rd Generation Partnership Project
AMC	Adaptive Modulation and Coding
ASIC	Application-Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
B.W	Bandwith
BCJR	Bahl-Cocke-Jelinek-Raviv
CAM	Content Address Memory
CLB	Configurable Logic Block
CPLD	Complex Programmable Logic Device
CPU	Central Processor Unite
CRC	Cyclic Redundancy Check
DCM	Digital Clock Management
EEPROM	Electrically Erasable Programmable Read-Only Memory
FDD	Frequency Division Duplex
FFT	Fast Fourier Transform
FoM	Figure of Merit
FPGA	Field Programmable Gate Array
GAL	Generic Array Logic
HLS	High Level Synthesis
IOB	Input-Output Block
IoT	Internet of Things
LCA	Logic Cell Array
LTE	Long-Term Evolution
LUT	Look-Up Table
MAP	Maximum A Posteriori
MIMO	Multiple Input Multiple Output
NB-IoT	Narrowband IoT
OFDM	Orthogonal Frequency Multiplexing
PAL	Programmable Array Logic
PBCH	Physical Broadcast Channel
PCFICH	Physical Control Format Indicator Channel
PDCCH	Physical Downlink Control Channel
PDR	Partial Dynamic Reconfiguration
PDSCH	Physical Downlink Shared CHannel

PDSCH	Physical Downlink Shared CHannel
PHICH	Physical Hybrid Indicator Channel
PMC	Performance Metrics Cost
PMCH	Physical Multicast Channel
PROM	Programmable Read Only Memory
PSS	Primary synchronization signals
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
RSC	Recursive Systematic Convolution
RTL	Register Transfer Level
SDK	Software Development Kit
SDSoC	Software Defined System-on-Chip
SISO	Single Input Single Output
SNR	Signal to Noise Ratio
SoC	Systems on Chip
SSS	Secondary Synchronization Signals
TDD	Time Division Duplex

LIST OF PUBLICATIONS

1. Mohamed Eladawy, Ahmed Kamal, Hassan Mostafa, Sameh Said, “Performance Evaluation of Turbo Encoder Implementation on a Heterogeneous FPGA-CPU Platform Using SDSoC”, International Conference on Advanced Control Circuits Systems (ACCS) Systems & International Conference on New Paradigms in Electronics and Information Technology (PEIT), 2017.

2. Mohamed Eladawy, Mahmoud Kishky, Hassan Mostafa, Sameh Said, “Automated Performance Based Design Technique for an efficient LTE PDSCH Implementation Using SDSoC Tool”. International Journal of Communication Systems (IJCS), 2018.

Abstract

Systems on Chip (SoC) creates massive design challenges for the SoC-based designers. The design challenges start from functional complexity, architectural design, verification tests and finally meeting performance constraints. Furthermore, the heterogeneity of components and tools introduce long and large design cycles. In addition, the hardware-software co-design includes complicated design process. The design complexity includes the following: first, co-specification, where the roles of software and hardware in implementing system functionality are specified and the implementation is assigned to either software or hardware. Second, co-development, where the software, hardware, and interfaces are developed. Third, co-verification where the optimization and refining of the SW/HW components are performed to meet the design constraints.

The Software Defined System on Chip (SDSoC) tool is developed by Xilinx to create custom SoC on a heterogeneous FPGA-CPU platform. The SDSoC tool offers a fast and short design cycle for heterogeneous FPGA-CPU platform development. The SDSoC tool also integrates multiple tools to make the co-design of the hardware-software more flexible.

In this thesis, the typical SDSoC design flow is presented. In addition, the thesis provides a new automated SDSoc design technique to design SoC on a heterogeneous FPGA-CPU platform based on performance metrics such as area, power ...etc. The new design technique used to explore the performance metrics for all possible combination between software implementation and hardware-accelerated implementation for "n” functions. Moreover, the new design technique used to determine platforms that achieve performance metrics and to select the platform that achieves the best overall performance.

As a case study, design of Physical Downlink Shared CHannel (PDSCH) in Long-Term Evolution (LTE) is employed. The architecture of transmitter and receiver of the LTE PDSCH are studied and the LTE PDSCH transmitter/receiver software functions are written using C programming language.

The objective of this thesis is to implement the LTE PDSCH transmitter functions and the LTE PDSCH receiver functions using SDSoC tool, to select the platform that meets performance metrics constraints, and to select the platform that achieves the best overall performance.

Chapter 1 : Introduction

Smart homes, automated vehicles, and Internet of Things (IoT) are examples of electronic products that are almost involved in every aspect of our lives. The rapid growth of the electronics industry encourages developers to find faster and efficient design methods to decrease time-market requirements for design and system development cycle [1].

During the last few years, the electronics industry shifted from the Application Specific Integrated Circuit (ASIC) to SoC design production to pursue the large production of electronic devices [2]. The SoCs designer should build a product with an efficient architecture that is a key to ensure that system design meets its performance requirements. Design an efficient architecture might consume a lot of time, cost and a very long and complicated design cycle.

Besides that, hardware/software partitioning method was introduced by Edwards and Forrest in reference [3]. The main objective of hardware/software codesign method is to produce systems containing an optimum balance of hardware and software components which work together to achieve a specified behavior and fulfill specified design constraints. Hardware/software codesign examine the parallel method to design hardware and software components of complex electronic systems [4]. Hardware/software codesign method tries to achieve the corporation of hardware and software with the goal to optimize constraints such as power, area, ..., etc. and it targets to reduce the time-to-market frame significantly.

Vista™ platform is a tool introduced by Mentor graphic company used in hardware/software codesign [5]. Vista Virtual Prototyping provides an early functional model of the hardware to software engineers even before the hardware design is implemented in RTL. It can run software on embedded processor models at speeds par with physical hardware boards, providing sufficiently fast simulation models for OS and application software validation. The features included in the tool are as follows: Vista tool provides architecture design and exploration, Vista tool allows hardware/software tradeoffs analysis, Vista tool supports an early assessment of and finally Vista tool includes a virtual platform for software integration and validation.

Xilinx announces SDSoC tool target developing SoC on a heterogeneous FPGA-CPU platform [6]. In this thesis, hardware-software co-design of LTE PDSCH transmitter and receiver synthesized by SDSoC for heterogeneous FPGA-CPU platform is proposed.

1.1. LONG TERM EVALUATION (LTE)

Connecting to the Internet wirelessly through a cell phone is one of the greatest technological innovations of the last decade of years. Wireless communication is an old idea that began with Morse signals down to the fourth-generation technology. The fast growth of mobile phone markets promises the 3rd Generation Partnership Project (3GPP) to develop the Long-Term Evolution (LTE) standard for high-speed wireless communication for mobile devices [7].

LTE is a mobile communication standard is developed to improve the mobile phone standard to follow up with future technology evolutions. The LTE network supported by a number of key technologies including Orthogonal Frequency Multiplexing (OFDM) [8], multi-carrier modulation technology, Adaptive Modulation and Coding (AMC) technology, Multiple Input Multiple Output (MIMO) and smart antenna technology [9]. The LTE standard supports users and telecommunication companies' requirements [10].

The requirements include reducing cost per bit use of current and new frequency bands, simplification architecture, providing more services at lower cost and reasonable power consumption. The objectives of the LTE standard are increasing efficiency spectrum utilization, improving system capacity and increasing data rate up to 100 Mbps.

The LTE mobile communication system has the following features:

- Faster transfer rate: the LTE supports data rate up to 2Mbps for large-scale high-speed mobile users (250km/h), 20Mbps for medium-speed mobile users (60km/h), and 100Mbps for low-speed mobile users (indoor or pedestrian) [9].
- Efficient spectrum utilization: the LTE uses many powerful breakthrough technologies in the development process. The use of wireless spectrum is much more efficient than the second and third generation systems, and the speed is quite fast.
- Wider network spectrum: each LTE channel occupies 100MHz or more bandwidth, while the bandwidth of the 3G network is between 5~20MH [10].
- More flexibility: the LTE system adopts intelligent technology that used to adapt allocation resources and adopt intelligent signal processing technology to transmit and receive signals in various complex environments with different channel conditions.

- Higher quality multimedia communication: the LTE network supports multimedia communication services include voice, data, video, …etc. A large amount of information is transmitted through the broadband channel allowing users to connect to the system at any time and any location.

- Smoother compatibility: the LTE systems have global roaming, open interfaces, interconnection with multiple networks, diversification of terminals and a smooth transition from the second generation.

1.2. FIELD PROGRAMMABLE GATE ARRAY (FPGA)

FPGA is a product of further development based on programmable devices such as Programmable Array Logic (PAL), Generic Array Logic (GAL) and Complex Programmable Logic Device (CPLD) [11]. The FPGA has powerful processing functions and complete design freedom so that its industry rival ASIC designers use FPGAs to simulate the entire system at board level before they manufacture wafers. FPGA proposes an effective electronic design by integrating board design, programmable logic design, and software development.

Since the birth of FPGAs, electronic product design evolves into a programmable logic design and embedded software design. At the same time, electronic design shifted to be more of a "soft" design [12]. The design through the development of language and tools and FPGA become this a "soft" design carrier.

Availability of low-cost and large-scale programmable device in the form of an FPGA makes it possible for designers to transfer all system core functions to a soft design and take advantage of the soft design. These "soft" design advantages include: easier to protect system functions from being copied or reverse engineered, and easier to modify the functions architecture. For that reason, "soft" design becoming the development direction of the electronic design.

1.3. SDSOC

The fast-growing of IoT devices promises electronic devices developers to find a rapid solution for developing IoT products to the markets [13]. Xilinx announces Software Defined System-on-Chip (SDSoC) tool target developing IoT products and creates custom SoC on a heterogeneous FPGA-CPU platform.

The SDSoC tool provides a short design cycle to develop heterogeneous FPGA-CPU platform with simple interface logic generated by the tool to handle the data flow between hardware and software. In addition, SDSoC supports estimating performance, hardware utilization and latency calculations that make developing design short and fast [14].

1.4. ORGANIZATION OF THE THESIS

Section 2 divided into two topics. First, section 2.1 about the FPGA structure and working principle. Second, section 2.2 about the LTE and illustrates in details the architecture of the LTE PDSCH transmitter and receiver chain used in this thesis.

Section 3 about SDSoC tool and explains the typical design flow using SDSoC and the proposed automated performance-based adaptive design technique.

Section 4 shows the implementation and comparative studies results. The implementation of turbo encoder is presented as a case study to test the SDSoC tool and to explore the different performance metrics used for design. In addition, the implementation of the LTE PDSCH transmitter and receiver using SDSoC tool is presented as a case study for the new design technique developed in this work.

In conclusion, the main achievements are highlighted together with the future lines of this work

Chapter 2 : FPGA STRUCTURE AND LTE ARCHITECTURE

2.1. FPGA STRUCTURE

The first generation of programmable devices is familiar with Programmable Read Only Memory (PROM), Erasable Programmable Read Only Memory (EPROM) and Electrically Erasable Programmable Read-Only Memory (EEPROM) [15]. The programmable principle of these programmable devices is to change the carrier density inside the triode or MOS transistor by applying a high voltage or ultraviolet light. These devices are called programmable, but it is difficult to achieve single-programmable or programmable state.

FPGA is different because it adopts a new concept such as Logic Cell Array (LCA), which includes Configurable Logic Block (CLB), Input-Output Block (IOB) and Interconnect [16]. The programming of the FPGA changes the trigger state of the CLB and IOB, so that multiple repeated programming is realized. Most FPGAs use a Look-Up Table (LUT) structure based on SRAM technology, and some military and aerospace-class FPGAs use Flash or fuse and anti-fuse process look-up table structures. The repeated configuration of the FPGA is accomplished by programming the file to change the contents of the lookup table.

The designer uses different programming methods according to different configuration modes. The design of FPGAs are based on the SRAM process and need to connect an off-chip memory to save the program. At power-on, the FPGA reads the data in the external memory into the on-chip RAM. After the configuration is completed, FPGA enters the working state. After turned off the power of FPGA, the FPGA returns to the white chip, and the internal logic disappears.

In this way, the FPGAs are programmed repeatedly without a dedicated FPGA programmer. However, the design of some FPGAs are based on anti-fuse technology FPGAs, which have the advantages of radiation resistance, high and low-temperature resistance, low power consumption and high speed. They are widely used in military and aerospace applications, but such FPGAs are not repeatedly erased [17].

Figure 2.1 shows the main FPGA components which consist of the following parts: programmable Input and Output Block (IOB) unit, Configurable Logic Block (CLB), Digital Clock Management (DCM) module, embedded Block RAM (BRAM), Switch Matrix (SW) for routing, and embedded functional unit [18].

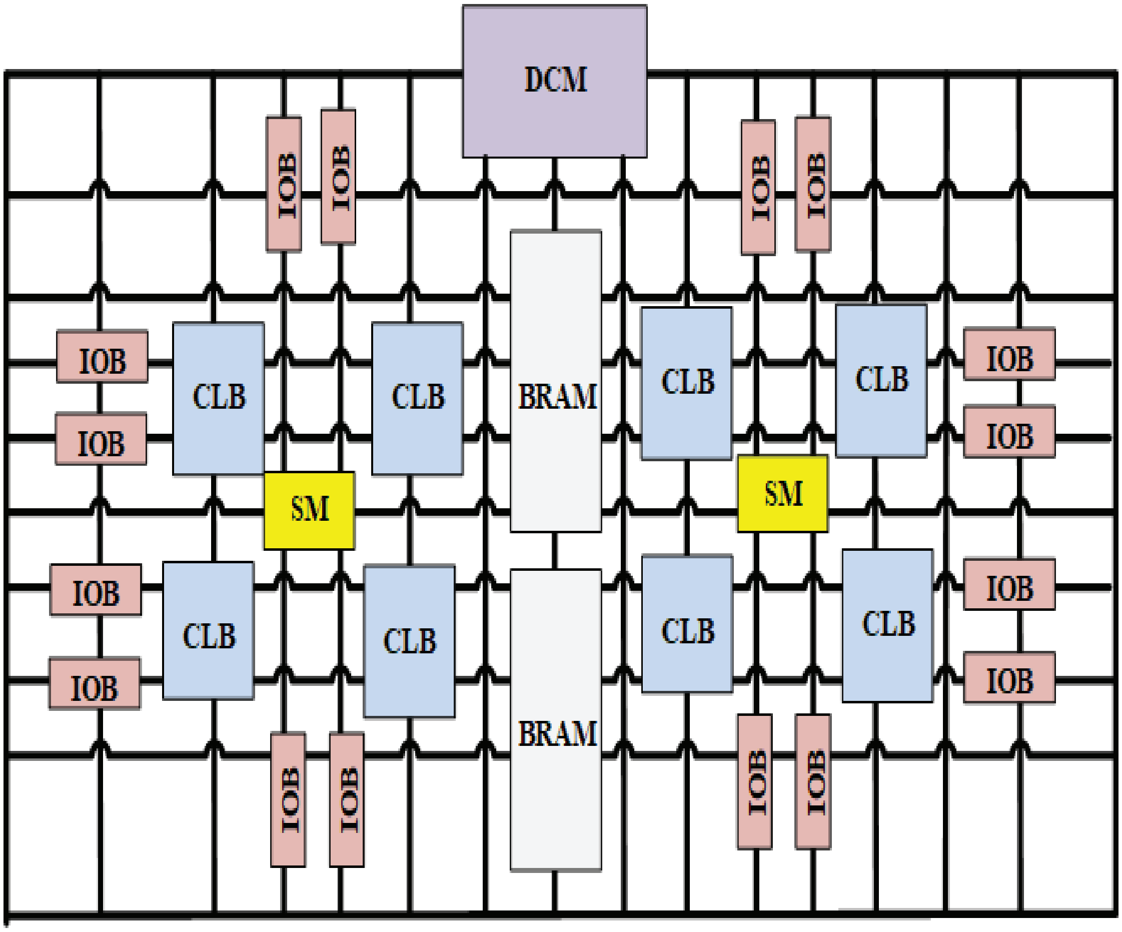


Figure 2.1: Internal structure of the FPGA chip [18]

2.1.1. Programmable input and output block unit

The programmable Input/Output Block (IOB) unit is the interface part between the chip and the external circuit. It performs the driving and matching requirements of the input/output signals under different electrical characteristics. The schematic structure of IOB is shown in Figure 2.2 [19].

The IOBs are designed to have a flexible software configuration and different electrical standards [19]. The IOBs driving current is adjusted and the upper and lower pull-down resistors are changed according to software configuration

In order to facilitate management and adaption to various device standards, IOBs within the FPGA are designed in groups, and each group supports different IOB standard independently. The IOB of the FPGA is divided into several banks. The interface standard of each bank is determined by its interface supported voltage level. The voltage level is depending on the FPGA generation and is coming down as new generations come. The different voltage level supported by different FPGA generations includes 5V, 3.3V, 2.5V, 1.8V, 1.5V and 1.2.V [18].

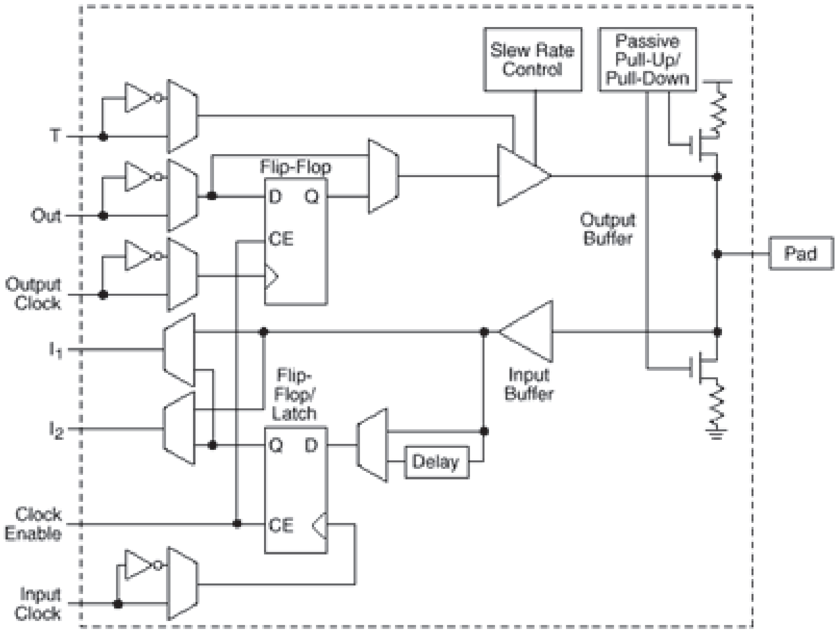


Figure 2.2: Internal structure of IOB [19]

2.1.2. Configurable logic block

CLB is the basic logical unit within the FPGA. The actual number and characteristics of CLBs vary from device to device. In Xilinx's FPGA devices, the CLB consists of multiple identical slices and additional logic. The CLB modules not only used to implement combinatorial logic, timing logic but also as distributed RAM and distributed ROM. Slice is the basic logical unit defined by Xilinx company. The internal structure of the slice is shown in Figure 2.3. A slice consists of two 4-input functions LUT, carry logic, arithmetic logic, storage logic and function multiplexer [19].

LUT is typically viewed a RAM. For example, 4 input LUTs are used in FPGAs viewed as a RAM with a 4-bit address line. When the user describes a logic circuit through the schematic or HDL language, the FPGA development software automatically calculates all possible results of the logic circuit and writes the truth table to the RAM in advance. So, performing a logical operation is equivalent to inputting an address to look up the table, finding out the content corresponding to the address, and then outputting it.

The LUT has the same function as the logic circuit. However, LUTs have faster execution speeds and larger scales. FPGAs device densities of LUT range from tens of thousands to tens of millions of gates, allowing extremely complex times sequence and logic combine logic circuit functions, so it is suitable for high-speed, high-density high-end digital logic circuit design.

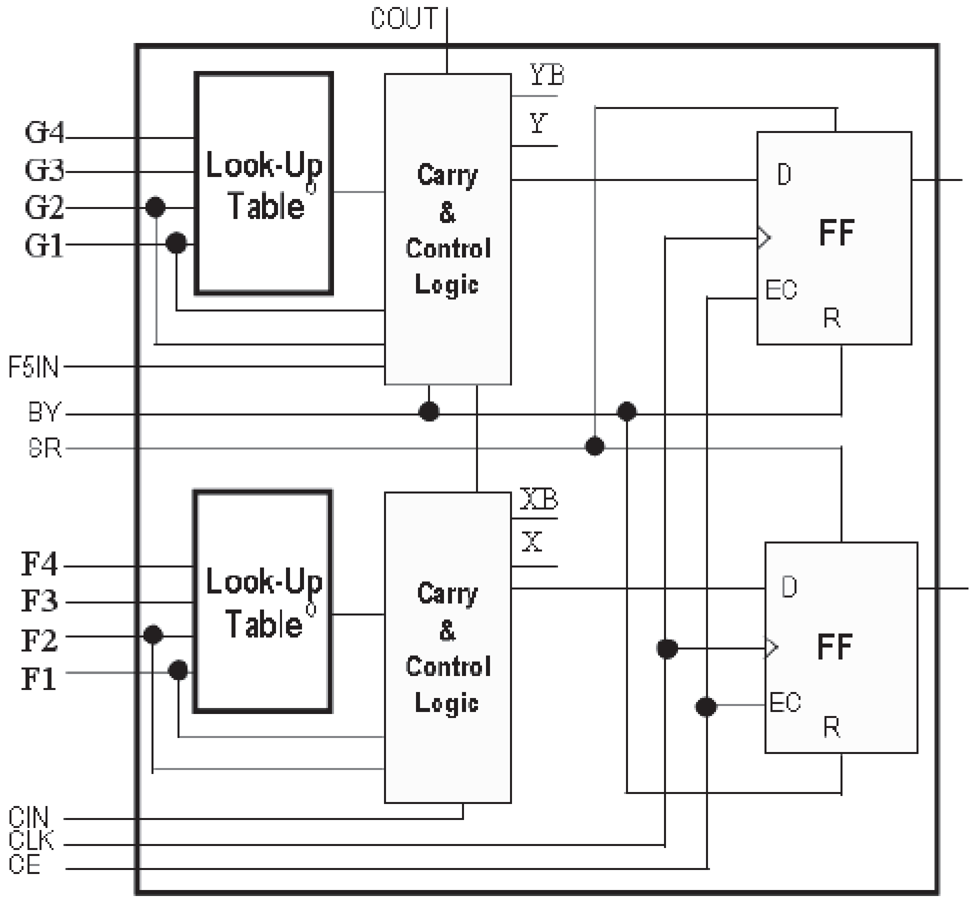


Figure 2.3: Internal structure of slice [19]

2.1.3. Digital clock management module

Digital clock management (DCM) is used on FPGAs for dealing with all aspects of clock management [19]. Xilinx FPGAs have different DCM circuit implementation includes a digital phase shifter, digital frequency synthesizer and a delay-locked loop (DLL). DCM supports advanced clocking capabilities for multiplying or dividing the incoming clock frequency to synthesize a new clock. DCM also eliminates clock skew to improve the system performance and is able to phase output clock shift to delay the incoming clock by a fraction of the clock period.

2.1.4. Embedded block RAM

FPGAs have embedded block RAM (BRAM), which expands the range and flexibility of FPGA applications hugely. BRAMs are used every time you need a bunch of data to be stored on a chip. BRAMs are dedicated ram that does not use any additional LUT in your design. The Block RAM is used as the following configuration: A common storage structure such as single-port RAM, dual-port RAM, Content Address memory (CAM), and FIFO. The amount of BRAM inside the chip is also an important factor in selecting an FPGA chip [20].

2.1.5. Routing resource

The Switch Matrices (SM) routing resources (SM) is used to connect all the cells inside the FPGA together. The length and process of the wires determine the driving capability and transmission speed of the signals on the wires [21]. The FPGA chip has a wealth of routing resources and is divided into four different categories according to the process, length, width, and distribution.

- The first type is the global routing resource, which is used for the internal global clock of the chip and the global reset/set wiring.
- The second type is the long-line resource, which is used to complete the wiring of the high-speed signal between the chip Bank and the second global clock signal.
- The third type is the short-term resources, which is used to complete the logical interconnection and routing between basic logic cells.
- The fourth type is the distributed routing resources, which is used for control signals such as proprietary clocks, resets … etc.

The designer does not need to select the routing resources directly. The place and route router select the routing resources to connect the various module units according to the topology and constraints of the input logical network table automatically.

2.1.6. Embedded functional unit

The FPGAs chip manufacturers have integrated specialized hard cores inside the chip to improve the FPGA performance [22]. The hardcore has powerful FPGA processing is equivalent to the ASIC circuit. For example, dedicated multipliers are integrated into FPGAs to increase the multiplication speed of FPGAs. Many high-end FPGAs integrate serial-to-parallel transceivers reach tens of Gbps, which used to implement communication bus and interface standards.

In addition, the new Xilinx FPGA generations consist of built-in PCI Express and a Tristate Ethernet MAC hardcore (TEMAC) [18]. The Xilinx Tri-Mode Ethernet MAC core is a parameterize core that is ideal for use in network equipment such as switches and routers. The customizable TEMAC core enables system designers to implement a wide range of integrated Ethernet designs, from low-cost 10/100 Ethernet to higher-performance 1GB ports. The TEMAC core design is compliant with the IEEE 802.3 specification and operate in 1000Mbps, 100 Mbps, and 10 Mbps modes. In addition, it supports half-duplex and full-duplex operation.

Xilinx has not only integrated specialized hard cores but also Power PC series CPUs such ARM. Through the platforms such as ARM, it is possible to develop standard DSP processors and related applications to achieve the development goals of SOC.

The hard-core refers to the netlist with planning information in the FPGA design. Hard-core views as a soft-core with layout planning. A mixture of RTL code and corresponding specific process netlist provides hard-core. The RTL description is combined with a specific standard cell library to form a gate-level netlist, then gate-level netlist used by the place-and-route tool [23]. The advantages of hard-core are high flexibility and portability. The disadvantage of hard-core is that the predictability of the module is low, there is a possibility of error in the subsequent design, and there is a certain design risk.

On the other hand, the soft-core refers to the pre-integration register transfer level (RTL) model. In the FPGA design, the soft-core is the hardware language description of the circuit, including logical descriptions, netlists, and help documentation. The soft-core is only functionally simulated and needs to be integrated and laid out to be used.

Comparing the hard-core with the soft-core, the design flexibility of the solid core is slightly worse, but the reliability is greatly improved. Comparing the hardcore with the soft-core implementation, the hardcore reduces the power consumption by 5~10 times, saving nearly 90% of the logical resources [24].

2.2. LTE ARCHITECTURE

LTE is a mobile communication developed by 3GPP to improve the mobile phone standard to follow up with future technology evolutions and needs. The LTE standard supports users and telecommunication companies' requirements. The requirements include the following: use of current and new frequency bands, simple architecture, increase service provisioning more services at lower cost, reduce cost per bit and reasonable power consumption.

The LTE radio transmission and reception specifications are described in reference [25] for the User Equipment (UE) and in reference [26] for the Base Station (BS). Downlink and uplink transmission in LTE are based on the use of multiple access technologies, Orthogonal Frequency Division Multiple Access (OFDMA) is used for the downlink and single-carrier frequency division multiple access

The bandwidths defined by the standard are 1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 15 MHz, and 20 MHz [27]. Table 2.1 shows how many subcarriers and resource blocks there are in each bandwidth.

Table 2.1: LTE standard different bandwidths [27]

Channel B.W (MHz)	1.4	3	5	10	15	20
Number of RB	6	15	25	50	75	100
Number of SC	72	180	30	600	900	1200
FFT/IFFT length	128	256	512	1024	1536	2048
Sample rate (MHz)s	1.92	3.84	7.68	15.36	23.04	30.72

The LTE standard defines six downlink channels, three channels for controlling information and three channels for carrying user data [28].

- The control channels are Physical Hybrid Indicator Channel (PHICH), Physical Control Format Indicator Channel (PCFICH) and Physical Downlink Control Channel (PDCCH).
- The data channels are Physical Broadcast Channel (PBCH), Physical Multicast Channel (PMCH) and Physical Downlink Shared Channel (PDSCH).

This thesis focuses only on the design of the PDSCH channel because this is the LTE channel carrying user data and processing it.

2.2.1. LTE frame structure

The LTE physical frame structure defines two types of a frame structure in the 3GPP standard [29], the Frequency Division Duplex (FDD) type and Time Division Duplex (TDD) type. The FDD-LTE is one of the duplex technologies used in the LTE mobile communication where the uplink and downlink are distinguished by different frequency points.

The FDD-LTE mode is characterized by receiving and transmitting on two separate symmetric frequency channels, and separating the receiving and transmitting channels with a guaranteed frequency band. In the FDD-LTE system, the uplink and downlink frequency intervals reach 190MHz. The FDD-LTE uplink theoretical rate is up to 40Mbps, and the downlink theoretical rate is 150Mbps [10].

Figure 2.4 shows the structure of the LTE frame in the FDD Mode [30]. The frame has 10 ms duration and each frame consists of 10 sub-frames has 1 ms duration. Each sub-frame consists of two slots has 0.5 ms duration. Each slot consists of OFDM symbols depending on the type of Cycle Prefix (CP) of each slot. Each slot consists of either 7 symbols for normal CP or 6 symbols for extended CP.

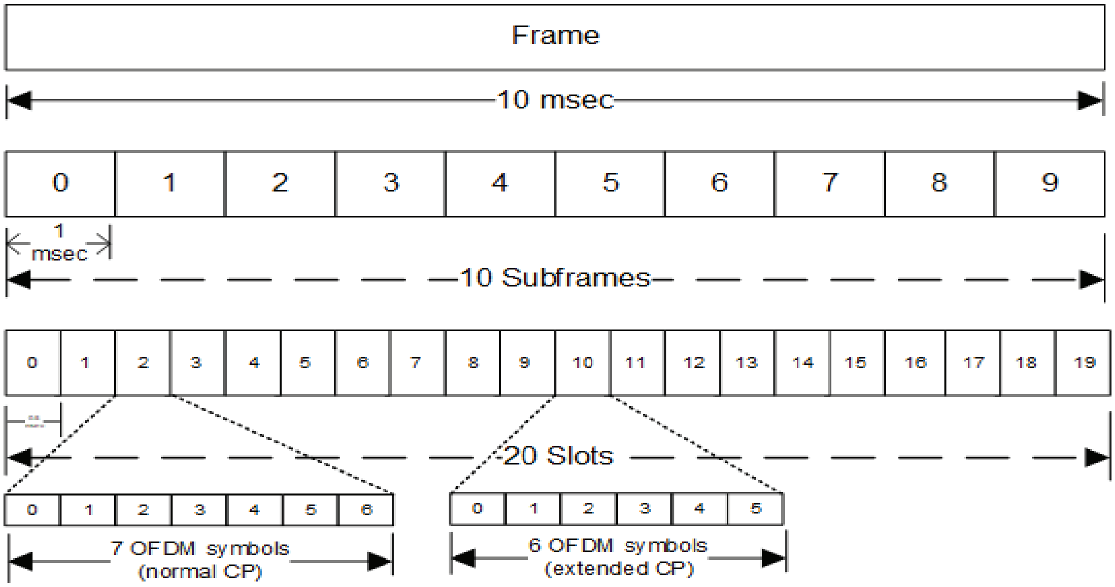


Figure 2.4: Structure of the LTE frame in FDD mode [30]

2.2.2. LTE PDSCH transmitter and receiver model

The PDSCH is a physical channel that carries user data. The transmitter and the receiver model of Single Input Single Output (SISO) PDSCH chain [31] is shown in figure 2.5. The PDSCH chain model is tested over Additive White Gaussian Noise (AWGN) channel for high Signal to Noise Ratio (SNR).

The data input to the LTE PDSCH transmitter chain is called a transport block. The transport block flow goes into Cyclic Redundancy Check (CRC) calculation and appending, segmentation, turbo encoding, interleaving, rate matching, scrambling and modulation followed by the resource element mapper. The data from the transmitter is passed to channel, and then it is fed to the LTE PDSCH receiver.

The received input flow goes into the resource element de-mapper, demodulation, de-scrambling, rate de-matching, de-interleaving, turbo decoding and de-segmentation followed by CRC calculation and extraction.

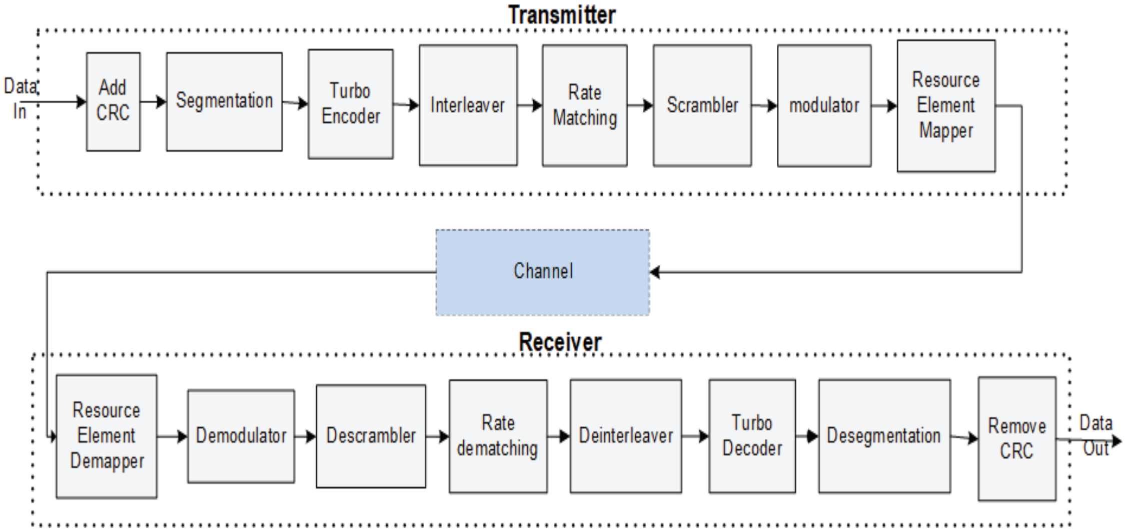


Figure 2.5: LTE PDSCH transmitter and receiver model [31]

2.2.2.1. CRC addition and CRC removing

CRC is a sequence of redundant bits used for error detection on transport blocks. CRC parity bits are calculated and appended to the transport block. The CRC parity 24A is calculated using a CRC generator polynomial [32].

The equation for CRC24A polynomial generation is as follows:

$$g_{CRC24A}(\mathrm{D}) = 1 + D^1 + D^3 + D^4 + D^5 + D^6 + D^7 + D^{10} + D^{11} + D^{14} + D^{17} + D^{18} + D^{23} + D^{24} \qquad \mathbf{(2.1)}$$

Let assume input bits to the CRC addition are as follows: -

$$Input\_crc_{bits} = a_0\ , a_1\ , a_2\ , a_3\ \ldots\ldots\ldots a_{N-1} \qquad \mathbf{(2.2)}$$

Let assume the parity bits appended to the input bits are as follows: -

$$Parity_{bits} = p_0\ , p_1\ , p_2\ , p_3\ \ldots\ldots\ldots p_{L-1} \qquad \mathbf{(2.3)}$$

Where N is the size of the input sequence bits and L is the length of the parity bits.

The encoding is performed in a systematic form as follows: -

$$poly_{equ}\colon a_0 D^{23} + a_1 D^{22} + \cdots + a_{\mathrm{N}-1} D^{24} + p_0 D^{23} + p_1 D^{22} + \cdots + p_{22} D^1 + p_{23} \qquad \mathbf{(2.4)}$$

The CRC ($Parity_{bits}$) are appended to the end of the data bits, so the length of input bits after CRC appending is (N+24) because CRC24A is used.

On the receiver side, the CRC polynomial is generated using the same CRC polynomial generator equation 2.1. As shown in Figure 2.6, The CRC is checked for any error in the received bits, if no error, remove the CRC from the transport block else if any error is found in a particular block, that transport block is retransmitted [33].

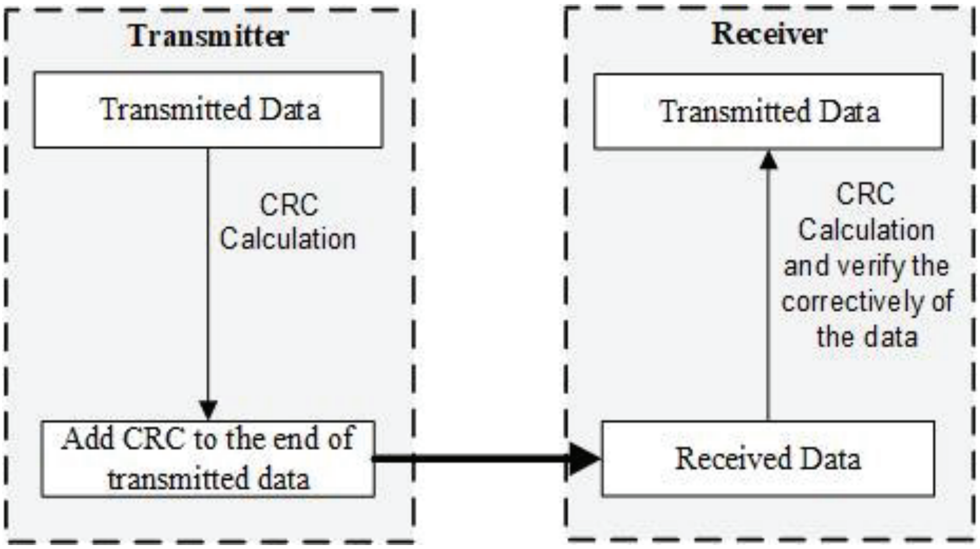

Transmitter
Transmitted Data
CRC Calculation
Add CRC to the end of transmitted data
Receiver
Transmitted Data
CRC Calculation and verify the correctively of the data
Received Data

Figure 2.6: Role of CRC block in transmitter and receiver

2.2.2.2. Segmentation and de-segmentation

Large amount of data bits from transport block should be transmitted at the same time. The transport block is divided into smaller blocks called code blocks as shown in figure 2.7. The LTE standard defines the minimum code block size is 40 bits and the maximum code block size is 6144 bits [32].

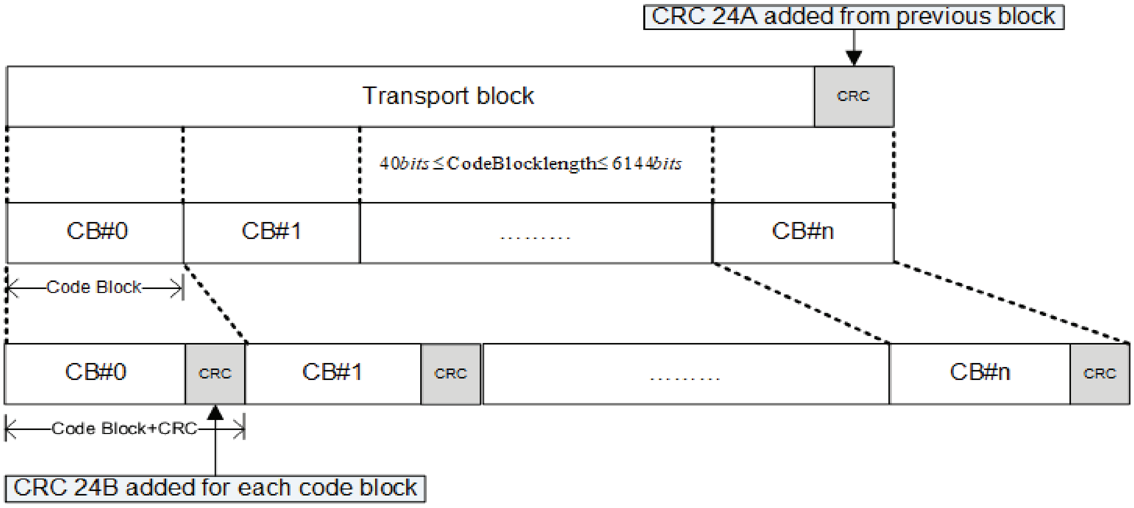

CRC 24A added from previous block
Transport block
CRC
40bits ≤ CodeBlocklength ≤ 6144bits
CB#0
CB#1
....
CB#n
Code Block
CB#0
CRC
CB#1
CRC
....
CB#n
CRC
Code Block+CRC
CRC 24B added for each code block

Figure 2.7: Code block segmentation process

If input bits to the code block segmentation are less than 40 bits, add filling bits. If input bits to the code block segmentation are larger than 6144 bits, perform segmentation of input bits and append other 24 bits CRC of type 24B to each of the code blocks. Table 2.2 shows the code block sizes defined by the LTE 3GPP standard.

To calculate the number of code blocks, let assume the input bits sequence are as follows:

$$Input\_seg_{bits} = b_0\ , b_1\ , b_2\ , b_3\ \dots\dots\dots b_{B-1} \qquad (2.5)$$

where B is the input block size.

Assume Z is the maximum code block size equal to 6144 and F is the number of filler bits where filler bits are added to the beginning of the first block if necessary. Also, assume L is the number of CRC bit equal to 24 and B_{new} is the new size of the input block after CRC addition.

Table 2.2: Code block size for segmentations [32]

40	48	56	64	72	80	88	96	104	112
120	128	136	144	152	160	168	176	184	192
200	208	216	224	232	240	248	256	264	272
280	288	296	304	312	320	328	336	344	352
360	368	376	384	392	400	408	416	424	432
440	448	456	464	472	480	488	496	504	512
528	544	560	576	592	608	624	640	656	672
688	704	720	736	752	768	784	800	816	832
848	864	880	896	912	928	944	960	976	992
1008	1024	1056	1088	1120	1152	1184	1216	1248	1280
1312	1344	1376	1408	1440	1472	1504	1536	1568	1600
1632	1664	1696	1728	1760	1792	1824	1856	1888	1920
1952	1984	2016	2048	2112	2176	2240	2304	2368	2432
2496	2560	2624	2688	2752	2816	2880	2944	3008	3072
3136	3200	3264	3328	3392	3456	3520	3584	3648	3712
3776	3840	3904	3968	4032	4096	4160	4224	4288	4352
4416	4480	4544	4608	4672	4736	4800	4864	4928	4992
5056	5120	5184	5248	5312	5376	5440	5504	5568	5632
5696	5760	5824	5888	5952	6016	6080	6144		

The total number of code blocks C are defined by the following Pseudo-code:

List 2.1: Pseudo-code of segmentation to determine number of code block [32]

```
$if\ (B < Z)$
$L = 0$
$C = 1$
$B_{new} = B$
$else$
 $L = 24$
$C = \frac{B}{Z-1}$
$B_{new} = B + C * L$
$endif$
```

From pseudo-code shown in list 2.1, if the block sizes less than 6144 bits, there is no need for segmentation process. On the other hand, if B is larger than 6144 bits, segmentation is applied and a CRC sequence is appended to each code block segment [34].

Let assume the output from code block segmentation are

$$Output\_seg_{bits} = c_0\ , c_1\ , c_2\ , c_3\ \ldots\ldots\ldots c_{r(K-1)} \qquad \textbf{(2.6)}$$

Where r is the code block number and K is the number of bits for code block .

The number of bits in each code blocks (K) are defined (for C ≠ 0 only) by the following pseudo code:

List 2.2: Number of bits in each code blocks calculation Pseudo-code [34]

```
$K_+ =\ the\ minmum\ value\ of\ k\ from\ table\ 2$ such that $C * K \geq B_{new}$
$if\ (C = 1)$
    $C_+ = 1, C_- = 0, K_- = 0,$
$else\ if\ (C > 1)$
    $K_-\ = the\ maximu\ value\ f\ k\ from\ table\ 2$ such that $K < K_+$
    $\Delta_k = K_+ - K_-$
    $C_- = floor(\frac{C * K_+ - B_{new}}{\Delta_k})$
    $C_+ = C - C_-$
$endif$
$F = K_+ * C_+ + K_- * C_- - B_{new}$
```

where K_+ is the first segmentation size, K_- is the second segmentation size, C_+ is the number of code blocks with length K_+ , C_- is the number of code blocks with length K_- and F is the number of filler bits.

As described in list 2.2, in case of data length larger than 6144bit, segmentation is performed and CRC is appended at the end of each code block. The CRC type used is CRC24B.

The CRC24B polynomial generation equation is as follows:

$$g_{CRC24A} = 1 + 1 + D^5 + D^6 + D^{23} + D^{24} \qquad \mathbf{(2.7)}$$

Let assume the parity bits append to the input bits are as follows-:

$$Parity\_seg_{bits} = p_0 , p_1 , p_2 , p_3 \ \dots\dots\dots p_{L-1} \qquad \mathbf{(2.8)}$$

The following pseudo-code shows how insertion processor and CRC appending process:

List 2.3: Filler bits in each code blocks insertion pseudo-code [34]

```
$for\ n = 1$ to $F - 1$
$c_{0K} = zero$
$end\ for$
$n = F$
$s = 0$
$for\ r = 0$ to $C - 1$
    $if\ (r < C_-)$        $K_r = K_-$
    $else$                 $K_r = K_+$
    $end\ if$
    $while\ (k < K_r - L)$
        $c_{rk} = b_s$
        $k = k + 1$
        $s = s + 1$
    $end\ while$
    $if\ C < 1$
        $The\ sequance\ c_{r0}, c_{r1}, c_{r2}, \dots\ c_{r(k-L-1)}$ $is\ used\ to\ calculate\ the$
        $CRC\ parity\ bits, p_{r0}, p_{r1}, p_{r2}, \dots\ p_{r(k-L-1)}$ $as\ described\ in\ sec.$
        $XXXXX, using\ the\ generator\ polynomial\ g_{CRC24A}$ $shown\ in\ eq.$ 8
        $Assume\ that\ filler\ bits\ are\ present\ and\ have\ initial\ value$ 0

        $while\ k < K_r - L$
            $c_{rk} = p_{r(k+L-K_r)}$
            $k = k + 1$
        $end\ while$
    $end\ if$
    $k = 0$
$end\ for$
```

On the receiver side, the de-segmentation block performs the inverse operation of segmentation block. CRC is checked for any error in the received code block bits, if any errors are found in a particular block, the transport block is retransmitted else if no error, removes the CRC from the code block and concatenates the multiple code blocks to form the transport block frame as shown in figure 2.8.

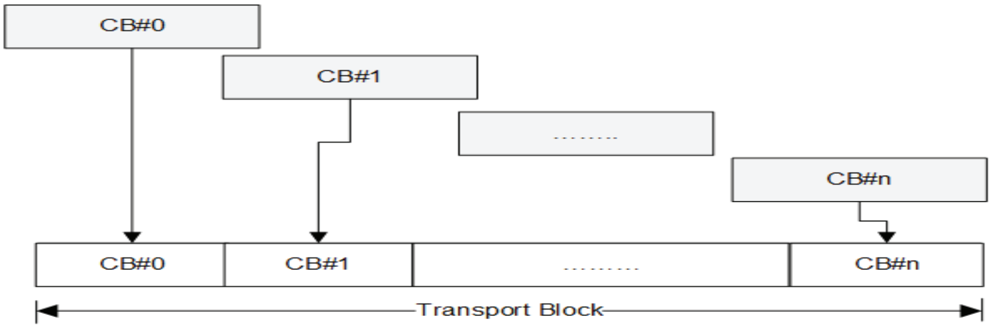


Figure 2.8: Code block De-segmentation

2.2.2.3. Channel coding and channel de-coding

The channel coding used in the PDSCH is turbo coding [35]. Turbo encoder with constant coding rate 1/3 is used for input data coding as described in reference [36]. The scheme of the turbo encoder is parallel of Recursive Systematic Convolution (RSC). The scheme of the turbo encoder is shown in figure 2.9.

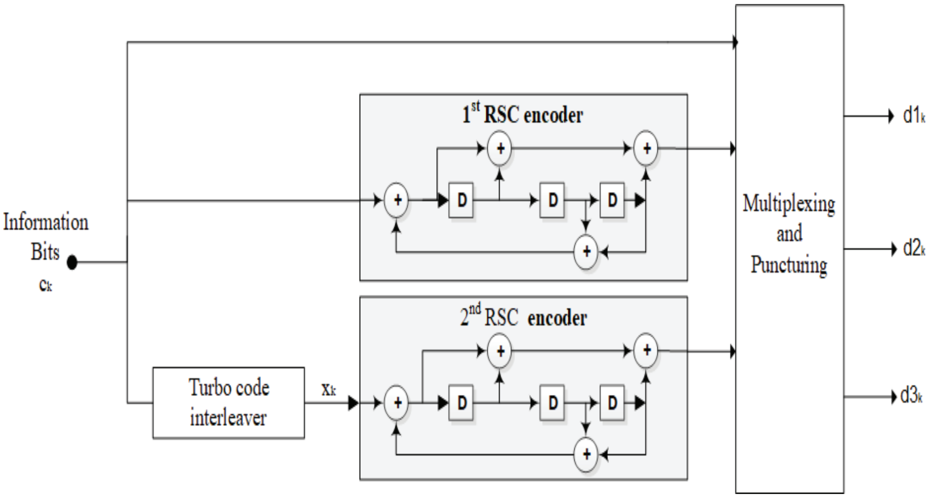


Figure 2.9: Turbo encoder block diagram [35]

The turbo encoder consists of a parallel of Recursive Systematic Convolution (RSC) encoder separated by internal code interleaver. The input goes into the first RSC encoder and after interleaving, it feeds a second RSC encoder. The multiplexing and puncturing block accepts inputs and generates coded bits. The turbo interleaver permutes the indices of the input bits, which improves the turbo code performance. The transfer function of the RSC turbo encoder is defined as follows:

$$G(z) = [1, \frac{G1(z)}{G0(z)}] \tag{2.9}$$

where:

$$G0(z) = 1 + z^{-2} + z^{-3} \tag{2.9.1}$$

$$G1(z) = 1 + z^{-2} + z^{-3} \tag{2.9.2}$$

The initial values of the shift register of the RSC encoder are all zeros when starting to encode the input bits. Let assume the input bits to the turbo encoder block are as follows:

$$Input\_turbo_{bits} = c_0 , c_1 , c_2 , c_3 \ \ldots \ldots \ldots c_k \tag{2.10}$$

The outputs bits from the turbo encoder is as follows: -

$$Output\_turbo_{bits} = d0_0 , d1_0 , d2_0 , d0_1, d1_1, d2_1, \ldots \ldots d0_\mathrm{k}, d1_\mathrm{k}, d2_\mathrm{k} \tag{2.11}$$

where: $d0_\mathrm{k}$ bits are matched to the input bits c_k , $d1_\mathrm{k}$ bits are output from first RSC encoder and $d2_\mathrm{k}$ bits are output from second RSC encoder after interleaving input bits. Let assume the output bits from turbo code interlaver are as follows:

$$output\_turbo\_interalver_{bits} = x_0 , x_1 , x_2 , x_3 \ \ldots \ldots \ldots x_k \tag{2.12}$$

The relation between the output and input bits of turbo code interleaver is as follows:

$$x_i = x_{\pi(i)} \text{ , where } \mathrm{i} = 0,1, \ldots \mathrm{k} - 1 \tag{2.12.1}$$

where the relation between the input $\pi(i)$ index and the output index (i) is as follows:

$$\pi(i) = (f_1 i + f_2 i^2) mod\ k \tag{2.12.2}$$

where the values of f_1 and f_2 depends on the block size k as shown in table 2.3. Table 2.3 shows the relation between output indexes (i) and values of f_1 and f_2 depends on the block size k.

Table 2.3: Turbo code interalver parameter (part1 of 2) [32]

i	K_i	f_1	f_2	I	K_i	f_1	f_2	i	K_i	f_1	f_2	I	K_i	f_1	f_2
1	40	3	10	**48**	416	25	52	**95**	1120	67	140	**142**	3200	111	240
2	48	7	12	**49**	424	51	106	**96**	1152	35	72	**143**	3264	443	204
3	56	19	42	**50**	432	47	72	**97**	1184	19	74	**144**	3328	51	104
4	64	7	16	**51**	440	91	110	**98**	1216	39	76	**145**	3392	51	212
5	72	7	18	**52**	448	29	168	**99**	1248	19	78	**146**	3456	451	192
6	80	11	20	**53**	456	29	114	**100**	1280	199	240	**147**	3520	257	220
7	88	5	22	**54**	464	247	58	**101**	1312	21	82	**148**	3584	57	336
8	96	11	24	**55**	472	29	118	**102**	1344	211	252	**149**	3648	313	228
9	104	7	26	**56**	480	89	180	**103**	1376	21	86	**150**	3712	271	232
10	112	41	84	**57**	488	91	122	**104**	1408	43	88	**151**	3776	179	236
11	120	103	90	**58**	496	157	62	**105**	1440	149	60	**152**	3840	331	120
12	128	15	32	**59**	504	55	84	**106**	1472	45	92	**153**	3904	363	244
13	136	9	34	**60**	512	31	64	**107**	1504	49	846	**154**	3968	375	248
14	144	17	108	**61**	528	17	66	**108**	1536	71	48	**155**	4032	127	168
15	152	9	38	**62**	544	35	68	**109**	1568	13	28	**156**	4096	31	64
16	160	21	120	**63**	560	227	420	**110**	1600	17	80	**157**	4160	33	130
17	168	101	84	**64**	576	65	96	**111**	1632	25	102	**158**	4224	43	264
18	176	21	44	**65**	592	19	74	**112**	1664	183	104	**159**	4288	33	134
19	184	57	46	**66**	608	37	76	**113**	1696	55	954	**160**	4352	477	408
20	192	23	48	**67**	624	41	234	**114**	1728	127	96	**161**	4416	35	138
21	200	13	50	**68**	640	39	80	**115**	1760	27	110	**162**	4480	233	280
22	208	27	52	**69**	656	185	82	**116**	1792	29	112	**163**	4544	357	142
23	216	11	36	**70**	672	43	252	**117**	1824	29	114	**164**	4608	337	480
24	224	27	56	**71**	688	21	86	**118**	1856	57	116	**165**	4672	37	146
25	232	85	58	**72**	704	155	44	**119**	1888	45	354	**166**	4736	71	444
26	240	29	60	**73**	720	79	120	**120**	1920	31	120	**167**	4800	71	120
27	248	33	62	**74**	736	139	92	**121**	1952	59	610	**168**	4864	37	152

Table 2.3: Turbo code interalver parameter (part2 of 2) [32]

i	K_i	f_1	f_2	I	K_i	f_1	f_2	i	K_i	f_1	f_2	I	K_i	f_1	f_2
28	256	15	32	**75**	752	23	94	**122**	1984	185	124	**169**	4928	39	462
29	264	17	198	**76**	768	217	48	**123**	2016	113	420	**170**	4992	127	234
30	272	33	68	**77**	784	25	98	**124**	2048	31	64	**171**	5056	39	158
31	280	103	210	**78**	800	17	80	**125**	2112	17	66	**172**	5120	39	80
32	288	19	36	**79**	816	127	102	**126**	2176	171	136	**173**	5184	31	96
33	296	19	74	**80**	832	25	52	**127**	2240	209	420	**174**	5248	113	902
34	304	37	76	**81**	848	239	106	**128**	2304	253	216	**175**	5312	41	166
35	312	19	78	**82**	864	17	48	**129**	2368	367	444	**176**	5376	251	336
36	320	21	120	**83**	880	137	110	**130**	2432	265	456	**177**	5440	43	170
37	328	21	82	**84**	896	215	112	**131**	2496	181	468	**178**	5504	21	86
38	336	115	84	**85**	912	29	114	**132**	2560	39	80	**179**	5568	43	174
39	344	193	86	**86**	928	15	58	**133**	2624	27	164	**180**	5632	45	176
40	352	21	44	**87**	944	147	118	**134**	2688	127	504	**181**	5696	45	178
41	360	133	90	**88**	960	29	60	**135**	2752	143	172	**182**	5760	161	120
42	368	81	46	**89**	976	59	122	**136**	2816	43	88	**183**	5824	89	182
43	376	45	94	**90**	992	65	124	**137**	2880	29	300	**184**	5888	323	184
44	384	23	48	**91**	1008	55	84	**138**	2944	45	92	**185**	5952	47	186
45	392	243	98	**92**	1024	31	64	**139**	3008	157	188	**186**	6016	23	94
46	400	151	40	**93**	1056	17	66	**140**	3072	47	96	**187**	6080	47	190
47	408	155	102	**94**	1088	171	204	**141**	3136	13	28	**188**	6144	263	480

On the receiver side, Turbo decoder is used to reverse the operation of channel coding. Turbo decoder block accepts input from the de-interleaver block, then performs turbo decoding using a sub-log-MAP (Max-Log-MAP) algorithm to decoded input bits to output bits.

Figure 2.10 shows the block diagram of the turbo decoder [37]. The turbo decoder consists of the following blocks:

- Maximum A Posteriori (MAP) decoder block.
- De-multiplex block.
- Turbo interleaver block.
- Turbo de-interleaver block.

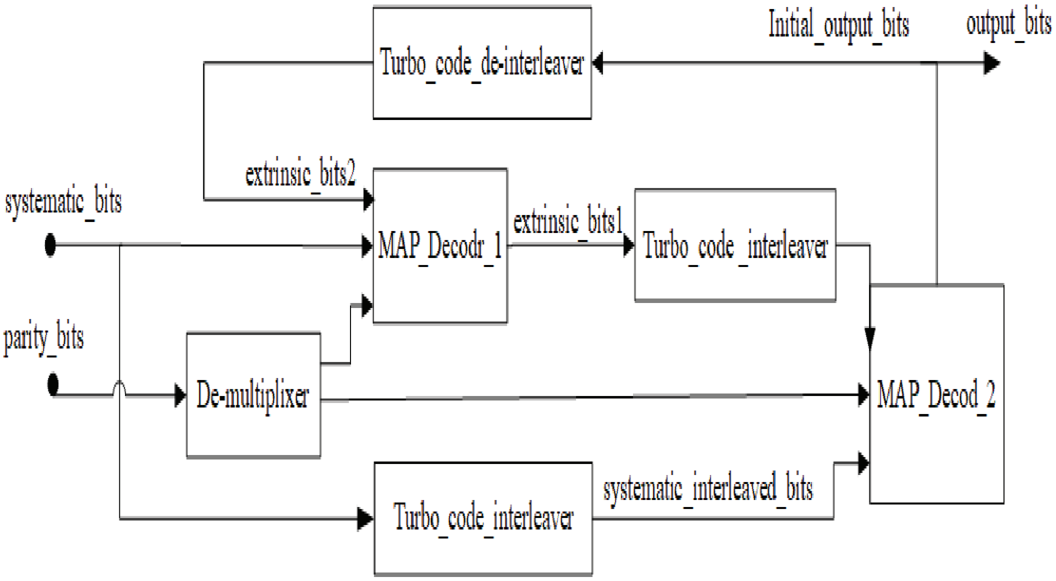


Figure 2.10: Turbo decoder block diagram [37]

The MAP decoder is a decoder designed using Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm. The BCJR algorithm is an algorithm for error correcting codes defined on trellises.The BCJR algorithm calculates forward probabilities, backward probabilities and smoothed probabilities based on channel information. The operation of turbo decoder is performed as the following steps:

1. MAP_decoder_1 accepts the systematic_bits and the parity_bits then generates the extrinsic_bits1 (extrinsic bit is soft estimate bits do not contain any information).
2. The extrinsic_bits1 bits are interleaved and extrinsic_intrelavd_bits1 are generate, also the systematic bits are interleaved and the systematic_intrelavd_bits are generated.
3. The MAP_decoder_2 accepts extrinsic_intrelavd_bits1, the parity_bits, the systematic_intrelavd_bits and generates the initial_output_bit.
4. The initial_output_bit is passed to the de-interleaver and generates the extrinsic_intrelavd_bits2.
5. The MAP_decoder_1 accepts the systematic_bits, the parity_bits and the extrinsic_intrelavd_bits2 then generates the extrinsic_bits1. The steps from 1 to 5 are repeated iteratively until the bit error rate is reached to zero. At the end of the process, the output_bits hard bits are generated according to threshold operation.

2.2.2.4. Interleaver and de-interleaver

Interleaving is the process of reordering data so that successive bunch of data is distributed over a larger sequence of data to reduce the effect of burst errors. Using Interleaver increases the performance of the error protection decoder to correct the burst error [32]. Error protection coding process cannot correct the errors that occur in groups, so using of interleaver allows reducing such error. At the receiver side, the de-interleaver block reverses the operation of interleaver.

2.2.2.5. Rate matching and rate de-matching

The LTE turbo encoder has a fixed coding rate of 1/3. The communication standard added a feature for adapting the throughput based on the channel conditions [36]. In degraded channels, smaller coding rates are used to increase the number of error correction bits and vise verse. Rate matching is used to arrive to at any desired rate by repeating or puncturing. In case of reducing the encoding rate lower than 1/3 repeat the turbo coder output bit. In case of increasing rate higher than 1/3 puncture (remover) some of the turbo coder output bits. Figure 2.11 shows the Rate matching block diagram [38]. The Rate matching block consists of the following sub-blocks: Sub-block interleaver, Bit_collection and Bit_selection.

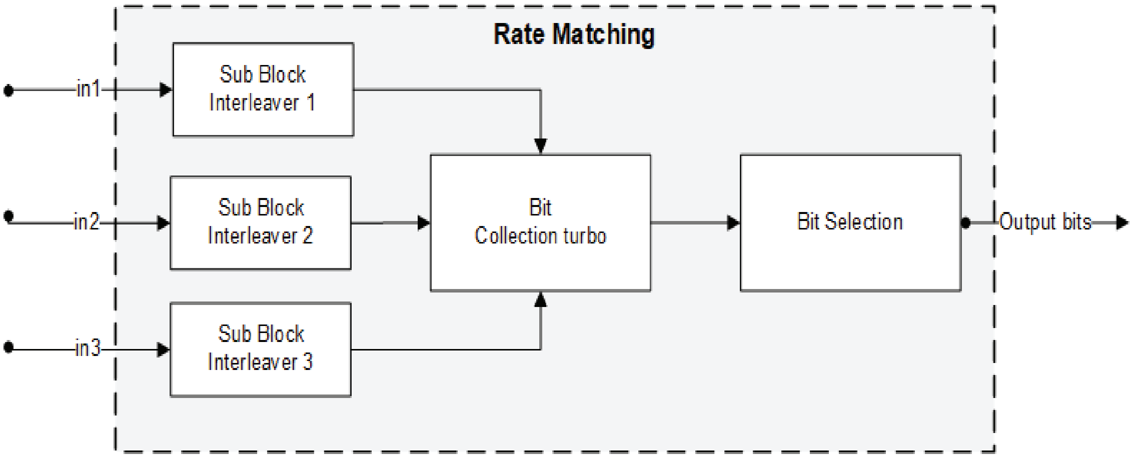


Figure 2.11: Rate matching block diagram

The rate matching accepts three input streams from turbo encoder $d1_k$, $d2_k$ *and* $d3_k$. The bits of each of the three streams are written row-by-row into a matrix with 32 columns. After a column permutation, bits are read out from the matrix column-by-column.

Then information bit streams are passed to sub-block interleaver followed by bit_collection block and bit_selection block. The sub-block_interleaver is based on the classic row-column interleaver with 32 columns and a length-32 intra-column permutation. If input is not multiple of 32 bits, complete the matrix by adding Dummy bits so the block able to interleave the given data.

- The bit stream $d1_k$ is interleaved and output sequence generated as $v1_1$, $v1_2$, $v1_3$, …, $v_{k\pi-1}$.
- The bit stream $d2_k$ is interleaved and output sequence generated as $v2_1$, $v2_2$, $v2_3$, …, $v_{k\pi-1}$.
- The bit stream $d3_k$ is interleaved and output sequence generated as $v3_1$, $v3_2$, v_3 , …, $v_{k\pi-1}$.

The output bits sequence from the sub-block_interleaver is generated as follows:

Let assume the number of columns of the matrix is $C_{subblock} = 32$.

Let assume the number of rows of the matrix is $R_{subblock}$, by finding minimum integer $R_{subblock}$ such that $D < R_{subblock} * C_{subblock}$.

The relation between of D is determine by the column permutation of the sub-block_interleaver.

The output of the first sub-block interleaver is denoted by $v0_1$, $v0_2$, $v0_3$, …, $v0_{k\pi}$ where $v_i = y * \pi_i$ where

$$\pi_i = mod\left(\left(P\left(\frac{i}{R_{subblock}}\right) + C_{subblock} * mod(i, R_{subblock}) + 1\right), k\pi\right) \quad \mathbf{(2.13)}$$

Where $k\pi$ is the output length of the sub-block_interleaver and given as follows: $k\pi = R_{subblock} * C_{subblock}$
The column permutation function (P) of the sub-block_interleaver is given in table 2.4:

Table 2.4: Inter-column permutation matric for sub-block interleaver [32]

Number of columns	**Inter-column permutation pattern**
32	[0,16,8,24,4,20,12,28,2,18,10,26,6,22,14,30,1,17,9,25,5,21,13,29,3,19,11,27,7,23,15,3]

Next, the bit_collection block accepts inputs from three sub-block_interleaver and generates output depending on coding type. The bit_collection block is worked based in circular buffer.

The circular buffer length K_w for the r code block is generated for $K_w = 3 * k\pi$ as follows:

- $w_k = v0_k \; for \; k = 0, \ldots, k\pi - 1$ $\quad$ $(\mathbf{2.14.1})$
- $w_{k\pi+2k} = v1_k \; for \; k = 0, \ldots, k\pi - 1$ $\quad$ $(\mathbf{2.14.2})$
- $w_{k\pi+2k+1} = v2_k \; for \; k = 0, \ldots, k\pi - 1$ $\quad$ $(\mathbf{2.14.3})$

The soft buffer size for the r-th code block is denoted by N_{cb}. The size N_{cb} for the downlink is obtained as follows:

$$N_{cb} = \min\left(\left|\frac{N_{IR}}{C}\right|, K_w\right) \qquad (\mathbf{2.15.1})$$

where N_{IR} is equal to $N_{IR} = \left(\frac{N_{soft}}{K_{MIMO} * \min\left(M_{DL_{HARD}}, M_{limit}\right)}\right)$ $\quad$ $(\mathbf{2.15.2})$

where N_{soft} is the total number of soft channel bits, K_{MIMO} is constant dependent in transmission mode and equal to 2 if the UE is configured to receive PDSCH transmission, M_{limit} is constant equal to 8 and M_{DL_HARD} is the maximum number of downlink processor. Finally, the bit_selection receives inputs from bit_collection, skip dummy bits, and output the required output bits with the proper size.

On the receiver side, the rate de-matching is used to reverse the operation of rate matching block [39]. Figure 2.13 shows the block diagram of the rate de-matching. The bit_de-selection block accepts input bits and divides it into three outputs; each output has a length equal to the code block length. Three bit_de-interleaver blocks accept outputs from the bit_de- selection block. Each one of the bit_de-interleaver blocks accepts input from bits de-deslection block with a length equal to the code block length. The bit_de-interleaver block inverse the operation of sub-block_interleaver.

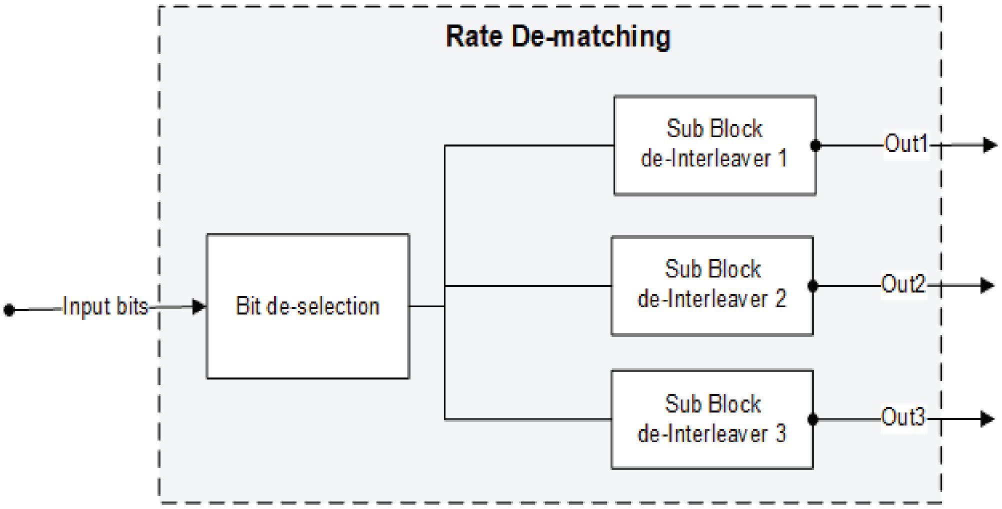

Rate De-matching
Input bits
Bit de-selection
Sub Block de-Interleaver 1
Sub Block de-Interleaver 2
Sub Block de-Interleaver 3
Out1
Out2
Out3

Figure 2.12: Rate de-matching block diagram

2.2.2.6. Scrambler and de-scrambler

The scrambler is a block that pseudo-randomly changes the values of bits into a data block, thus ensure that the interference is randomized for each different cell or to introduce security as part of an encryption procedure. The data bits are scrambled with a sequence that is unique to each cell by initializing the sequence generators in the cell based on the physical cell identity.

Let assume the input bits to the turbo encoder block are as follows:

$$Input\_scrambler_{bits} = e_0\ , e_1\ , e_2\ , e_3\ \dots\dots\dots e_{M-1} \qquad \mathbf{(2.16)}$$

where M is the number of bits transmitted on the physical channel in one sub-frame. The output of the scrambler is determined according to the following equation:

$$Output\_scrambler_{bits} = (e_{\mathrm{i}} + \ c_{\mathrm{i}})\ \mathrm{mod2}\ , \mathrm{i} = 0,1,2, \dots \mathrm{M} \qquad \mathbf{(2.17)}$$

where c_{i} are the pseudo-random sequences.

The pseudo-random sequences are defined by a length-31 Gold sequence [40]. The output sequence $c(n)$ of length M bits, where n= 0, 1... M-1 is defined by the following equations:

- $c(n) = \big(x1(m + NC), x1(n + NC)\big)\ mod2 \qquad \mathbf{(2.18.1)}$
- $x1(n + 31) = \big(x1(n + 3), x1(n)\big)\ mod2 \qquad \mathbf{(2.18.2)}$
- $x2(n + 31) = \big(x2(n + 3), x2(n + 2), x2(n + 1), x2(n)\big)\ mod2 \qquad \mathbf{(2.18.3)}$

$where\ Nc = 1600$

The scrambler block operation is divided into two steps as follows:

1. Calculating initial values for X2, X1 sequence generators for NC iterations.
2. Apply X1, X2 sequence generation output to the input after NC iterations.

where the initialization of sequence is performed as follows:

1. The first m-sequence shall be initialized with
   $\mathrm{x1}(0) = 1, \mathrm{x1}(\mathrm{n}) = 0, \mathrm{n} = 1,2, \dots 30 \qquad \mathbf{(2.19.1)}$
2. The initialization of the second m-sequence is denoted by

$$C_{initial} = \sum_{i=0}^{i=30} x2(i) = 2^{i} \qquad \mathbf{(2.19.2)}$$

where value of $C_{initial}$ is depending on the application of the sequence. For the shared channel, the scrambler sequence generator should be initialized at the start of each sub-frame with value of $C_{initial}$ for PDSCH channel

$$C_{initial} = N_{RNTI} * 2^{14} + q * 2^{13} + \frac{n_s}{2} * 2^9 8 + N_{ID} \quad (\mathbf{2.19.3})$$

where N_{RNTI} is the Radio Network Temporary Identifier, q is the code word index, n_s is the slot index within frame, and N_{ID} is the physical layer cell identify.

On the de-scrambler function, same Gold sequence generator illustrated is used to invert the scrambling operation. Scrambler Sequence Generation in the Receiver of LTE PDSCH is the same as that of the Transmitter. The de-scrambler block operates on the LLR outputs of the demodulator, converting the Gold-sequence bits into either 1 or -1. The de-scrambler block operation is divided into three steps: -

1. Calculating initial values for X2, X1 sequence generators for NC iterations.
2. Calculating value of $c(n)$ where $c(n) = x1(n + NC) XOR\ x2\ (n + NC)$ for NC iterations.
3. Generates de-scrambler results, if $(c(n) == 0)$ the de-scrambler output is LLR (n) else, the de-scrambler output -LLR (n).

where NC=1600 and LLR (n) is a Log-Likelihood Ratio calculated values.

2.2.2.7. Modulator and de-modulator

The Modulator accepts groups of input bits and maps them to specific constellation symbols, according to the modulation method that you specify. The LTE standard supports QPSK 16QAM and 64QAM modulation schemes types for the LTE PDSCH [41]. Figure 2.14 shows constellation diagrams of these three modulation schemes.

Multiple modulation schemes allow adaptive modulation based on channel conditions. When the Signal-to-Noise Ratio (SNR) is high, denser constellations (ex: 64QAM) are used to increase the throughput. However, when the Signal-to-Noise Ratio (SNR) is low, modulation schemes with more inter-symbol separation should be used to reduce the throughput and decrease the bit error rate.

The modulation process generates complex symbols depending on the input bits. Let assume the input sequence bits to the modulation block are as follows:

$$Input\_modulation_{bits} = f_0\,, f_1\,, f_2\,, f_3\ \dots\dots\dots f_{M-1} \quad (\mathbf{2.20})$$

where M is the total number of sequence bits.

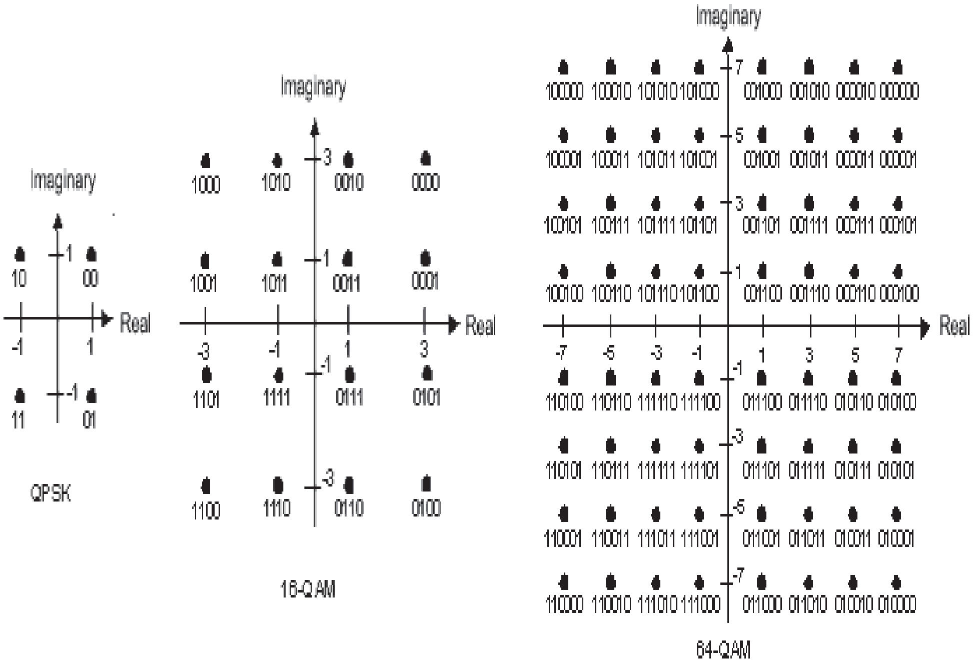


Figure 2.13: Constellation diagrams of modulation schemes [41]

For QPSK modulation scheme, every two bits of input bit sequence $(f_i and\ f_{i+1})$ are modulated into a complex symbol $g_i = I + jQ$ where $I\ and\ Q$ are generated as shown in table 2.5.

Table 2.5: QBSK modulation lookup table [32]

f_i, f_{i+1}	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	$\frac{-1}{\sqrt{2}}$	$\frac{-1}{\sqrt{2}}$

For 16QAM modulation scheme, every four bits of input bit sequence $(f_i, f_{i+1}, f_{i+2}\ and\ f_{i+3})$ are modulated into a complex symbol $g_i = I + jQ$ where $I\ and\ Q$ are generated as shown in table 2.6.

Table 2.6: 16QAM modulation lookup table [32]

$f_i, f_{i+1}, f_{i+2}, f_{i+3}$	I	Q
0000	$\frac{1}{\sqrt{10}}$	$\frac{1}{\sqrt{10}}$
0001	$\frac{1}{\sqrt{10}}$	$\frac{3}{\sqrt{10}}$
0010	$\frac{3}{\sqrt{10}}$	$\frac{1}{\sqrt{10}}$
0011	$\frac{3}{\sqrt{10}}$	$\frac{3}{\sqrt{10}}$
0100	$\frac{1}{\sqrt{10}}$	$\frac{-1}{\sqrt{10}}$
0101	$\frac{1}{\sqrt{10}}$	$\frac{-3}{\sqrt{10}}$
0110	$\frac{3}{\sqrt{10}}$	$\frac{-1}{\sqrt{10}}$
0111	$\frac{3}{\sqrt{10}}$	$\frac{-3}{\sqrt{10}}$
1000	$\frac{-1}{\sqrt{10}}$	$\frac{1}{\sqrt{10}}$
1001	$\frac{-1}{\sqrt{10}}$	$\frac{3}{\sqrt{10}}$
1010	$\frac{-3}{\sqrt{10}}$	$\frac{1}{\sqrt{10}}$
1011	$\frac{-3}{\sqrt{10}}$	$\frac{3}{\sqrt{10}}$
1100	$\frac{-1}{\sqrt{10}}$	$\frac{-1}{\sqrt{10}}$
1101	$\frac{-1}{\sqrt{10}}$	$\frac{-3}{\sqrt{10}}$
1110	$\frac{-3}{\sqrt{10}}$	$\frac{-1}{\sqrt{10}}$
1111	$\frac{-3}{\sqrt{10}}$	$\frac{-3}{\sqrt{10}}$

For 64QAM modulation scheme, every six bits of input bit sequence $(f_{\mathrm{i}}, f_{\mathrm{i+1}}, f_{\mathrm{i+2}}, f_{\mathrm{i+3}}, f_{\mathrm{i+4}}, f_{\mathrm{i+5}}\ and\ f_{\mathrm{i+6}})$ are modulated into a complex symbol $g_{\mathrm{i}} = I + jQ$ where $I\ and\ Q$ are generated as shown in Appendix A.

On the receiver side, the de-modulator block de-maps QPSK symbol, 16-QAM symbol and 64-QAM symbol to soft bits according to the modulation method specified. The soft de-mapper outputs are the Log-Likelihood Ratio (LLR) for a certain constellation-mode [42]. In soft bit demodulator, is a modulation process that estimates from the received bit not only hard bits (hard bits are typically binary), but also their confidence levels.

On the transmitter side, only hard bits are involved, and they take binary digit 0 or 1 but on the receiver side, the binary digit the confidence levels are the magnitudes of the binary digit. In this case, the negative and positive polarities are expressed as numeric values of -1 and +1[43].

The soft decision process returns an integer sequence is used for PDSCH demodulation, as the turbo decoder requires a soft encoded input. For example, integer values from 0 to 7 are used for QPSK demodulation. Value 0 indicates a strongest possibility of value 0 in original bit sequence while value 3 indicates the weakest possibility of value 0 in the original bit sequence. Similarly, value 4 indicates a week 1 while 7 indicates a strong 1. Let assume the soft input bits to the de-modulation block are as follows:

$$Input\_de\_modulation_{softbits} = d_0\ , d_1\ , d_2\ , d_3\ \dots\dots\dots d_{M-1} \qquad (\mathbf{2.21})$$

where M is the total number of sequence bits. The soft decision values are calculated according to Table 2.7.

Table 2.7: Soft decision QPSK demodulation lookup table (part1 of 2) [43]

$\boldsymbol{Re(d_{\mathrm{i}})}$	$\boldsymbol{b_{\mathrm{i}}}$
$Re(d_{\mathrm{i}}) \geq 0.9$	0
$0.9 > Re(d_{\mathrm{i}}) \geq 0.9$	1
$0.6 > Re(d_{\mathrm{i}}) \geq 0.3$	2
$0.3 > Re(d_{\mathrm{i}}) \geq 0$	3
$0 > Re(d_{\mathrm{i}}) \geq -0.3$	4
$-0.3 > Re(d_{\mathrm{i}}) \geq -0.6$	5
$-0.6 > Re(d_{\mathrm{i}}) \geq -0.9$	6
$-0.9 > Re(d_{\mathrm{i}})$	7

Table 2.7: Soft decision QPSK demodulation lookup table (part2 of 2) [43]

$Im(d_i)$	b_{i+1}
$Im(d_i) \geq 0.9$	0
$0.9 > Im(d_i) \geq 0.9$	1
$0.6 > Im(d_i) \geq 0.3$	2
$0.3 > Im(d_i) \geq 0$	3
$0 > Im(d_i) \geq -0.3$	4
$-0.3 > Im(d_i) \geq -0.6$	5
$-0.6 > Im(d_i) \geq -0.9$	6
$-0.9 > Im(d_i)$	7

2.2.2.8. Resource element mapper and resource element de-mapper

The resource element mapper is a time-frequency representation of data organized as the resource grid as shown in figure 2.15. The resource grid is a two-dimensional map of symbol in the horizontal axis (time domain) and sub-carrier on the vertical axis (frequency domain) [31].

The placement of data within the resource grid is important and Depending on which sub-frame is in use. The type of data placed in a resource grid includes the following [44]:

- The Physical Downlink Shared Channel PDSCH signal, which carries user data. This signal placed in all sub-frames
- The Cell Specific Reference (CRS) signal, which used by the receiver for estimation channel frequency response and cross-channel effects. This signal placed in all sub-frames.
- The Physical Downlink Control Channel (PDCCH) signal, which helps the carries important information for processing (ex: modulation scheme ...etc.). These signals placed at the beginning of each sub-frame.
- The Primary synchronization signals (PSS) data and Secondary Synchronization Signals (SSS) data, which help determine the frame timing and cell identification. These signals placed in sub-frames 0 and 5.
- The broadcast channel (BCH) signal, which carries the Master Information such as cell bandwidth. These signals placed in sub-frame 0 and repeated every 10 sub-frames.

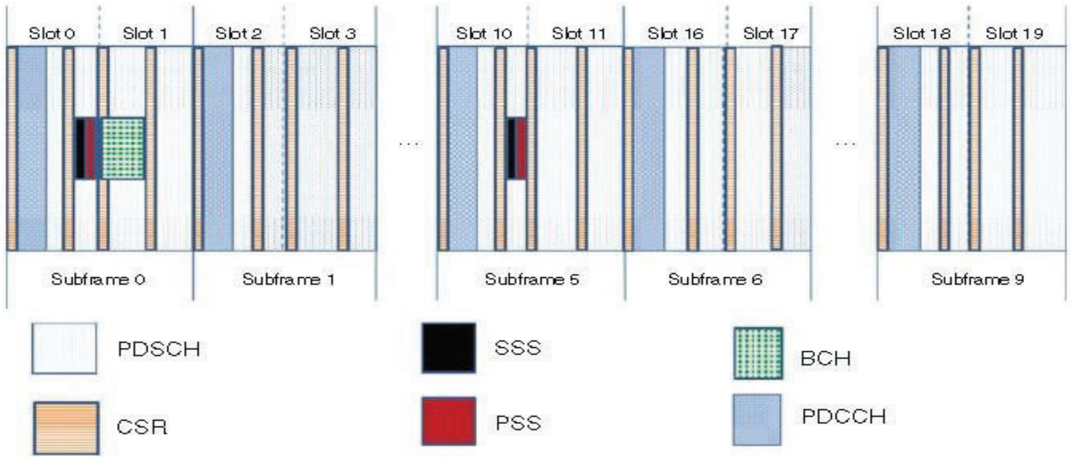


Figure 2.14: LTE resource grid [31]

On the receiver side, the resource element de-mapper inverts the operations of resource grid mapping at the receiver side. The rule of a resource element de-mapper is extracting PDSCH, CRS …etc. from the resource grid and feed each type of data to the corresponding next stage. The PDSCH receiver chain should accept PDSCH input from the resource grid to complete the processing of data in the receiver chain.

Chapter 3 : SDSoC

The SDSoC is a novel development tool used to create hardware-software co-design on a heterogeneous FPGA-CPU platform. The SDSoC tool makes simplest and shortest development cycles to implement heterogeneous FPGA-CPU as well as generating required hardware interface logic to handle the data flow between hardware and software automatically [45]. The SDSoC tool is integrated with a High-level synthesis (HLS) which is used to implement hardware in an FPGA synthesized from a C/C++ language description [46]. Transforming C/C++ code to an RTL implementation process using HLS is provided in [47]. The Table 3.1 shows the constructs of mapping C/C++ to RTL.

Table 3.1: Mapping of C-code to RTL construct

C-function	RTL
Function	Modules
Arguments	Input/output ports
Operators	Functional Units
Scalars	Wires or register
Arrays	Memories
Control flows	Control logics

In addition, the SDSoC tool includes system-level profiling and performance analysis capability. The profiling and performance analysis tool includes hardware utilization calculation, latency calculation and hardware acceleration improving estimation. A detailed description of the features of the tool is provided in [48].

This chapter includes the following topics: SDSOC pragma specification, the typical HW/SW co-design design flow using SDSoC design flow and the proposed automated performance-based design technique developed to generate platform using SDSOC tool to meet performance metrics constraints.

3.1. SDSoC pragma specification

This section describes pragmas for the SDSoC compilers used for system optimization. The SDSoC pragmas are used in HLS to enhance the hardware function performance. All pragmas specific to the SDSoC environment are prefixed with #pragma [49]. The pragma should be inserted prior to a function declaration or at a function call in C/C++ source code.

The pragmas should be placed within the boundaries of the required location in the C/C++ source. The pragmas need to be written in the C-code depending in its type. The pragma syntax has been defined to be consistent with standards like OpenACC. The different Pragmas supported by the SDSOC tool include the following: function optimization, loop optimization, array optimization and Interface management.

3.1.1. Function optimization

The function optimization pragmas include inline function and function pipelining.

1) Inline function

Inlining function pragma used to remove function hierarchy. Removing function hierarchy leads to enhancing latency and throughput by removing of a cycle overhead to enter and exit functions.The following Pragma example used to prevent "func_top" function from being in-lined, the Pragma should be applied to the top level function "func_top".

#pragma AP inline off

2) Function pipelining

Pipelining pragma is a powerful method used to optimize the communication between functions and improve throughput. The following pragma example used to pipeline the function "func" with II as 4.

#pragma AP pipeline II=4 enable_flush

3.1.2. Loop optimizations

The loop optimization pragmas include inline unrolling, merging and flattening nested loop.

1) Unrolling

Unroll for loops pragma used to create multiple independent operations instead of a single collection of operations. The following pragma example used to unroll the loop L1 in function "func" with unrolling factor e 2.

#pragma AP unroll skip_exit_check factor=2

2) Merging

Merging pragma for loops used to combine multiple sequential loops to prevent the creation of additional unnecessary clock cycles. The following pragma example used to merge all consecutive loops into a single loop in the function “func”.

#pragma AP loop_merge

3) Flattening nested loops

Flattening pragma used to grab nested loop to a single loop to improve latency. This because hardware implementation requires one clock cycle to move from an outer loop to an inner loop and from an inner loop to an outer loop. The following pragma example used to flatten for loop L1 in function “func” where loop L1 is the inner loop has the body and the “func” function.

#pragma AP loop_flatten

3.1.3. Array optimizations

The array optimization pragmas include horizontal mapping, vertical mapping and array partitioning.

1) Horizontal mapping

Horizontal mapping pragma used to concatenate two arrays into one array. The following pragma example used to concatenate arrray1 and array2 into array3.

#pragma AP array_map variable=array1 instance=array3 horizontal
#pragma AP array_map variable=array2 instance=array3 horizontal

2) Vertical mapping

Horizontal mapping pragma used to concatenate two array vectors into one array vector. The following pragma example used to concatenate arrray1 vector and array2 vector into array3 vector.

#pragma AP array_map variable=array1 instance=array3 vertical
#pragma AP array_map variable=array2 instance=array3 vertical

3) Array partitioning

Array partitioning pragma used to divide array into smaller arrays to increases throughput. The following pragma example partitions array Z(21) in function “func” into five arrays. Because 5 is not an integer multiple of 21, four of the of the arrays have 4 elements and one have 5 (containing elements Z(16:21)

#pragma AP array_partition variable=Z block factor=5

3.1.4. Interface management

The attitude of the interface has specified either behavior or explicitly depending on the type of input source. This allows different IO protocol to be used so the function interfaces with any hardware resource. The different interface management pragmas include ap_bus, ap_memor, and ap_fifo.

1) ap_bus

An ap_bus interface used to communicate with a bus bridge. The interface does not adhere to any specific bus standard but is generic enough to be used with a bus bridge. The bus bridge must be able to cache all burst writes.

2) ap_memory

The ap_memory port interface is used to communicate with memory elements (RAMs, ROMs) as shown in figure 3.1. The ap_memory used when the implementation requires random accesses to the memory address locations. Array arguments are typically implemented using the ap_memory interface.

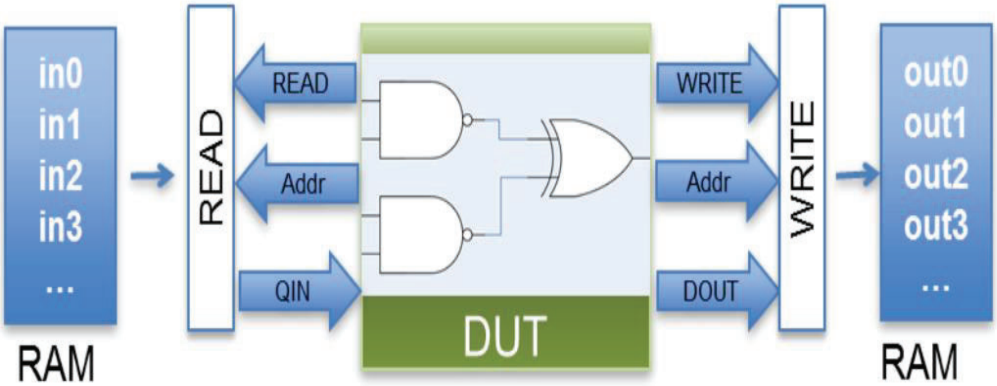


Figure 3.1: Memory interface management

3) ap_fifo

The ap_memory port interface is used to communicate with memory elements (FIFO) as shown in figure 3.2. The ap_fifo interface is used if required access to a memory element and this access is performed in a sequential manner (no random access).

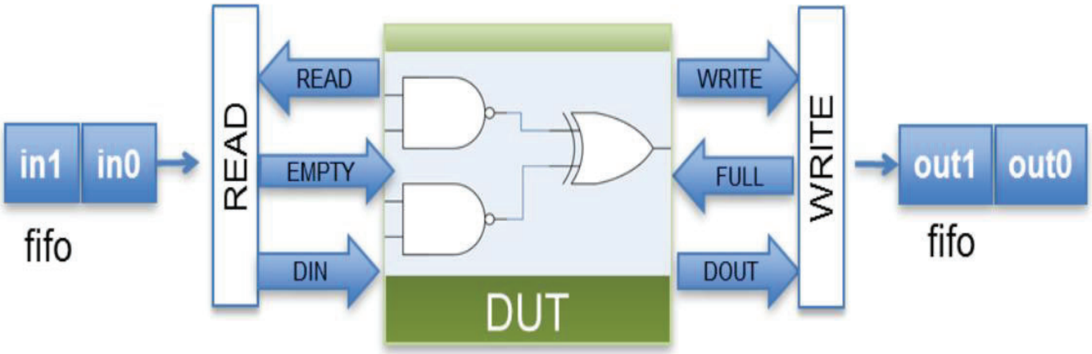


Figure 3.2: FIFO Interface management

3.2. Typical HW/SW co-design design flow using SDSoC design flow

The typical HW/SW co-design flow using SDSoC is shown in figure 3.3. First, the developer should design the application coded in C/C++. Next, the user should define the requirement of each C/C++ functions, so the user should select manually which functions must be implemented as software functions or hardware-accelerated functions synthesized by HLS [48].

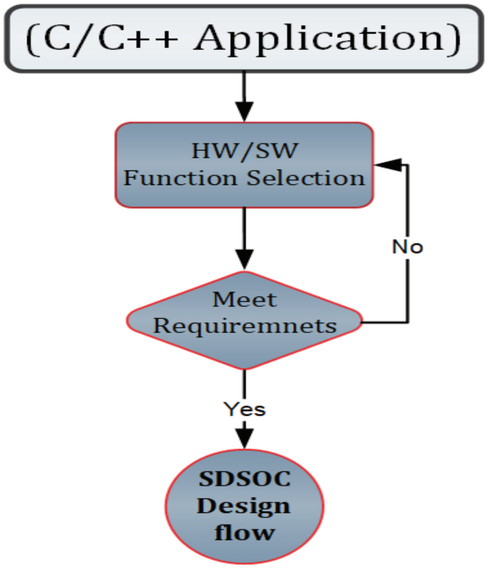


Figure 3.3: Typical HW/SW co-design flow using SDSoC [48]

After refining all C/C++ functions, the SDSoC design flow is executed. The SDSoC design flow is shown in figure 3.4. First, all C/C++ functions are compiled, next, the implementation of the C/C++ function had to be partitioned into software implementation functions or hardware-accelerated functions depending on user selection.

The Software Development Kit (SDK) tools and Vivado Tool (Xilinx Inc.) are the elements of hardware and software system design. The Vivdo tool includes High-Level Synthesis (HLS) is used for creating the hardware system component by transforming the C/C++ code to an RTL implementation [45].

The SDK tool is a software design suite that includes driver support, C/C++ Compiler library supported for ARM and tools for debugging and profiling. Finally, the integration, the necessary communication blocks between hardware, and software and the SoC platform creation is done by SDSoC tool.

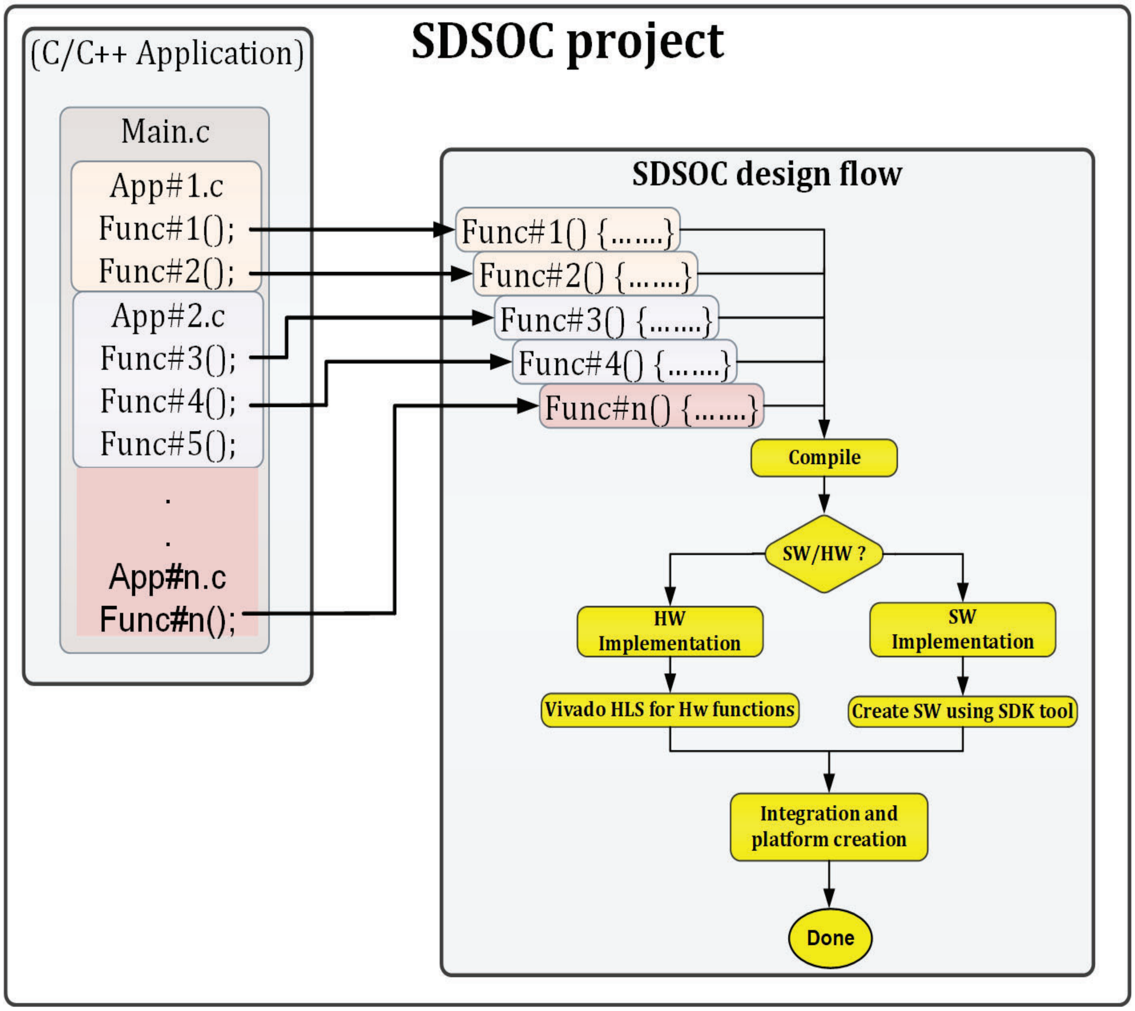


Figure 3.4: SDSoC design flow

The SDSoC tool generates the embedded FPGA SoC platform as shown in figure 3.5. This platform allows executing part of the C/C++ code on the arm processor as a software functions and the other parts of the codes on the FPGA as hardware-accelerated functions. The embedded FPGA platform consists of:

1. Processing system includes dual-core ARM cortex-A9 processor hardcore processor.
2. Interfacing logic includes ACP port interconnect, data movers and reset blocks.
3. Hardware logics includes HLS generated block.

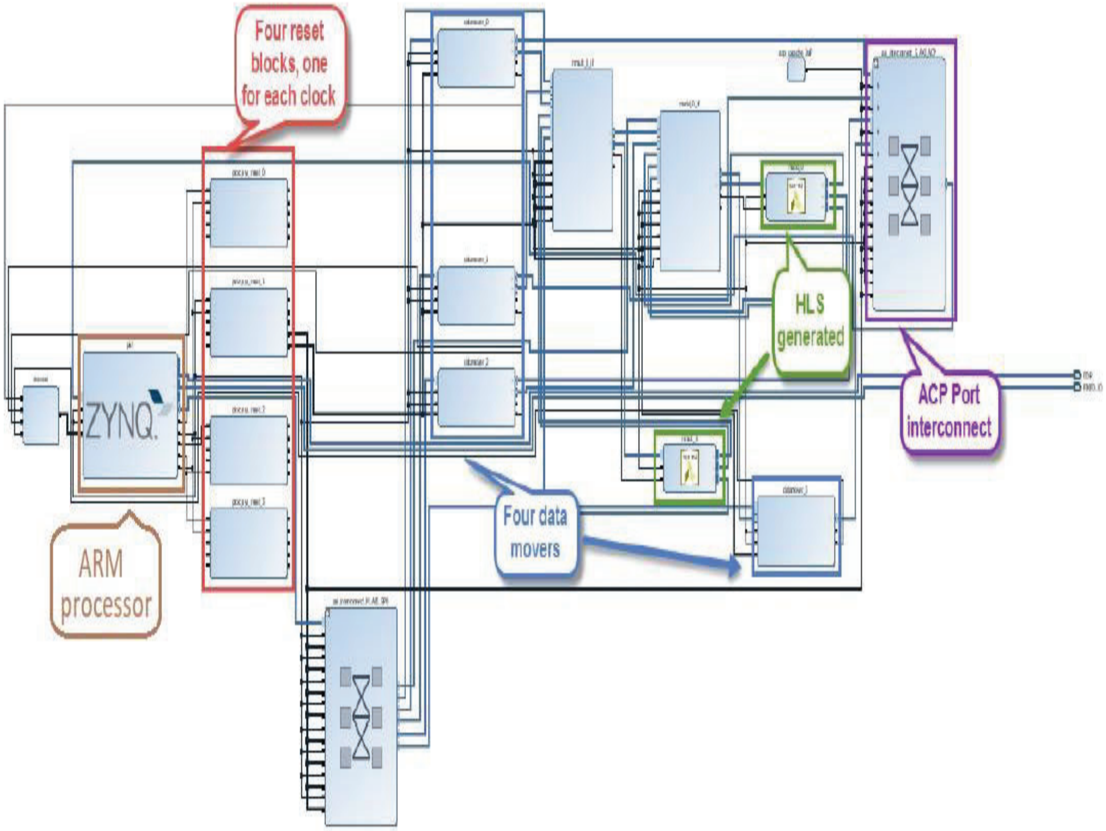


Figure 3.5: Embedded FPGA platform [48]

3.3. Proposed automated performance-based design technique

This section illustrates in details the proposed automated performance-based design technique using SDSoC tool. The objective of this technique is to determine platforms that achieve performance metrics and select the platform that achieves the best performance.

The performance metrics constraints to the technique are hardware utilization, latency and dynamic power. Therefore, a designer defines the upper limit of this performance metrics according to the implementation requirement.

On the other hand, the performance metrics output from the technique are hardware utilization, latency, dynamic power, hardware acceleration and Figure of Merit (FoM) [50]. Therefore, a designer able to explore this performance metrics results for all possible implementations.

Figure 3.6 shows the proposed performance-based techniques flow diagram. First, the developer should design the application code written in C/C++ and define functions that should be implemented either as a software function or as a hardware-accelerated function (Let for example define a number of functions $=(n)$).

The introduced algorithm is fully automated and uses a set of shell scripts for executing the SDSoC tool to generate required platforms. There are three flow control files are defined in the technique used to drive the execution of the shell scripts according to designer requirement.

The flow control files are divided as follows:

- First, the target_function_list file contains the modules functions names which designed to be implemented either as software function or hardware-accelerated function.
- Second, the implementation_configuration file which used to define the type of FPGA device, operating system, the clock frequency, type of design flow...etc.
- Third, the performance_metrics_constrain file which is the design constraints and includes constraints on hard- ware utilization, dynamic power, and latency.

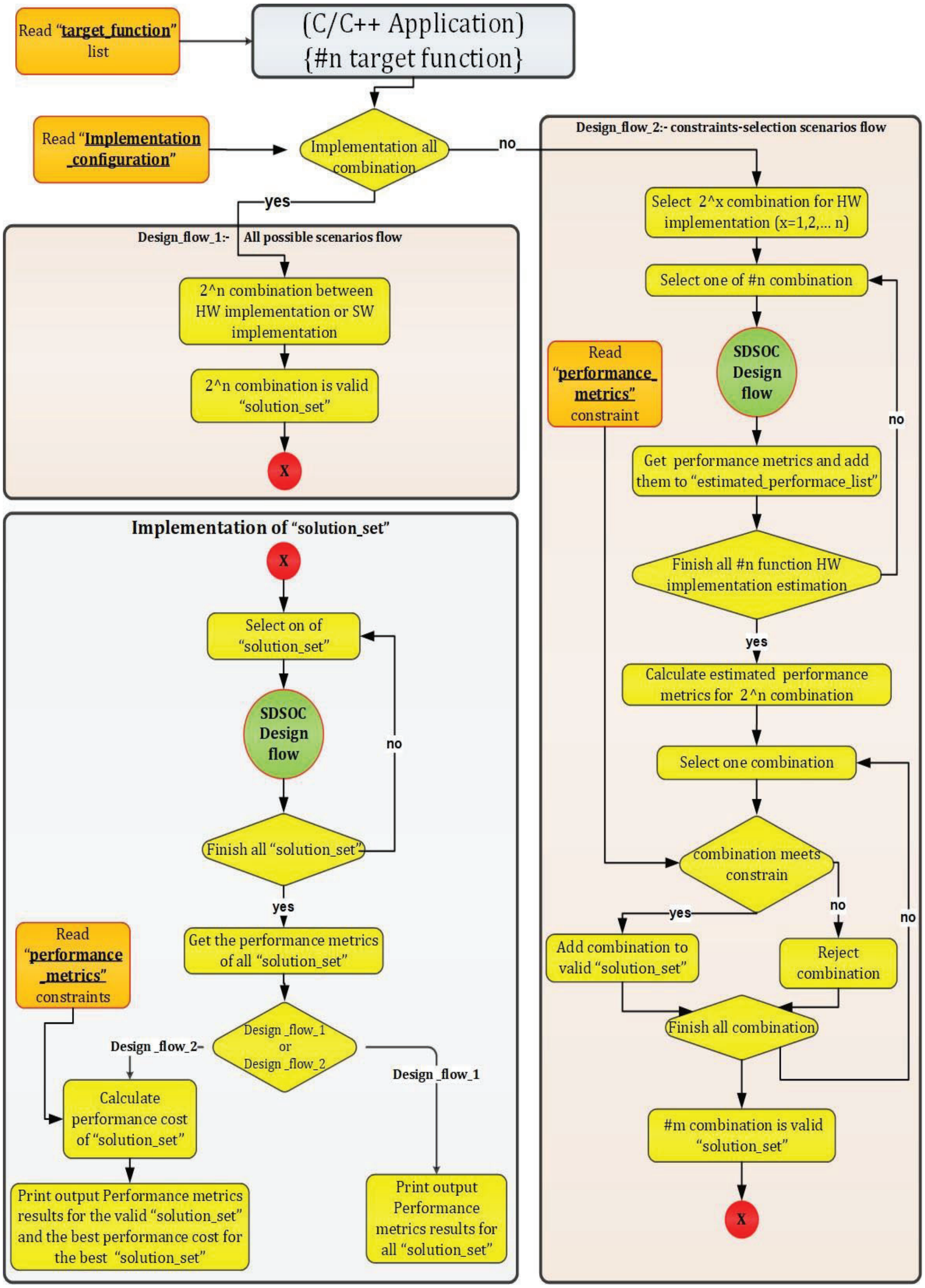

Read "target_function" list
(C/C++ Application)
{#n target function}
Read "Implementation _configuration"
Implementation all combination
no
yes
Design_flow_1:- All possible scenarios flow
2^n combination between HW implementation or SW implementation
2^n combination is valid "solution_set"
X
Implementation of "solution_set"
X
Select on of "solution_set"
SDSOC Design flow
no
Finish all "solution_set"
yes
Get the performance metrics of all "solution_set"
Read "performance _metrics" constraints
Design _flow_1 or Design _flow_2
Design _flow_2
Design _flow_1
Calculate performance cost of "solution_set"
Print output Performance metrics results for the valid "solution_set" and the best performance cost for the best "solution_set"
Print output Performance metrics results for all "solution_set"
Design_flow_2:- constraints-selection scenarios flow
Select 2^x combination for HW implementation (x=1,2,... n)
Select one of #n combination
Read "performance_ metrics" constraint
SDSOC Design flow
no
Get performance metrics and add them to "estimated_performace_list"
Finish all #n function HW implementation estimation
yes
Calculate estimated performance metrics for 2^n combination
Select one combination
combination meets constrain
yes
no
no
Add combination to valid "solution_set"
Reject combination
Finish all combination
#m combination is valid "solution_set"
X

Figure 3.6: Performance-based design technique flow chart

The developer should design the application code written in C/C++ and define the (n) functions in the target_function_list files which are the functions that could be implemented either as a software function or as a hardware-accelerated function (let for example the designer defines 3 functions which are fun1, fun2 and fun3). Next, the shell scripts read all C/C++ files and compile all C/C++ files for any Syntax errors or semantic error. Then the shell scripts read the implementation_configuration file. As shown in Figure 3.6 the technique is divided into two design flows depending on the implementation_configuration.

- The first is design_flow_1 (all possible scenarios flow).
- The second is design_flow_2 (constrained-selection scenarios flow).

The purpose of the design_flow_1 is to implement all possible scenarios for (n) function defined in the target_function_list file, thus (2^n) combinations between software implementation function or hardware-accelerated function are stated to be a valid solution_set. For example, the defined 3 functions in the target_function_list generate $(2^3 = 8)$ combinations between software implementation function or hardware-accelerated function as shown in table 3.2.

Table 3.2: Three function configuration scenario example:
(0) Software function and (1) Hardware-accelerated logic

platform name	fun1	fun2	fun3
platform0	0	0	0
platform1	1	0	0
platform2	0	1	0
platform3	1	1	0
platform4	0	0	1
platform5	1	0	1
platform6	0	1	1
platform7	1	1	1

The number (1) in the table indicates that this function in current platform is a hardware-accelerated logic and the number (0) indicates that this function in current platform is a software function. Therefore, platform0 is configuring that all the functions (fun1, fun2, and fun3) are implemented as software functions.

The platform1 is configuring that the functions (fun1) is implemented as software functions and other functions (fun2 and fun3) are implemented as a hardware accelerated function and so on for all possible combinations between software implementation or hardware-accelerated implementation. design_flow_1 allows exploring the performance metrics of every possible combination between software implementation and hardware-accelerated implementation of (n) function by taking into consideration the (2^n) combinations a valid solution_set.

Section 4.2 represent implementation of LTE PDSCH transmitter and receiver as a case-study for the design_flow_1

The purpose of the design_flow_2 is to implement specific possible scenarios that meet Performance constraints defined in the performance_metrics_constrain file. The Performance constraints described in file are hardware utilization, dynamic power, and latency. The steps of the design_flow_2 are the following:

1) Select the (2^x) combinations for hardware implementation where ($x = 0, 1, \dots n$). For current example, select combinations from table 3.2 which generate platform1, platform2 and platform4.
2) Select one of (2^x) combinations from step-1 to be input to the SDSoC design flow shown in Figure 3.4.
3) After finishing the execution of step-2, get the performance metrics from step-2 and add the estimate_performance_list.
4) Repeat from step1 to step-3 until finish the implementation and performance metrics estimation of all (2^x) combinations defined in step-1. For example, the estimate_performance_list will be as shown in table 3.3 where (h1, h2, h4, p1, p2, p4, l1, l2 and l4) numeric values generated in step3 are.

Table 3.3: Estimated performance list for selected combination

performance metrics	combination 1	combination 2	combination 4
hardware utilization	h1	h2	h4
dynamic power	p1	p2	p4
latency	l1	l2	l4

5) Calculate the estimated performance metrics for all (2^n) combinations using the information from estimate_performance_list. The estimated performance metrics for all (2^n) combinations are summation of the performance metrics of the (2^n) combinations. For current example, the estimated performance metrics for all (2^n) combinations are shown in in table 3.4.

   The combination0 is configuring that all the functions (fun1, fun2, and fun3) are implemented as software functions so it has zero latency and constant hardware utilization and constant dynamic power which they are the area and the power consumptions of the ARM processor.

   The objective of this thesis is to study the performance of hardware-accelerated functions synthesized by HLS therefore, combination0 is considered an ideal case and removed from the estimate_performance_list.

6) Compare the Calculated (2^n) combination performance metrics from step-5 against the performance metrics constraints defined in the performance_metrics_constrain file, the designer sets the maximum limit of hardware utilization, dynamic power, and latency in the performance_metrics_constrain file.

   Combinations which do not meet the designer's constraints will be rejected, only solutions that passed constraints will be considered in next steps as a valid solution_set.

So, in this case, number of valid solution_set is less than or equal (2^n) depending on designer constraints. Section 4.3 represents implementation of the LTE PDSCH transmitter and receiver as a case-study for the design_flow_2.

Table 3.4: Estimated performance list for all combination: (comb.) is abbreviation to combinations

performance metrics	**comb. 0**	**comb.1**	**comb.2**	**comb.3**	**comb .4**	**comb.5**	**comb.6**	**comb.7**
hardware utilization	0	h1	h2	h3=h1+h2	h4	h5=h1+h4	h6=h2+h4	h7=h1+h2+h4
dynamic power	0	p1	p2	p3=p1+p2	p4	p5=p1+p4	p6=p2+p4	p7=p1+p2+p4
Latency	0	l1	l2	l3=l1+l2	l4	l5=l1+l4	l6=l2+l4	l7=l1+l2+l4

After defining the valid solution_set from a (2^n) combination for design_flow_1 or design_flow_2, the implementation of solution_set is executed as the following steps:

1) Select one of possible solution_set generated from design_flow_1 or design_flow_2 to be input to the SDSoC design flow shown in Figure 3.4.
2) Repeat step-1 until finish implementation of all solution_set defined in step-1.
3) Get the performance metrics of all solution_set. If design_flow_1 is applied, print the output performance metrics results for all solution_set.
4) If design_flow_2 is applied, then read the performance_metrics_constrain file and calculate the Performance Metrics Cost (PMC) of all solution_set as shown in 1.
5) Print the performance metric for the valid solution_set and print the best solution_set that achieve target performance which is the least cost value calculated in step-4.

The Performance Metrics Cost (PMC) is calculated as the follows: -

$$PMC = \frac{|area - area\_t|}{area\_t} * area\_w + \frac{|power - power\_t|}{power\_t} * power\_w + \frac{|\text{latency} - \text{latency}\_t|}{\text{latency}\_t} * \text{latency}\_w \quad (3.1)$$

where $area\_t$ is target area, $area\_w$ is area weight, $power\_t$ is target power and $power\_w$ is power weight, latency_t is target latency and latency_w is latency weight.

The Performance Metrics Cost of solution indicates how far the solution performance is from the target performance. As shown in equation 3.1, the cost equation sets target of area (hardware utilization), power (dynamic power) and latency design constraints, so the SoC designer set the target required performance metrics. Also, the cost equation set weigh of area (hardware utilization), power (dynamic power) and latency design constraints.

Therefore, the SoC designer able to decide which parameter is more important and increase its weight, unimportant parameters weight can be set to zero weight. For example, if dynamic power is important metrics in system design, increase its weight and implement only the solution_set which generate a heterogeneous FPGA-CPU platform that consuming the minimum dynamic power

Chapter 4 : IMPLEMENTION AND RESLULTS

4.1. IMPLEMENTATION OF TURBO ENCODER USING SDSCOC TOOL

This section explains the implementation of turbo encoder on a heterogeneous FPGA-CPU platform using SDSoC tool. The turbo encoder function is written using C programming language and integrated with other functions to verify operation of it.

4.1.1. Turbo encoder block diagram

The design specs of the turbo encoder for LTE was introduced in reference [51]. Figure 4.1 shows the block of the turbo encoder. The turbo encoder is the parallel concatenation of Recursive Systematic Convolutional (RSC) encoder, separated by an interleaver.

The information bits flow goes into the first RSC encoder, and after interleaving, it feeds a second RSC encoder. The multiplexing and puncturing block accepts the information bits and outputs from the RSC encoder to generate the coded bits.

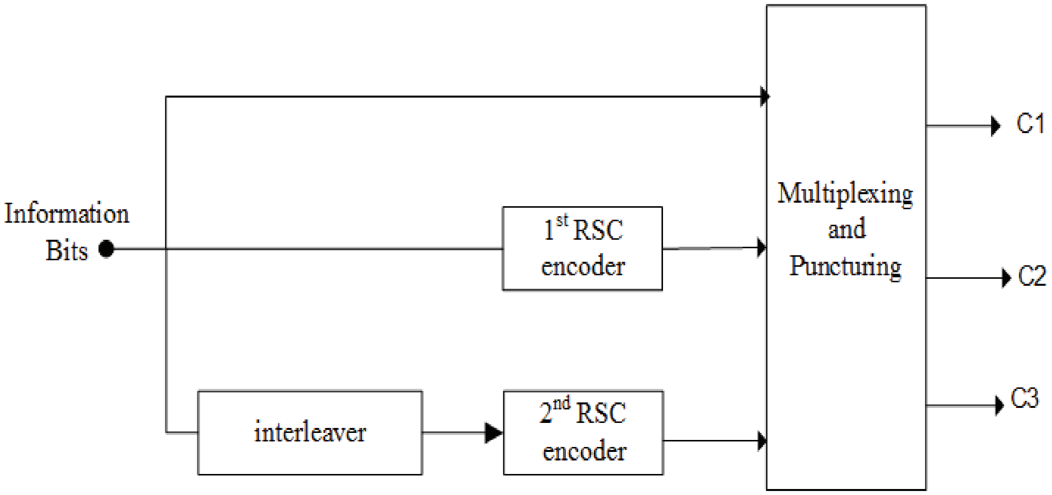


Figure 4.1: Turbo encoder block diagram [51]

4.1.2. Turbo encoder implementation

The objective of this section is to implement multiple scenarios for turbo encoder function. Each scenario generates an Embedded FPGA platform which dependent on the implementation of turbo encoder sub-functions either software function or hardware-accelerated function synthesized by HLS.

Table 4.1 shows all possible configuration scenarios to implement turbo encoder sub-function. For example, in the turbo11 platform, the Two RSC encoder and ineterleaver sub-functions are implemented as a hardware-accelerated function and multiplexer-puncturing sub-function is implemented as a software function.

Table 4.1: Turbo encoder sub-function configuration scenarios: (0) Software function and (1) Hardware-accelerated function

platform name	Interleaver	Mux_punc	RSC_Enc 2	RSC_Enc 1
turbo0	0	0	0	0
turbo1	0	0	0	1
turbo2	0	0	1	0
turbo3	0	0	1	1
turbo4	0	1	0	0
turbo5	0	1	0	1
turbo6	0	1	1	0
turbo7	0	1	1	1
turbo8	1	0	0	0
turbo9	1	0	0	1
turbo10	1	0	1	0
turbo11	1	0	1	1
turbo12	1	1	0	0
turbo13	1	1	0	1
turbo14	1	1	1	0
turbo15	1	1	1	1

4.1.3. Configurable Embedded FPGA Platform

The proposed configurable Embedded FPGA platform is shown in figure 4.2. The configurable embedded FPGA platform consists of a processing system and a programming logic. The processing system side is consisting of fixed implementation functions used for integration and verification of the operation of the turbo encoder function. Example of fixed implementation functions, the random_test function used to generate random information bits, the noise function used to generate AWGN noise, and finally, the main function that integrates all functions together. In addition, the processing system is consisting of the turbo_encoder function that is consist of the configurable implementation functions. The term configurable means that each sub-function of the turbo_encoder function is implemented as a software function or a hardware-accelerated function synthesized by HLS according to the configuration in table 4.1.

The programming logic side is consisting of fixed implemented hardware logic used to handle the data flow between the processing system and programming logic. The fixed logics are generated by SDSoC tool and dependent on the number of connection ports between software functions and hardware-accelerated functions. In addition, the programming logic is consisting of a turbo_encoder function implemented using HLS. As described in section 4.1.2, each sub-function of a turbo_encoder function is implemented as a software function or hardware-accelerated function synthesized by HLS according to the configuration in table 4.1.

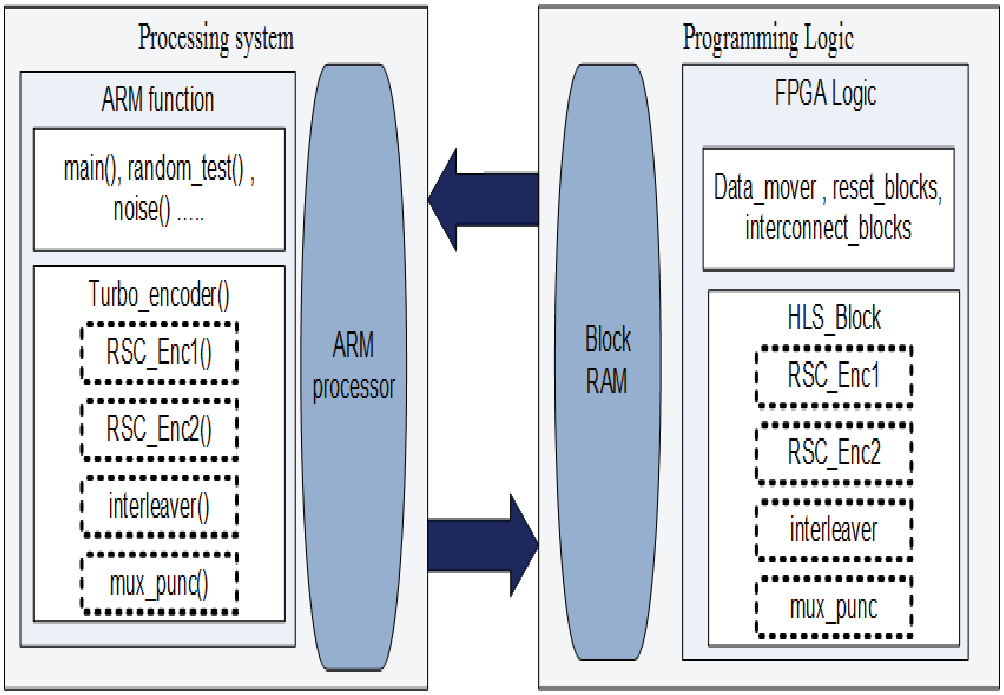


Figure 4.2: Configurable embedded FPGA platform for turbo encoder

4.1.4. Results and Comparative Studies

This section shows the implementation results of all possible configurations scenarios shown in table 4.1. Sixteen projects are generated to cover all possible scenarios between software implementation and hardware-accelerated implementation generated using HLS. Xilinx ZYNC ZC702 device was used for implementation [52]. It consists of dual ARM Cortex A9 core as the processing system and XC7Z020-CLG484 based FPGA as the programming logic.

4.1.4.1. Hardware Utilization for turbo encoder implementation

Figure 4.3 shows the hardware utilization of the generated platforms of the synthesized hardware. The Hardware utilization is sum of number of Look-Up Tables (LUT), number of flip-flops and number of Muxes. The turbo0 platform is the software implementation of all turbo encoder sub-functions so it has zero hardware utilization and not included in Figure 4.3.

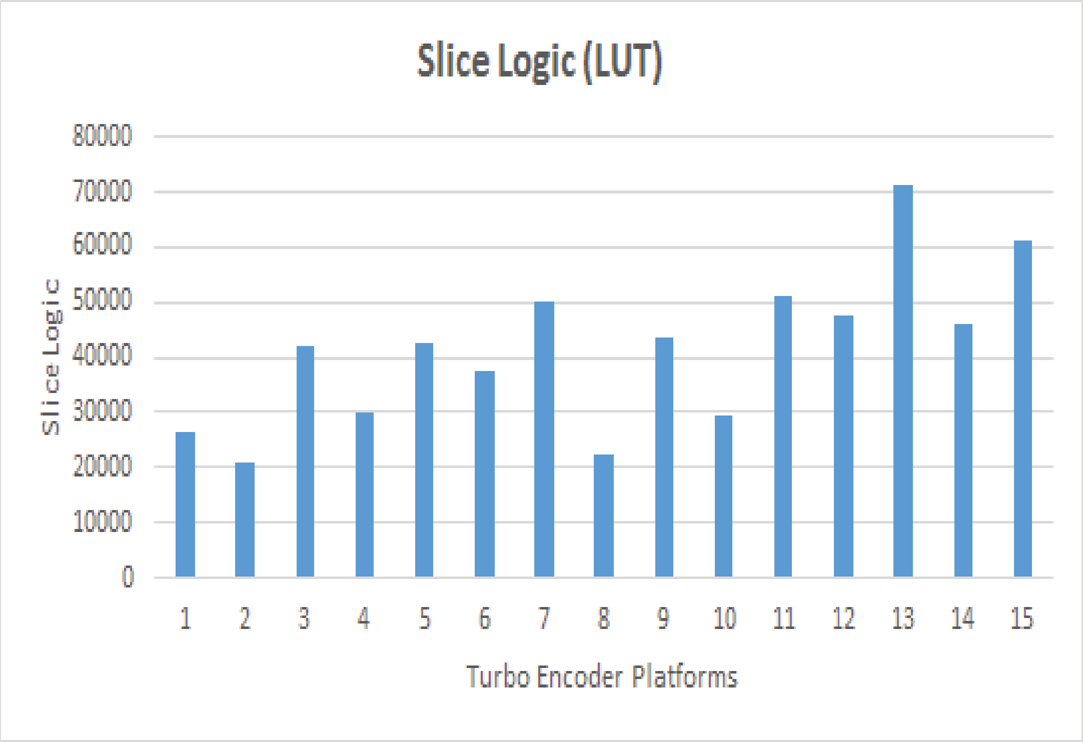


Figure 4.3: Hardware utilization for turbo encoder implementation

The turbo2 platform has the minimum hardware utilization. The turbo2 platform consist of RSC_Enc2 is implemented as a hardware-accelerated function and other turbo encoder sub-functions are implemented as a software function. The turbo13 platform has the maximum hardware utilization. The turbo13 platform consist of RSC_Enc1, interleaver and mux_pun are implemented as a hardware-accelerated functions and RSA_Enc2 is implemented as a software function.

4.1.4.2. Dynamic power for turbo encoder implementation

Power is an important metrics for any communication system. For FPGA platform, power calculation includes the power consumption in the arm processor and the static power calculation. We focus on the dynamic power only, so we subtract the arm processor power and the static power from the total power consumptions.

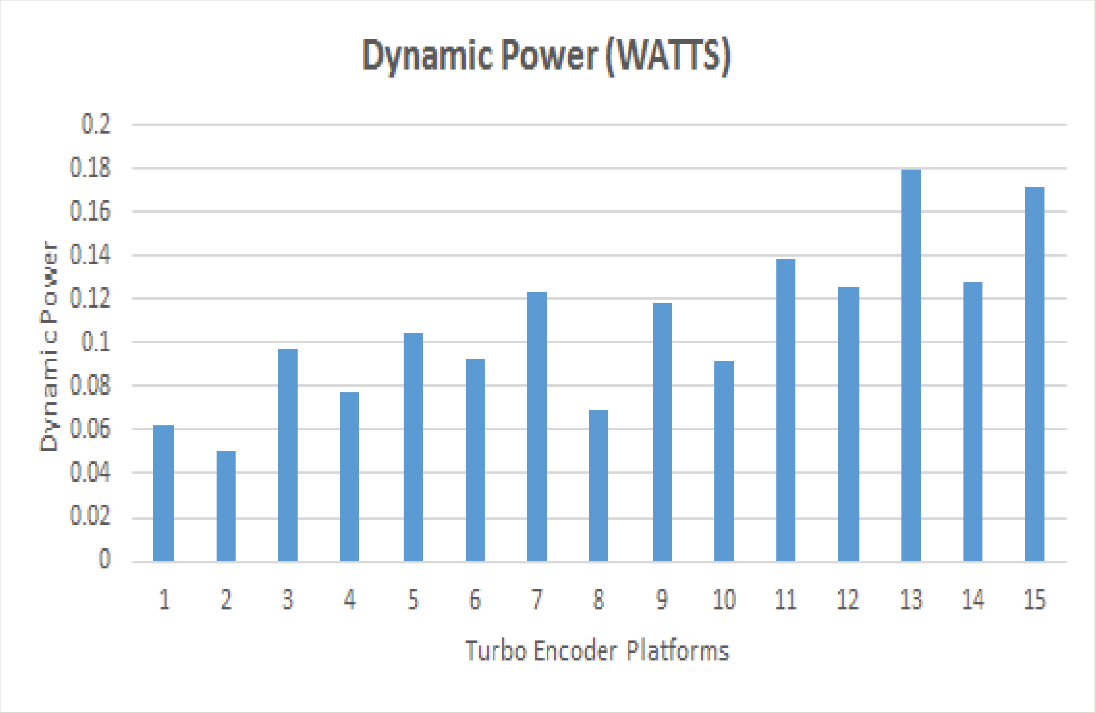


Figure 4.4: Dynamic power for turbo encoder implementation

Figure 4.4 shows the dynamic power for the generated platforms of the synthesized hardware. The dynamic power is measured in watts. The turbo0 platform is the software implementation of all turbo encoder sub-functions so it has zero dynamic power and not included in Figure 4.4.

The turbo2 platform has the minimum dynamic power. The turbo2 platform consists of RSA_Enc2 is implemented as a hardware-accelerated function and other turbo encoder sub-functions are implemented as a software function. The turbo13 platform has the maximum dynamic power. The turbo13 platform consist of RSC_Enc1, interleaver and mux_pun are implemented as a hardware-accelerated functions and RSA_Enc2 is implemented as a software function.

4.1.4.3. Hardware acceleration for turbo encoder implementation

Hardware acceleration is metrics defined by the SDSoC tool [48]. Hardware acceleration is the number of clock cycles improvement in execution of system if implementing the function as a hardware-accelerated function in the programming logic.

Figure 4.5 shows the hardware acceleration for the generated platforms of the synthesized hardware. The turbo0 platform is the software implementation of all turbo encoder sub-functions so the hardware acceleration is not defined and not included in figure 4.5. The turbo2 platform has the maximum hardware acceleration. The turbo2 platform consists of RSA_Enc1, and RSA_Enc2 are implemented as hardware-accelerated functions, and other turbo encoder sub-functions are implemented as a software function. The turbo4 platform has the minimum hardware acceleration. The turbo4 platform consist of mux_punc sub-function is implemented as a hardware-accelerated function and other turbo encoder sub-functions are implemented as software implementation.

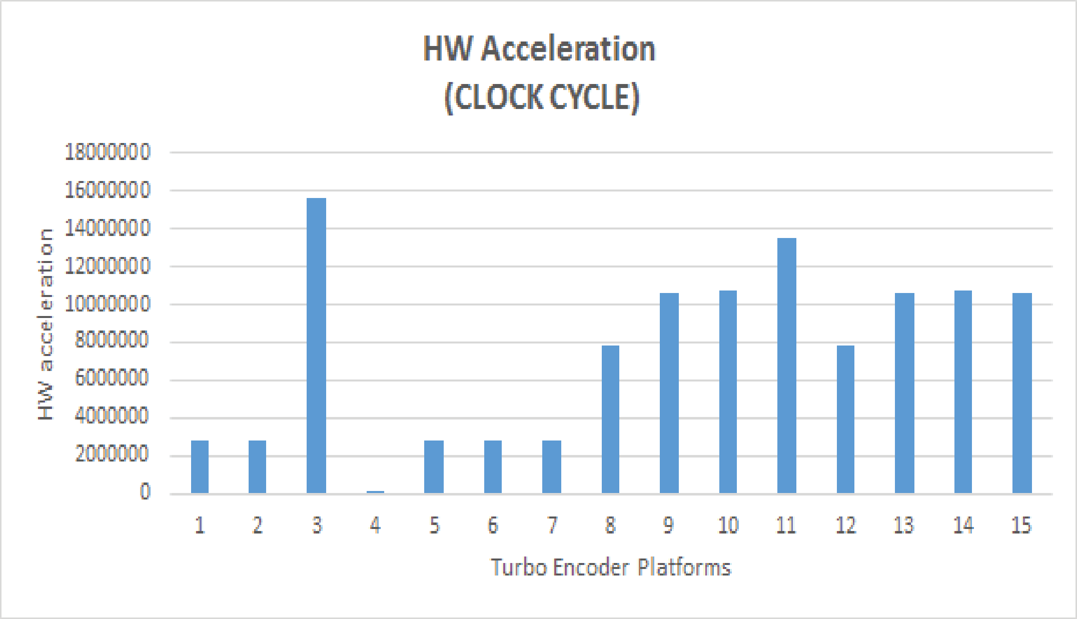


Figure 4.5: Hardware acceleration for turbo encoder implementation

4.1.4.4. Figure of Merit for turbo encoder implementation

The Figure of Merit (FoM) metrics is defined to know the platform that achieves the best overall performance [50]. The FoM metric is defined as follows:

$$FoM = \frac{(acceleration)}{(area) * (power) * (latency)} \tag{4.1}$$

Equation 4.1 shows that acceleration effect directly proportional with FoM performance and shows that area, power and latency effect reversely proportional with FoM performance.

Figure 4.6 shows the FoM calculations of the generated platforms of the synthesized hardware. The turbo2 platform has the best FoM calculation. The turbo2 consist of RSA_Enc1 and RSA_Enc2 are implemented as hardware-accelerated functions, and other turbo encoder sub-functions are implemented as software functions. The turbo7 platform has the worst FoM calculation. The turbo7 platform consist of RSA_Enc1, RSA_Enc2, and mux_punc are implemented as hardware-accelerated functions, and interleaver is implemented as a software function.

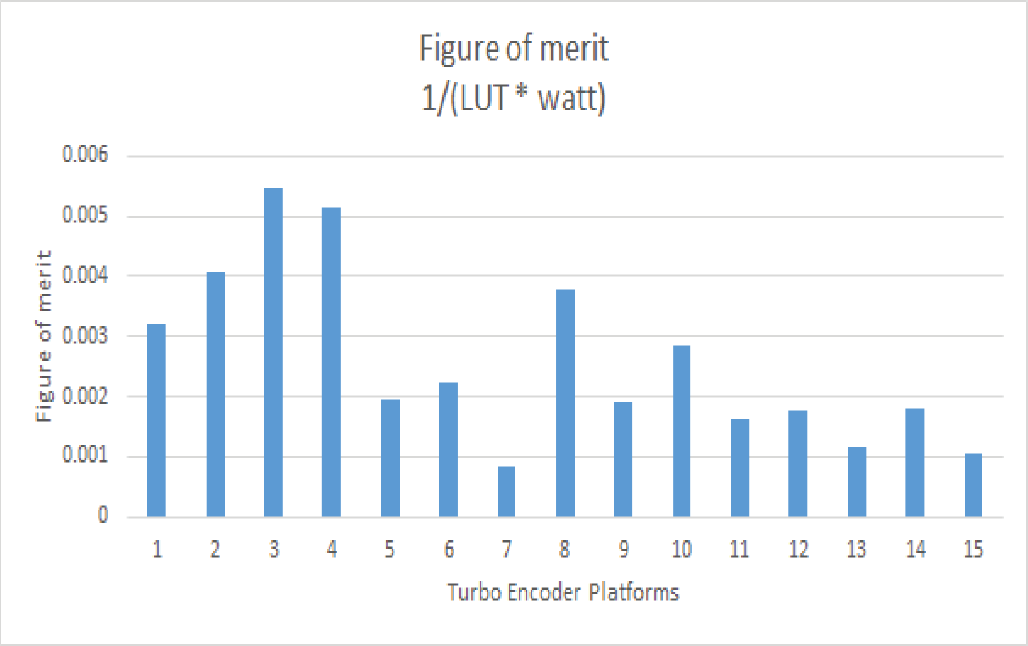


Figure 4.6: FoM for turbo encoder implementation

4.2. IMPLEMENTATION OF THE LTE PDSCH TRANSMITTER AND RECEIVER USING SDSOC TOOL

This section explains the implementation of the LTE PDSCH transmitter and LTE PDSCH receiver on a heterogeneous FPGA- CPU platform applying proposed performance-based design technique using SDSoC tool. The LTE PDSCH transmitter and LTE PDSCH receiver are written using C programming language and integrated into the main function including test- bench function to verify them.

In section 4.2.1, The LTE PDSCH transmitter function implemented with all possible solution scenarios. As LTE PDSCH transmitter consists of eight sub-functions two hundred fifty-six (2^8 = 256) configuration scenarios are generated. Similarity,

In section 4.2.2, The LTE PDSCH receiver function implemented with all possible solution scenarios, as LTE PDSCH receiver consists of eight sub-functions two hundred fifty-six (2^8 = 256) configuration scenarios are generated. In section 4.2.3, constraints are applied to the implementation of the LTE PDSCH transmitter and receiver, so in these cases configuration scenarios that meets constraints are only generated.

4.2.1. LTE PDSCH transmitter implementation

The objective of this section is to implement all multiple scenarios for the LTE PDSCH transmitter function. Each scenario generates an Embedded FPGA platform which depends on the implementation of the LTE PDSCH transmitter sub-functions either software function or hardware-accelerated function synthesized by HLS.

As shown in figure 2.5 the LTE PDSCH transmitter consists of eight sub-functions which are: CRC_addition, segmentation, turbo_encoder, interleaver, rate_matching, scrambler, modulator and resource_element_mapper. Having of eight sub-functions, two hundred and fifty-six projects are generated to cover all possible scenarios between software implementation and hardware-accelerated implementation generated using HLS. Table 4.2 shows all possible configuration scenarios to implement the LTE PDSCH transmitter sub-functions. The number (1) in the table indicates that this sub-function in the current platform is a hardware-accelerated logic and the number (0) indicates that this sub-function in the current platform is a software function. For example, in Tx6 platform the segmentation sub-function and Turbo_encoder sub-function are implemented as a hardware-accelerated logic and other LTE PDSCH transmitter sub-functions are implemented as a software function.

Table 4.2: LTE PDSCH transmitter sub-function configuration scenario: (0) software function and (1) hardware-accelerated logic

platform name	RE mapper	Modulator	Scrambler	Rate matching	Interleaver	Turbo encoder	Segmentation	CRC addtion
Tx0	0	0	0	0	0	0	0	0
Tx1	0	0	0	0	0	0	0	1
Tx2	0	0	0	0	0	0	1	0
Tx3	0	0	0	0	0	0	1	1
Tx4	0	0	0	0	0	1	0	0
Tx5	0	0	0	0	0	1	0	1
Tx6	0	0	0	0	0	1	1	0
Tx7	0	0	0	0	0	1	1	1
. . . .	. . . .	. . . .	. . . .	. . . .	. . . .	. . . .	. . . .	. . . .
Tx250	1	1	1	1	1	0	1	0
Tx251	1	1	1	1	1	0	1	1
Tx252	1	1	1	1	1	1	0	0
Tx253	1	1	1	1	1	1	0	1
Tx254	1	1	1	1	1	1	1	0
Tx255	1	1	1	1	1	1	1	1

4.2.1.1. Configurable embedded FPGA platform for LTE PDSCH transmitter

The proposed configurable Embedded FPGA platform for the LTE PDSCH transmitter is shown in figure 4.7. The configurable embedded FPGA platform consists of the processing system and programming logic. The processing system side consists of fixed implementation functions used for integrating and verifying the operation of the LTE PDSCH transmitter function. Examples of fixed implementation functions, the main function which integrates all functions together, the random_test function used to generate random information bits, the noise function used to generate AWGN noise, ...etc.

On the LTE PDSCH transmitter platform, all of the LTE PDSCH receiver functions configured to be implemented as a fixed software function because they are used in integrating and verifying the LTE PDSCH transmitter/receiver chain. In addition, the processing system consists of the LTE PDSCH Transmitter function that consists of the configurable implementation functions. The term configurable means that each sub-functions of the LTE PDSCH Transmitter function are implemented as a software function or hardware-accelerated function synthesized by HLS according to the configuration in table 4.2.

The programming logic side consists of fixed implemented hardware logic used to handle the data flow between the processing system and programming logic. The fixed logics are generated by SDSoC tool are depends on the number of connection ports between software functions and hardware-accelerated functions synthesized by HLS. In addition, the programming logic consists of LTE PDSCH Transmitter functions implemented using HLS. As described in section 4.2.1, each sub-function of LTE PDSCH Transmitter function is implemented as a software function or hardware-accelerated function synthesized by HLS according to the configuration in table 4.2.

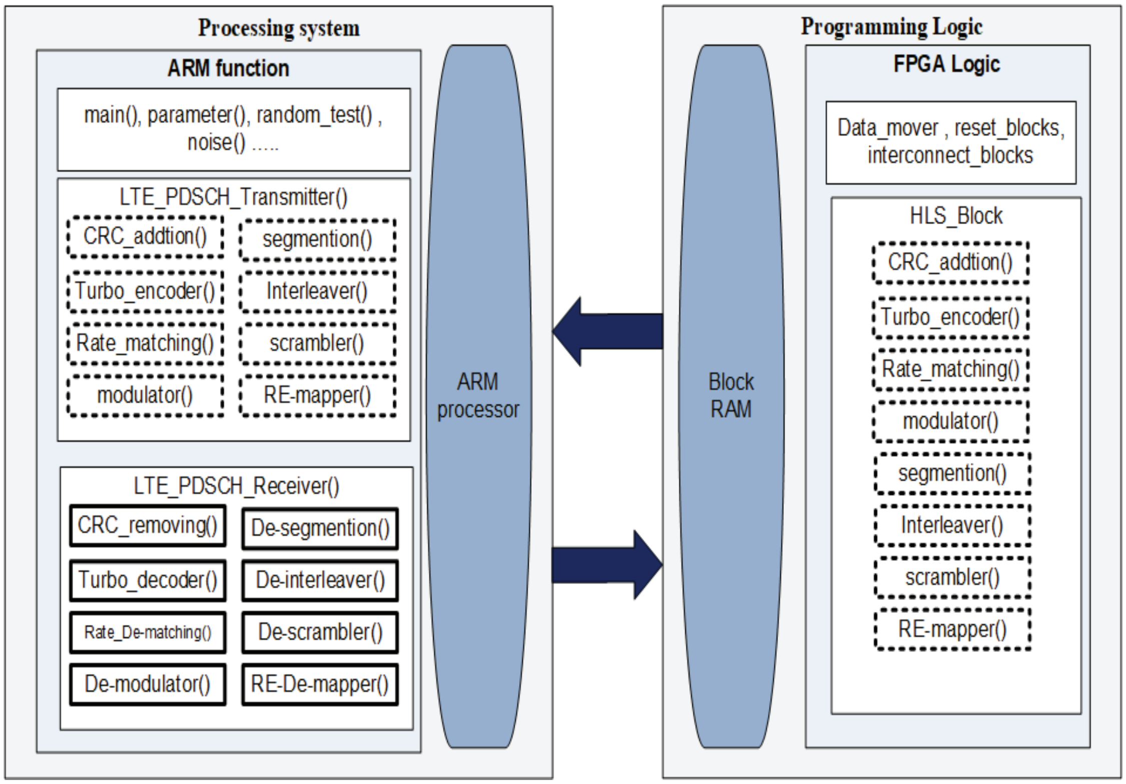


Figure 4.7: Configurable embedded FPGA platform for the LTE PDSCH transmitter

4.2.2. LTE PDSCH receiver implementation

The objective of this section is to implement multiple scenarios for the LTE PDSCH receiver function. Each scenario generates an Embedded FPGA platform which depends on the implementation of the LTE PDSCH receiver sub-functions either software function or hardware-accelerated function synthesized by HLS.

As shown in figure 2.5 the LTE PDSCH receiver consists of eight sub-functions which they are the following: CRC_removing, de-segmentation, turbo_decoder, de-interleaver, rate_de-matching, de-scrambler, de- modulator and resource_element_de-mapper. Having eight sub-functions, two hundred and fifty-six projects are generated to cover all the possible scenarios between software implementation and hardware-accelerated implementation generated using HLS.

Table 4.3 shows parts of all possible configuration scenarios to implement the LTE PDSCH receiver sub-functions. The number (1) in the table indicates that this sub-function in the current platform is a hardware-accelerated logic and the number (0) indicates that this sub-function in the current platform is a software function. For example, in Tx6 platform the de-Segmentation sub-function and Turbo_Decoder sub-function is implemented as a hardware-accelerated logic and other LTE PDSCH receiver sub-functions are implemented as a software function.

Configurable embedded FPGA platform for LTE PDSCH receiver.

Figure 4.8 shows the proposed configurable Embedded FPGA platform for the LTE PDSCH receiver. The configurable embedded FPGA platform consists of the processing system and programming logic. The processing system side consists of fixed implementation functions used for integrating and verifying the operation of the LTE PDSCH transmitter function. Examples of fixed implementation functions, the main function which is integrating all functions together, the random_test function used to generate random information bits, the noise function used to generate AWGN noise, ...etc.

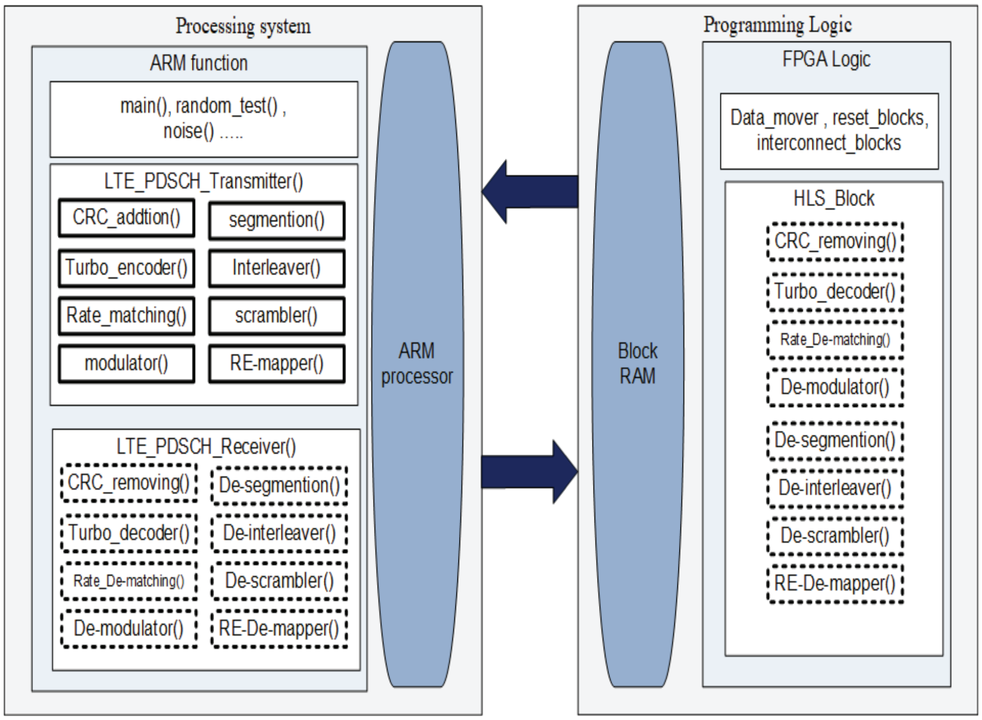


Figure 4.8: Configurable embedded FPGA platform for the LTE PDSCH receiver

Table 4.3: LTE PDSCH receiver sub-unction configuration scenario: (0) software function and (1) hardware-accelerated logic

platform name	RE de-mapper	De-modulator	De-scrambler	Rate de-matching	De-interleaver	Turbo decoder	De-segmentation	CRC removing
Rx0	0	0	0	0	0	0	0	0
Rx1	0	0	0	0	0	0	0	1
Rx2	0	0	0	0	0	0	1	0
Rx3	0	0	0	0	0	0	1	1
Rx4	0	0	0	0	0	1	0	0
Rx5	0	0	0	0	0	1	0	1
Rx6	0	0	0	0	0	1	1	0
Rx7	0	0	0	0	0	1	1	1
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
Rx250	1	1	1	1	1	0	1	0
Rx251	1	1	1	1	1	0	1	1
Rx252	1	1	1	1	1	1	0	0
Rx253	1	1	1	1	1	1	0	1
Rx254	1	1	1	1	1	1	1	0
Rx255	1	1	1	1	1	1	1	1

On the LTE PDSCH Receiver platform, all of the LTE PDSCH transmitter functions configured to be implemented as a fixed software function because they are used in integrating and verifying the LTE PDSCH transmitter/receiver chain. In addition, the processing system consists of the LTE PDSCH receiver function that consists of the configurable implementation functions.

The term configurable means that each sub-function of the LTE PDSCH receiver function is implemented as a software function or hardware-accelerated function synthesized by HLS according to the configuration in table 4.3.

The programming logic side consists of fixed implemented hardware logic used to handle the data flow between the processing system and programming logic. The fixed logics are generated by SDSoC tool and depends on the number of connection ports between software functions and hardware-accelerated functions synthesized by HLS. -

In addition, the programming logic consists of LTE PDSCH receiver functions implemented using HLS. As described in section 4.2.2, each sub-function of LTE PDSCH receiver function is implemented as a software function or hardware-accelerated function synthesized by HLS according to the configuration in table 4.3.

4.2.3. Constraints solution

This section introduces examples for implementation of the LTE PDSCH and LTE PDSCH under user constraints. The design flow applied is design_flow_2 described in section 3.3. The SDSOC.sh script described in Appendix C is modified to add the constraint file and to execute the design_flow_2.

Table 4.4 shows an example of design constraints applied to implement. As shown in the table the, sets of constraints are the following:

- Maximum limit of area, power, and latency. The combinations from table 4.2 or table 4.3 do not meet the designer's maximum limit constraints are rejected; only solutions that meet constraints will be added to the valid solution_sets.
- Target and weight of area, power, and latency. Both of them are used to calculate PMC using equation 3.1.

Table 4.4: Constraints solution examples

Model example	performance_constraints								
	area (LUT)			power (WATSS)			latency (CLOCK CYCLE)		
	Limit	target	weight	limit	target	weight	limit	target	weight
LTE PDSCH transmitter	21700	2000	1	0.22	0.1	1	10^9	10*10^6	1
LTE PDSCH receiver	35000	5200	0	0.249	0.249	0	20*10^6	1.22*10^6	1

4.2.4. Results and comparative studies

This section focuses on studying the performance of hardware-accelerated functions synthesized by HLS. Therefore, the first solution on table 4.2 which is Tx0 platform and the first solution on table 4.3 which is and Rx0 platform are assumed an ideal case. Tx0 and Rx0 solutions are considered as an ideal case because all of the LTE PDSCH transmitter/receiver sub-functions are implemented as a software function. In this case, the hardware-accelerated logics are not generated and the generated platform has a constant area which is the arm processor area and has a constant power consumption which is the power consumption of the arm processor.

The Xilinx ZYNC ZC702 device was used for the implementation. It consists of a dual ARM Cortex A9 core as the processing system and XC7Z020-CLG484 based FPGA as the programming logic [52]. The hardware-accelerated functions are synthesized and implemented at frequency equals 100 MHz.

This section shows the results implementation of all possible configuration scenarios shown in table 4.2. Two hundred and fifty-six projects are generated for the LTE PDSCH transmitter to cover all possible scenarios between software implementation and hardware-accelerated implementation generated using HLS.

In addition, this section shows the implementation results of all possible configuration scenarios shown in table 4.3. Two hundred and fifty-six projects are generated for the LTE PDSCH receiver to cover all possible scenarios between software implementation and hardware-accelerated implementation generated using HLS.

4.2.4.1. LTE TX implementation results

This section shows the results implementation of all possible configuration scenarios for LTE PDSCH transmitter chain.

4.2.4.1.1. Hardware utilization for the LTE TX implementation

Figure 4.9 shows the hardware utilization of the generated platforms of the synthesized hardware. The Tx1 platform has the minimum hardware utilization. The Tx1 platform is configuring such that CRC_addition block is implemented as hardware- accelerated logic and other LTE PDSCH transmitter sub-functions are implemented as software functions. The Tx255 platform has the maximum hardware utilization. The Tx255 platform is configuring such that all the LTE PDSCH transmitter sub-functions are implemented as hardware-accelerated functions. The area results are changed non-linear irregular changes because the different values of area are generated according to configuration in table 4.2.

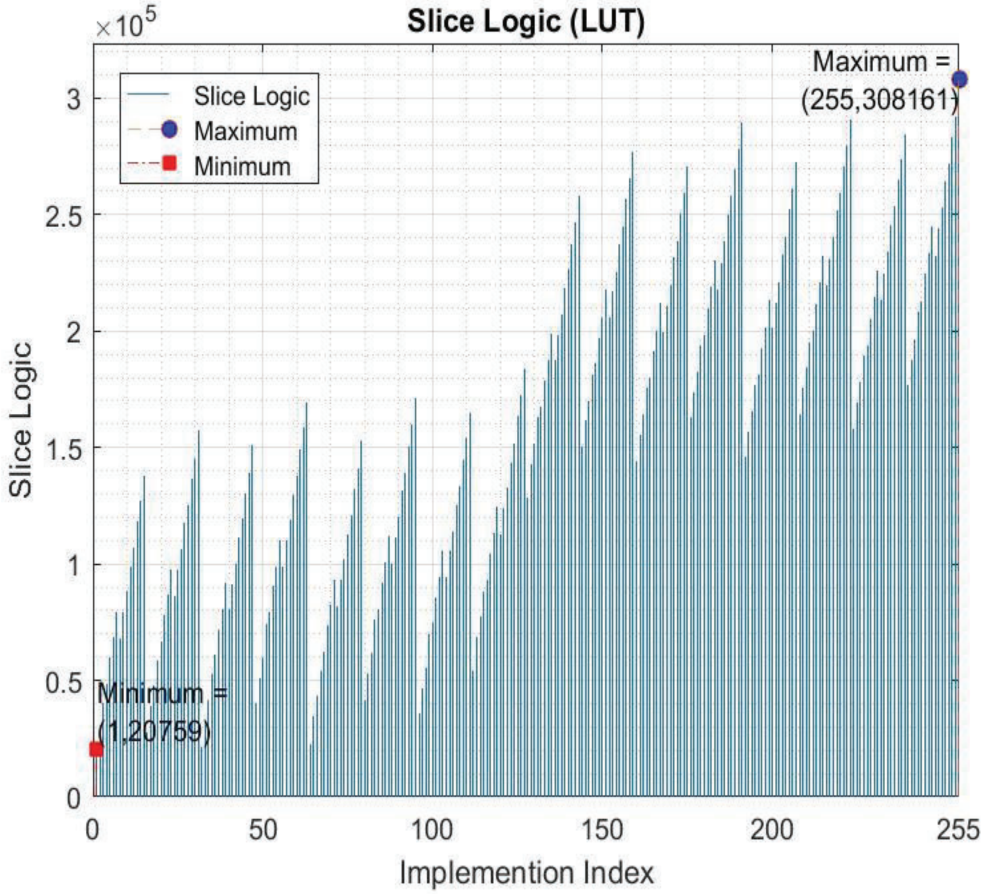


Figure 4.9: Hardware utilization for the LTE TX implementation

4.2.4.1.2. Latency for the LTE TX implementation

Figure 4.10 shows the latency calculation of the generated platforms of the synthesized hardware. The Tx1 platform is configuring such that CRC_addition block is implemented as hardware-accelerated logic and other LTE PDSCH transmitter sub-functions are implemented as software functions. The Tx255 platform has the maximum latency calculation.

The Tx255 platform is configuring such that all the LTE PDSCH transmitter sub-functions are implemented as hardware-accelerated functions. The total latency results are calculated for independent hardware-accelerated functions only, so the processing time of the arm processor is not included in the latency calculation.

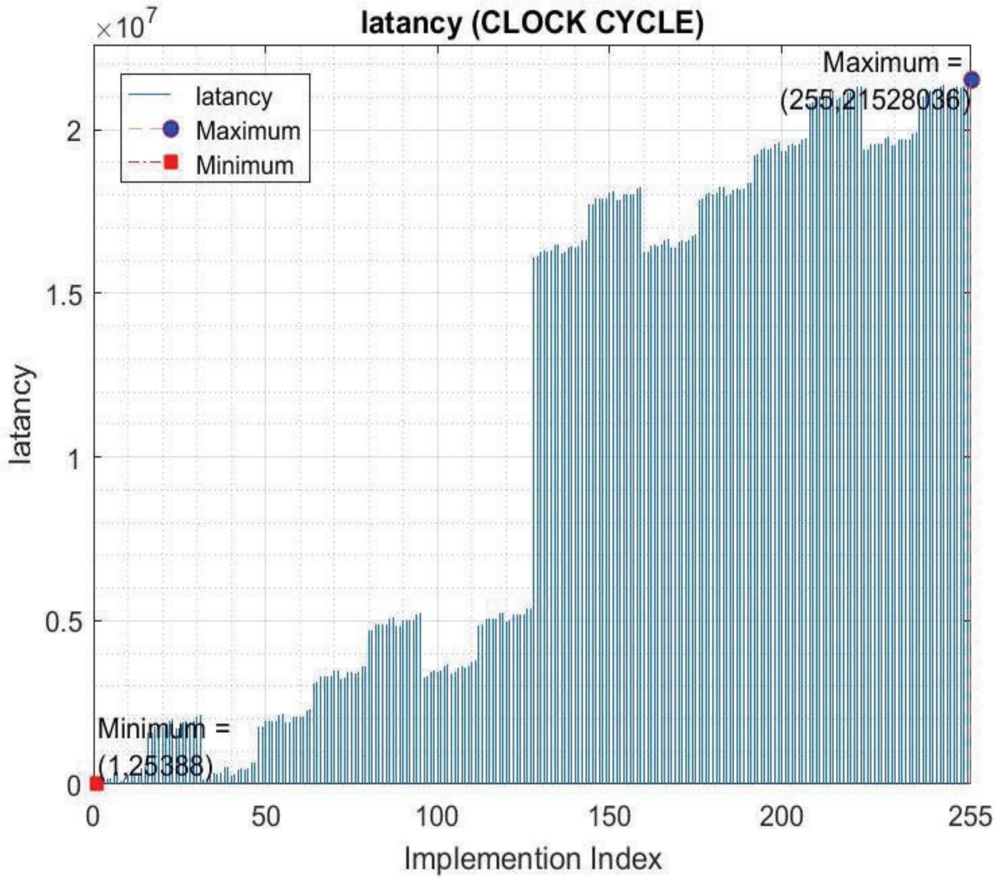


Figure 4.10: Latency for the LTE TX implementation

4.2.4.1.3. Dynamic Power for the LTE TX implementation

Figure 4.11 shows the dynamic power consumption of the generated platforms of the synthesized hardware. The Tx1 platform has the minimum dynamic power consumption. The Tx1 platform is configuring such that CRC_addition block is implemented as hardware-accelerated logic and other LTE PDSCH transmitter sub-functions are implemented as software functions.

The Tx255 platform has the maximum dynamic power consumption. The Tx255 platform is configuring such that all the LTE PDSCH transmitter sub-functions are implemented as hardware-accelerated functions.

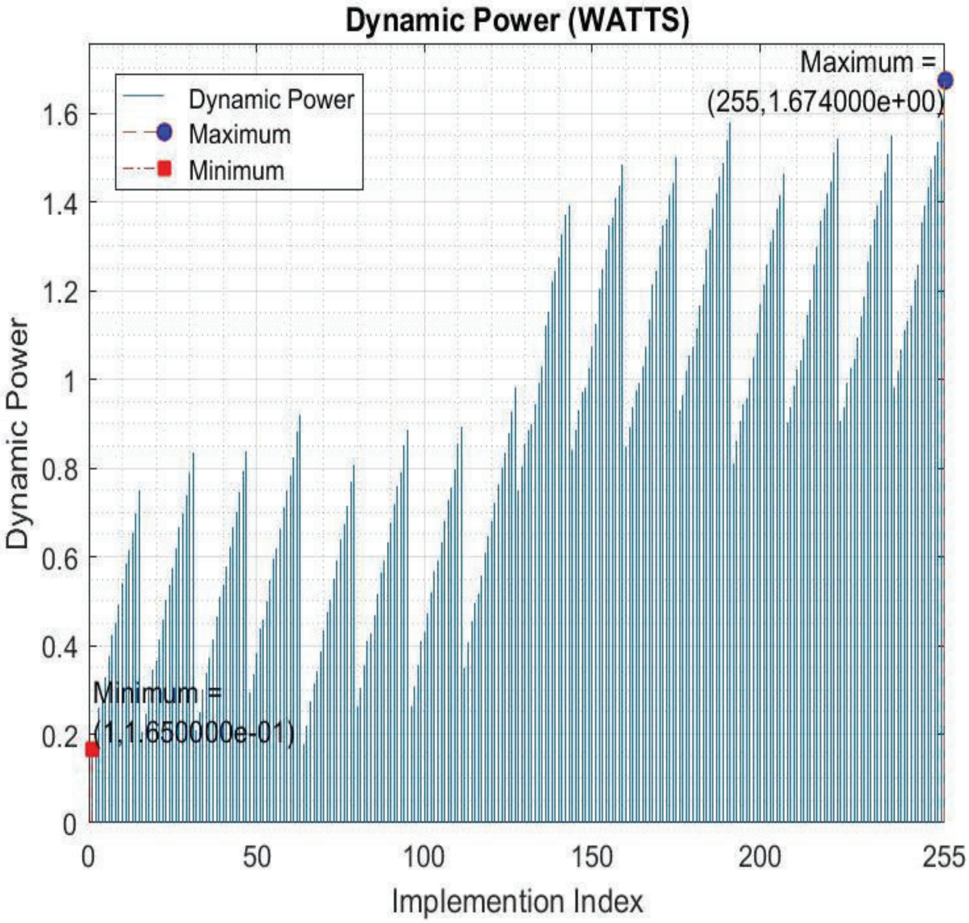


Figure 4.11: Dynamic power for the LTE TX implementation

4.2.4.1.4. Hardware acceleration for the LTE TX implementation

Figure 4.12shows the hardware acceleration calculations of the generated platforms of the synthesized hardware. The Tx1 platform has the minimum hardware acceleration calculation. The Tx1 platform is configuring such that CRC_addition block is implemented as hardware- accelerated logic and other LTE PDSCH transmitter sub-functions are implemented as software functions.

The Tx255 platform has the maximum hardware acceleration calculation. The Tx255 platform is configuring such that all the LTE PDSCH transmitter sub-functions are implemented as hardware-accelerated functions.

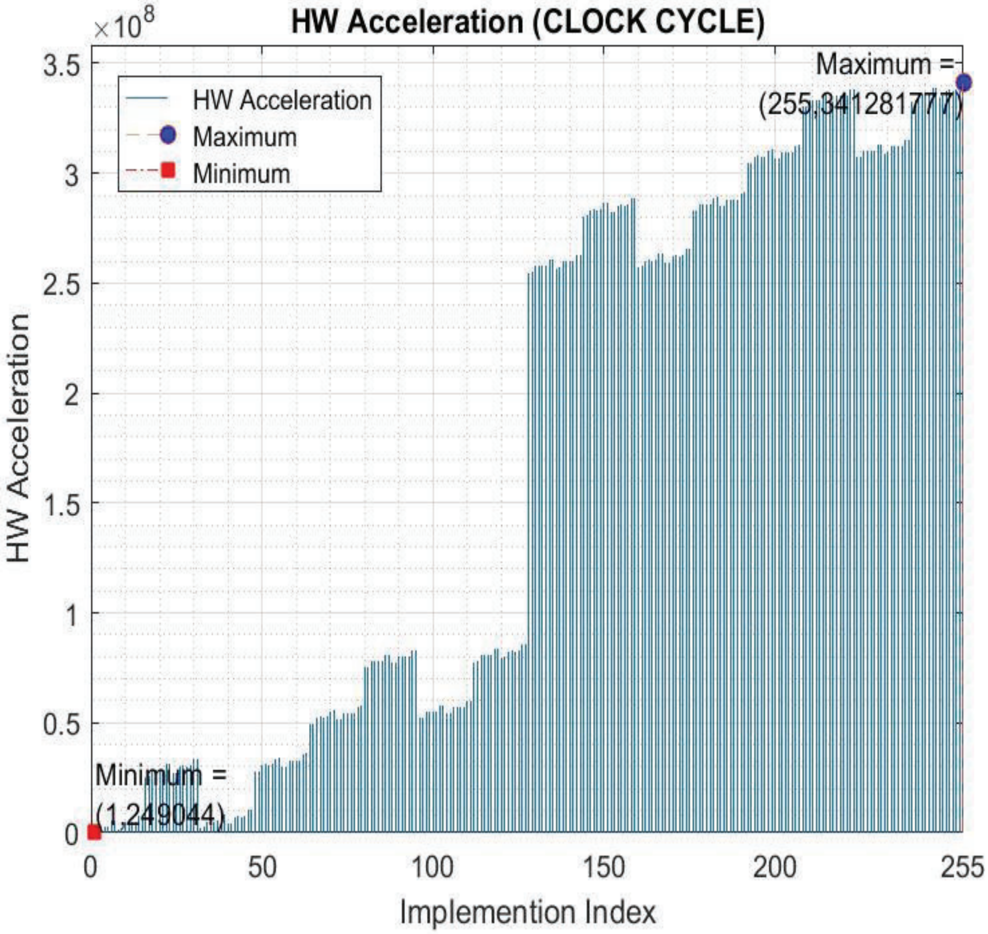


Figure 4.12: Hardware acceleration for the LTE TX implementation

4.2.4.1.5. FoM for the LTE TX implementation

Figure 4.13shows the FoM calculations of the generated platforms of the synthesized hardware. The Tx64 platform has the best FoM calculation. The Tx64 is configuring such that Segmentation block which is implemented as hardware-accelerated logic and other LTE PDSCH transmitter sub-functions sub-functions are implemented as software functions.

The Tx255 platform has the worst FoM calculation. The Tx255 platform is configuring such that all the LTE PDSCH transmitter sub-functions are implemented as hardware-accelerated functions.

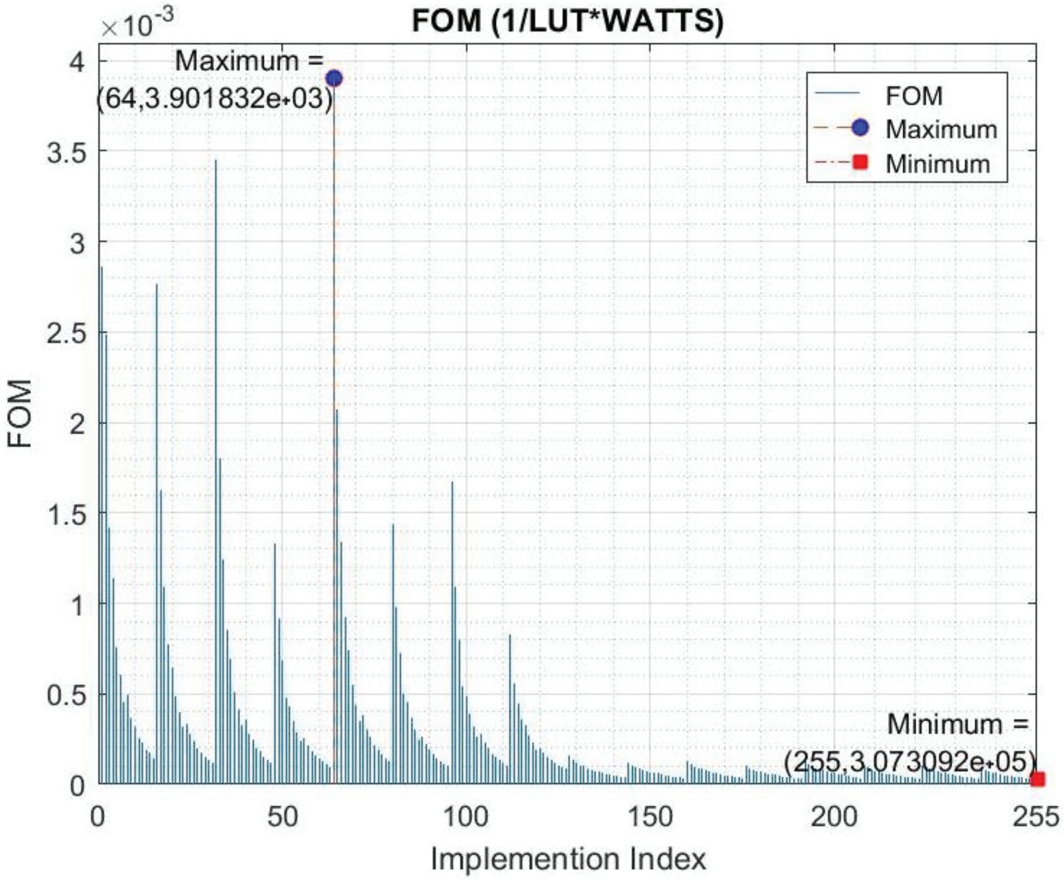


Figure 4.13: FoM for the LTE TX implementation

4.2.4.2. LTE RX implementation results

This section shows the results implementation of all possible configuration scenarios for LTE PDSCH receiver chain.

4.2.4.2.1. Hardware utilization for the LTE T RX implementation

Figure 4.14 shows the hardware utilization of the generated platforms of the synthesized hardware. The Rx1 platform has the minimum hardware utilization. The Rx1 platform is configuring such that CRC_removing block is implemented as hardware-accelerated logic and other LTE PDSCH receiver sub-functions are implemented as software functions. The Rx255 platform has the maximum hardware utilization.

The Rx255 platform is configuring such that all the LTE PDSCH transmitter sub-functions are implemented as hardware-accelerated function. The area results are changed non-linear irregular changes because the different values of area are generated according to configuration in table 4.3.

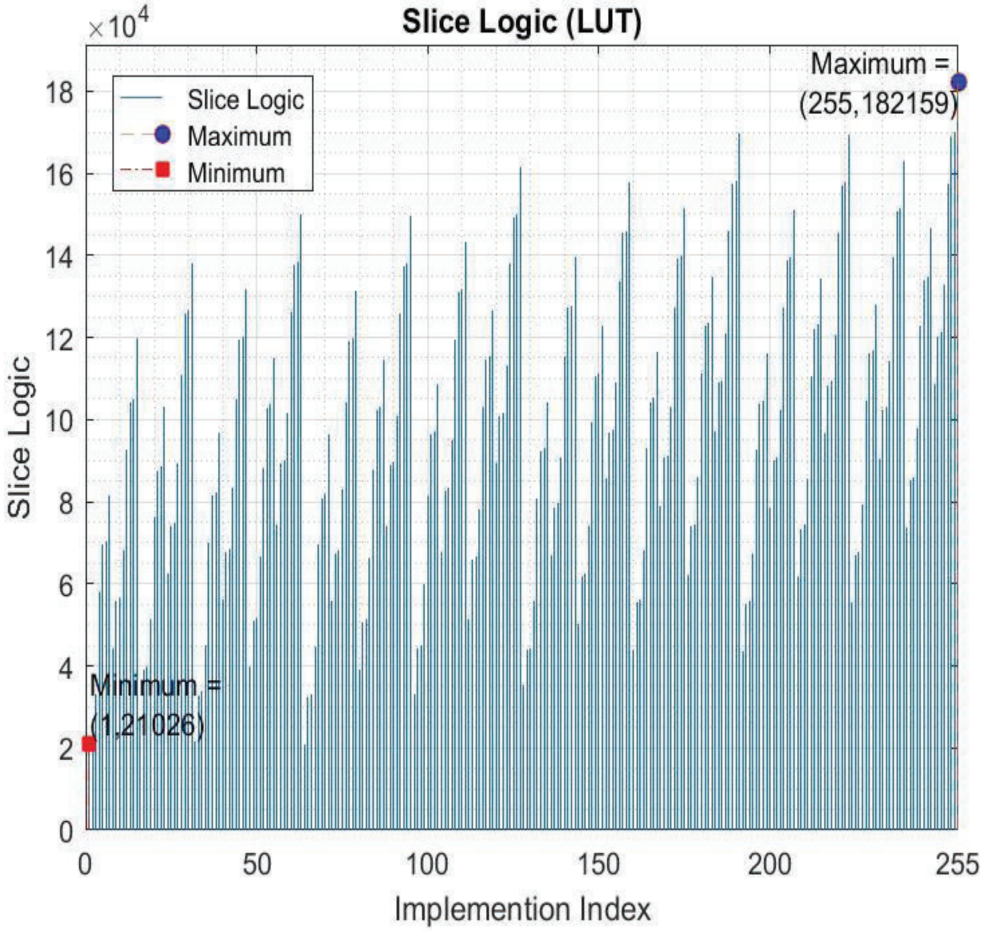


Figure 4.14: Hardware utilization for the LTE RX implementation

4.2.4.2.2. Latency for the LTE RX implementation

Figure 4.15 shows the latency calculation of the generated platforms of the synthesized hardware. The Rx1 platform has the minimum latency calculation. The Rx1 platform is configured such that CRC_removing block which is implemented as hardware-accelerated logic and other LTE PDSCH receiver sub-functions are implemented as software functions. The Rx255 platform has the maximum latency calculation.

The Rx255 platform that is configured such that all the LTE PDSCH receiver sub-functions are implemented as hardware-accelerated functions. The total latency results are calculated for independent hardware-accelerated functions only, so the processing time of the arm processor is not included in the latency calculation.

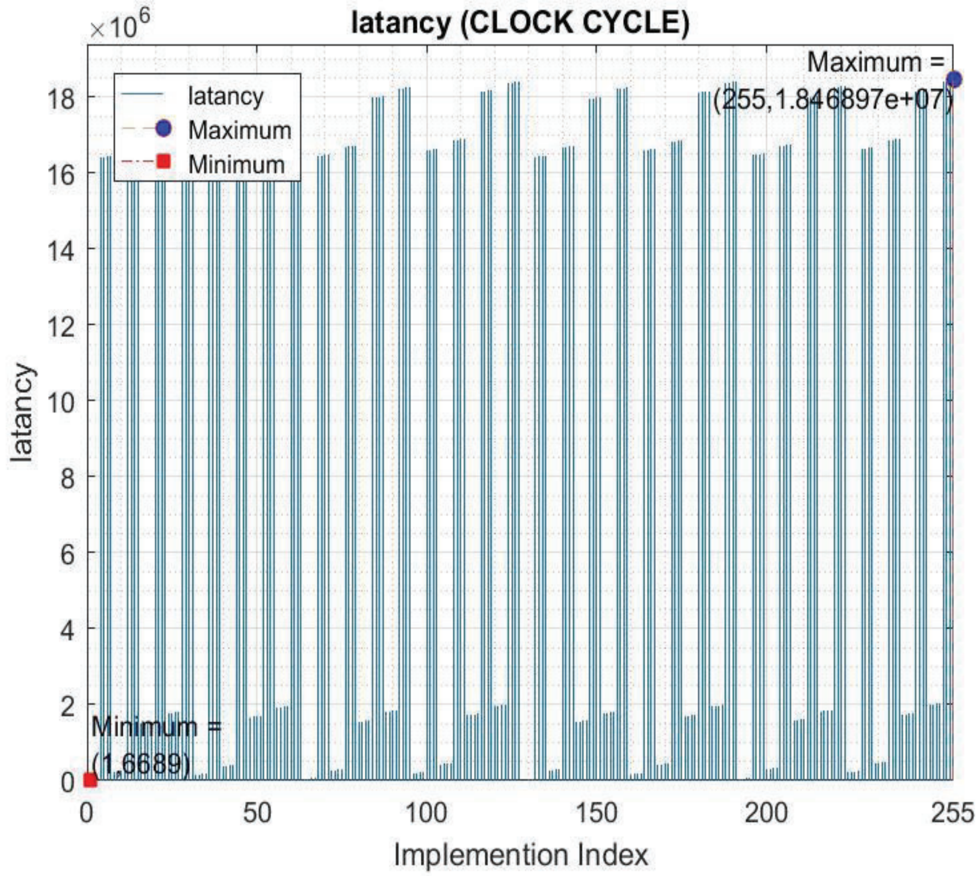


Figure 4.15: latency for the LTE RX implementation

4.2.4.2.3. Dynamic Power for the LTE RX implementation

Figure 4.16 shows the dynamic power consumption of the generated platforms of the synthesized hardware. The Rx1 platform has the minimum dynamic power consumption. The Rx1 platform is configuring such that CRC_removing block is implemented as hardware-accelerated logic and other LTE PDSCH receiver sub-functions are implemented as software functions.

The Rx255 platform has the maximum dynamic power consumption. The Rx255 platform is configuring such that all the LTE PDSCH receiver sub-functions are implemented as hardware-accelerated functions.

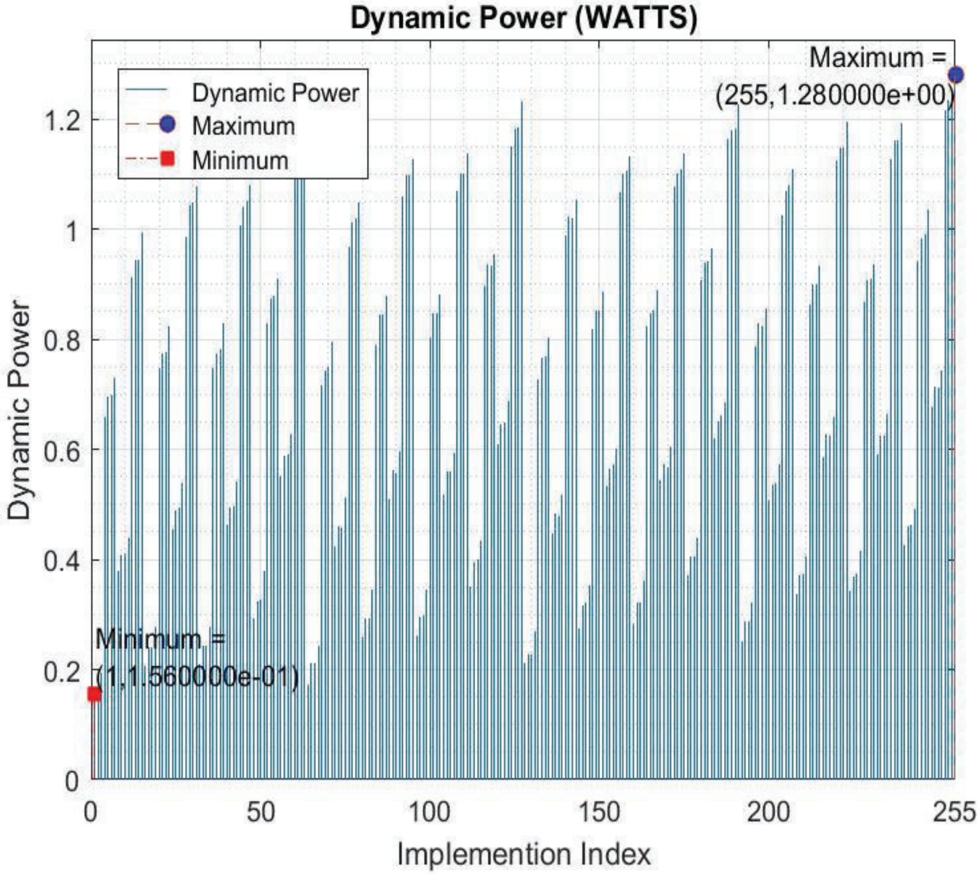


Figure 4.16: Dynamic power for the LTE RX implementation

4.2.4.2.4. Hardware acceleration for the LTE RX implementation

Figure 4.17 shows the hardware acceleration calculations of the generated platforms of the synthesized hardware. The Rx1 platform has the minimum hardware acceleration calculation. The Rx1 platform is configuring such that CRC_addition block is implemented as hardware-accelerated logic and other LTE PDSCH receiver sub-functions are implemented as software functions.

The Rx255 platform has the maximum hardware acceleration calculation. The Rx255 platform is configuring such that all the LTE PDSCH receiver sub-functions are implemented as hardware-accelerated functions.

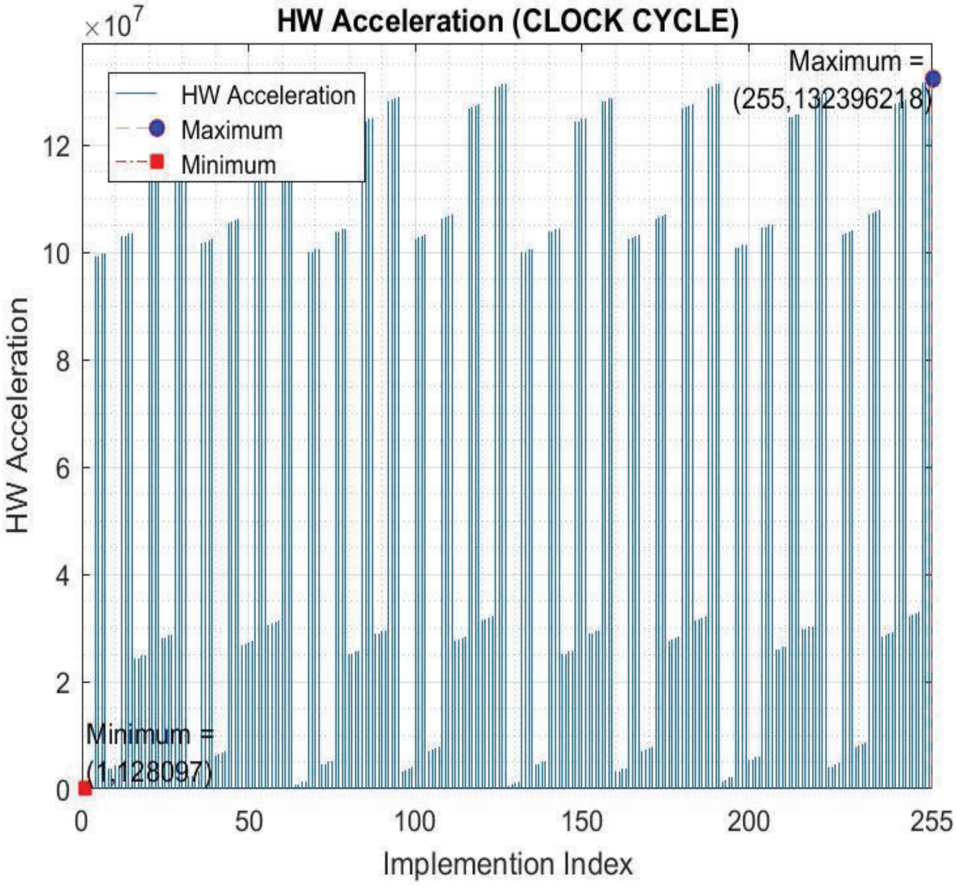


Figure 4.17: Hardware acceleration for the LTE T RX implementation

4.2.4.2.5. FoM for the LTE RX implementation

Figure 4.18 shows the FoM calculations of the generated platforms of the synthesized hardware. The Rx1 platform has the best FoM calculation. The Rx1 platform is configuring such that CRC_addition block is implemented as hardware-accelerated logic and other LTE PDSCH receiver sub-functions are implemented as software functions.

The Rx255 platform has the worst FoM calculation. The Rx255 platform is configuring such that all the LTE PDSCH receiver sub-functions are implemented as hardware-accelerated functions.

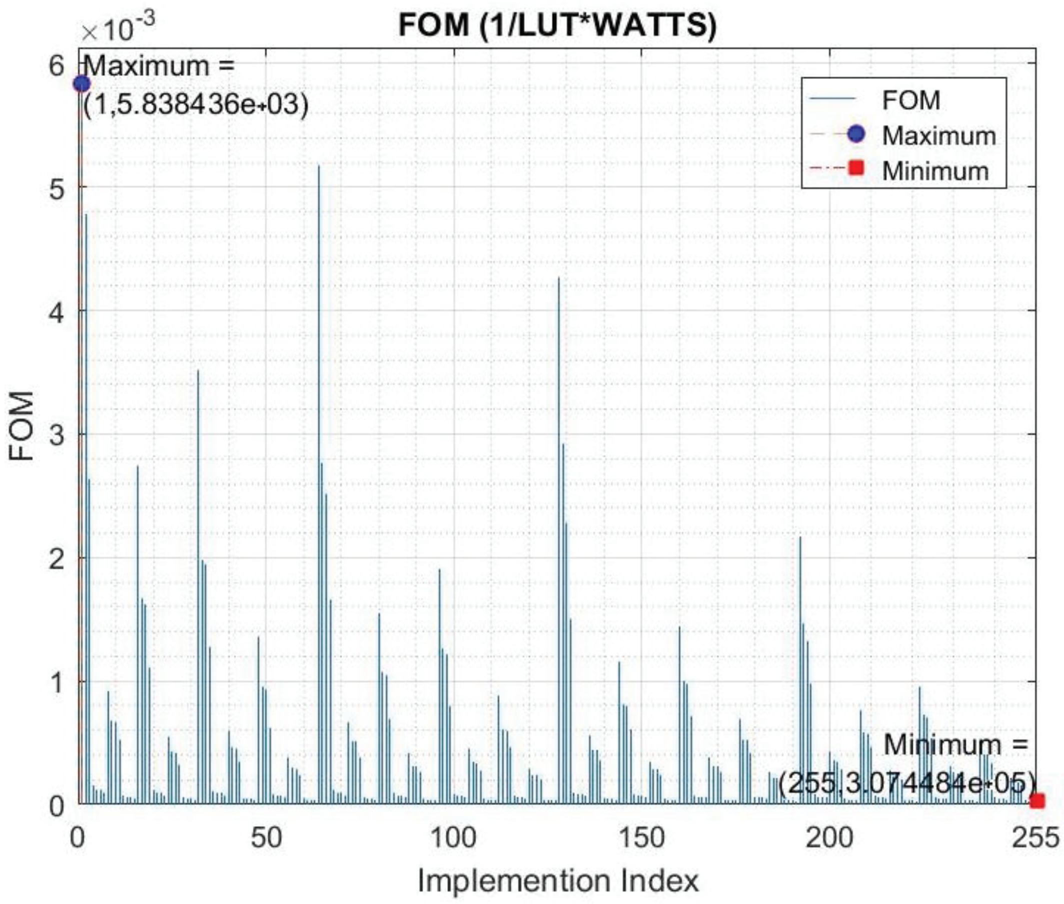


Figure 4.18: FoM for the LTE RX implementation

4.2.4.3. Constraints solution examples results

This section shows the implementation results to get the best heterogeneous FPGA-CPU SoC platform that meets constraints examples in section 4.2.3. The design_flow_2 illustrated in section 3.3 is applied. Performance Metrics Cost (PMC) Equation 3.1 is used for calculating the cost and selecting the best implementation that meets the design constraints.

Table 4.5 shows the implementation results after applying the target constraints in table 4.4. The valid_solution_sets column in tables 4.5 shows the valid solution_sets that meet the maximum limit design constraints in table 4.4. Next calculation the performance metrics cost for each the valid solution_sets using Performance Metrics Cost (PMC) equation 3.1. Finally, select the solution with the minimum calculated PMC value that meets the constraints.

Table 4.5: Constraints solution examples results

Model example	performance_constraints_results		
	valid_solution_sets	PMC	best PMC
LTE PDSCH transmitter	Tx1	9.0320	Tx1
	Tx2	13.9202	
	Tx32	10.1172	
LTE PDSCH receiver	Rx1	0.87540	Rx33
	Rx2	0.527322	
	Rx3	0.4027311	
	Rx4	2.16794	
	Rx16	1.141814	
	Rx32	0.119827	
	Rx33	0.00540819	

For LTE PDSCH transmitter chain, there are three configuration scenarios are met the required performance constraints in table 4.4. The Tx1, Tx2 and Tx32 configuration scenarios are added to the valid_solution_sets list, then the PMC for each configuration scenarios are calculated and then Tx1 was selected as the best configuration scenarios that meets the constraints because it generated the minimum PMC calculation.

For LTE PDSCH receiver chain, there are seven configuration scenarios are met the required performance constraints in table 4.4. The Rx1, Rx2, Rx3, Rx4, Rx16, Rx32 and Tx33 configuration scenarios are added to the valid_solution_sets list, then the PMC for each configuration scenarios are calculated and then Rx33 was selected as the best configuration scenarios that meets the constraints because it generated the minimum PMC calculation.

Conclusions

Designing using SDSoC tool helps SoCs designer by introducing a simple design environment. In addition, SDSoC design environment makes integration and verification of co-design heterogeneous FPGA-CPU faster and more efficient. This thesis introduced a new automated design technique used to implement a heterogeneous FPGA-CPU SoC platform.

In section 3, a new automated design technique is used to implement multiples of heterogeneous FPGA-CPU SoC platforms using SDSoC tool. The automated method is used in exploration of all possible scenarios between software implementation and hardware-accelerated implementation generated using HLS. In addition, the new design technique is used to design a heterogeneous FPGA-CPU SoC platform that meets pre-defined performance metrics constraint such as area and power.

The questions of what platform and what implementation, whether hardware or software is best suited for the best efficient platform. In this thesis, these questions are sought to be answered by introducing Figure of Merit (FoM) performance metric [50]. In addition, Performance Metrics Cost (PMC) equation helps to develop a platform that achieves specific performance metrics requirement.

This new design technique may lead to make quantum leap in the design of heterogeneous FPGA-CPU SoC platform by integrating performance design constraint requirement in the design cycle.

In section 4, as a case study, the LTE PDSCH transmitter/receiver software functions are written using C programming language, and the design of the LTE PDSCH transmitter/receiver are implemented on heterogeneous FPGA-CPU SoC platforms using SDSoC tool. The automated method is used to explorer all possible scenarios between software implementation and hardware-accelerated implementation for the LTE PDSCH transmitter and the LTE PDSCH receiver.

In addition, the platform that meets pre-defined performance metrics constraint is selected for the LTE PDSCH transmitter and the LTE PDSCH receiver. Moreover, the platform that achieves the best overall performance is selected for the LTE PDSCH transmitter and the LTE PDSCH receiver.

Future Work

Adaptive design implementation dependent on performance requirement could be developed and used to re-implement the SoC platform during run time. For example, the designer may develop SoC-based products depending on environmental variables (ex: availability of sunlight). The designer may develop SoC platform consuming high-power assuming availability of sunlight for recharging batteries. Let assume for some reason, the developed SoC-based product have to be work in new environment where sunlight is not available, the SoC platform could be re-implemented during run time to re-build a new SoC platform consuming less power.

Partial Dynamic Reconfiguration (PDR) techniques could be integrated and used to reconfigure the FPGA according to design environment status [53]. For example, platform with the best performance metrics is loaded initially to the FPGA. In case of low power mode, platform with the minimum dynamic power performance metrics is loaded to the FPGA using PDR [54].

In addition, the new design technique may lead to develop FPGA-CPU SoC platform with partially upgrading capability. For example, part of the LTE PDSCH chain has a fixed architecture in every LTE update release (ex: CRC calculation), so this module may be implemented as a hardware-accelerated logic. In the other hand, part of the LTE PDSCH chain has an adjustable architecture in every LTE update release (ex: resource element mapper), so this module may be implemented as a software function because software could be upgraded easily.

References

1. A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi, "Internet of Things for Smart Cities," in IEEE Internet of Things Journal, vol. 1, no. 1, pp. 22-32, Feb. 2014.
2. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 2, pp. 203-215, Feb. 2007.
3. M. D. Edwards and J. Forrest, "Hardware/software partitioning for performance enhancement," IEE Colloquium on Partitioning in Hardware-Software Codesigns, London, UK, pp. 2/1-2/5, 1995.
4. Peng Liua, Jigang Wu , Yongji Wang , “Hybrid algorithms for hardware/software partitioning and scheduling on reconfigurable devices” ,Mathematical and Computer Modelling Volume 58, Issues 1–2, Pages 409-420,2013.
5. Shabtay matalon, “VISTA VIRTUAL PROTOTYPING” , Mentor graphic company white paper, 2015.
6. C. Sekar and Hemasunder, "Tutorial T7: Designing with Xilinx SDSoC," 2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID), Hyderabad, pp. xl-xli, 2017.
7. Alexander Kukushkin, "Global System Mobile, GSM, 2G," in Introduction to Mobile Network Engineering: GSM, 3G-WCDMA, LTE and the Road to 5G, Wiley, pp. 59-102, 2018.
8. K. Riyazuddin, A. K. Sharma and P. V. N. Reddy, "Analyzing the behaviour of OFDM parameters in different LTE environment," 2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI), Chennai, pp. 2849-2853, 2017.
9. Y. Kim, O. Choi, Y. Kim and J. Park, "Performance analysis of LTE multi-antenna technology in live network," 2016 URSI Asia-Pacific Radio Science Conference (URSI AP-RASC), Seoul, pp. 1302-1305, 2016.
10. Alexander Kukushkin, "4G-Long Term Evolution (LTE) System," in Introduction to Mobile Network Engineering: GSM, 3G-WCDMA, LTE and the Road to 5G, Wiley, pp. 205-291, 2018.
11. LaMeres, Brock J., “Introduction to Logic Circuits & Logic Design with Verilog”, Springer International Publishing, 2017.
12. Deming Chen; Jason Cong; Peichan Pan, "FPGA Design Automation: A Survey, in FPGA Design Automation”, Vol. 1, No 3, pp. 195–330, 2006.
13. Dudhe, P.V. & Kadam, N.V. & M. Hushangabade, R & S. Deshmukh, M,"Internet of Things (IOT): An overview and its applications", International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), pp. 2650-2653, 2017.

14. S. Ahmad, V. Boppana, I. Ganusov, V. Kathail, V. Rajagopalan and R. Wittig, "A 16-nm Multiprocessing System-on-Chip Field-Programmable Gate Array Platform," in IEEE Micro, vol. 36, no. 2, pp. 48-62, Mar.-Apr. 2016.
15. Veendrick H., “Memory Circuits and IP. In: Bits on Chips”, Springer, Cham, 2018.
16. Ian Kuon; Russell Tessier; Jonathan Rose, "FPGA Architecture: Survey and Challenges," in FPGA Architecture: Survey and Challenges, 2008.
17. P. J. Kim, D. S. Ku, L. S. Jeong, J. H. Yun, S. Y. Choi and J. B. Kim, "Electrical properties of PIP anti-fuse for the logic circuit configuration," SICE 2003 Annual Conference (IEEE Cat. No.03TH8734), Fukui, Japan, pp. 2980-2983, 2003.
18. P. Alfke, "Xilinx Virtex-6 and Spartan-6 FPGA families," 2009 IEEE Hot Chips 21 Symposium (HCS), Stanford, CA, pp. 1-20, 2009.
19. Sarah L.Harris, David MoneyHarris, “Digital Design and Computer Architecture”, Elsevier, Pages 238-293, 2013.
20. J. Lin and B. C. Lai, "BRAM efficient multi-ported memory on FPGA," VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, pp. 1-4, 2015.
21. U. Farooq, I. Baig and B. A. Alzahrani, "An Efficient Inter-FPGA Routing Exploration Environment for Multi-FPGA Systems," in IEEE Access, vol. 6, pp. 56301-56310, 2018.
22. B. Ronak and S. A. Fahmy, "Multipumping Flexible DSP Blocks for Resource Reduction on Xilinx FPGAs," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 36, no. 9, pp. 1471-1482, Sept. 2017.
23. Clive MaxMaxfield, "FPGAs: Instant Access", Elsevier, 2008.
24. P. S. Zuchowski, C. B. Reynolds, R. J. Grupp, S. G. Davis, B. Cremen and B. Troxel, "A hybrid ASIC and FPGA architecture," IEEE/ACM International Conference on Computer Aided Design, pp. 187-194, 2002.
25. 3GPP, TS 36.101, “Evolved Universal Terrestrial Radio User Equipment (UE) radio transmission and reception”, Release 14, 2017.
26. 3GPP, TS 36.104, “Evolved Universal Terrestrial Radio Base Station (BS) radio transmission and reception”, Release 14, 2017.
27. 3GGP, “Technical Specification Group Radio Access Network; GSM/EDGE Physical layer on the radio path”, Release 14, 2017.
28. S. Syed Ameer Abbas, K. S. Geethu. and S. J. Thiruvengadam, "Implementation of physical downlink control channel (PDCCH) FOR LTE using FPGA," 2012 International Conference on Devices, Circuits and Systems (ICDCS), Coimbatore, pp. 335-339, 2012.
29. L. Kuchibhotla, R. Ghosh, A. Ratasuk, R. Classon, B. Blankenship, “Downlink control channel design for 3GPP LTE”, Wireless Communications and Networking Conference (WCNC), 813-818, 2008.
30. S. S. A. Abbas, P. A. J. Sheeba and S. J. Thiruvengadam, "Design of downlink PDSCH architecture for LTE using FPGA," 2011 International Conference on Recent Trends in Information Technology (ICRTIT), Chennai, Tamil Nadu, pp. 947-952, 2011.

31. Zarrinkoub Houman, “UNDERSTANDING LTE WITH MATLAB” John Wiley & Sons, 2014.
32. 3GPP TS 36.211, “Evolved Universal Terrestrial Radio Access (EUTRA); Multiplexing and Channel Coding” Release 15, 2017.
33. J. Cheng and H. Koorapaty, "Error Detection Reliability of LTE CRC Coding," 2008 IEEE 68th Vehicular Technology Conference, Calgary, BC, pp. 1-5, 2008.
34. K. G. Lenzi, J. A. B. Filho and F. A. P. Figueiredo, "Code block segmentation hardware architecture for LTE-Advanced," IEEE Wireless Communications and Networking Conference (WCNC), Shanghai, pp. 3312-3317, 2013.
35. R. Kaur, S. Chopra, “Iterative Decoding of Turbo Codes”, International Journal of Scientific and Engineering Research (IJSER), 2013.
36. 3GPP TS 36.211, “Evolved Universal Terrestrial Radio Access (EUTRA); Physical Channels and Modulation”, Release 15, 2017.
37. Sah, Dhaneshwar, “Iterative Decoding of Turbo Codes”, Journal of Advanced College of Engineering and Management, Vol.3, pp. 5-30, 2017.
38. Fu-Gang Wang, Yi Tang and Fan Yang, "The iterative decoding algorithm research of Turbo Product Codes," The 2010 International Conference on Apperceiving Computing and Intelligence Analysis Proceeding, Chengdu, pp. 97-100, 2010.
39. M. Bukris, I. Gazit, “Rate Matching and De-Rate Matching for an LTE Transport Channel”, U.S Patent, 2010.
40. ETR 289,” Support for use of scrambling and Conditional Access (CA) within digital broadcast systems”, European Telecommunications Standards Institute (ETSI), 1996.
41. K. G. Digish and R. Thilagavathy, "ASIC implementation of physical downlink shared channel for LTE," 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), Kanyakumari, pp. 370-376, 2014.
42. Jagdish, D. Kenea, Kishor, D. Kulatb, “Soft Output Decoding Algorithm for Turbo Codes Implementation in Mobile Wi-Max Environment”, International Conference on Communication, Computing and Security (ICCCS), Volume 6, PP. 666-673, 2012.
43. D. Zhu, V. J. Mathews and D. H. Detienne, "A Likelihood-Based Algorithm for Blind Identification of QAM and PSK Signals," in IEEE Transactions on Wireless Communications, vol. 17, no. 5, pp. 3417-3430, May 2018.
44. S. S. A. Abbas, S. J. Thiruvengadam and M. Punitha, "Realization of PDSCH transmitter and receiver architecture for 3GPP-LTE advanced," 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Chennai, pp. 1-6, 2016.
45. Valido, Manuel & Magdaleno, Eduardo & Perez, Fernando & García, Cristhian. “Automated Software Acceleration in Programmable Logic for an Efficient NFFT Algorithm Implementation: A Case Study”. Sensors - Open Access Journal, 2017.

46. K. Rupnow, Yun Liang, Yinan Li and Deming Chen, "A study of high-level synthesis: Promises and challenges," 2011 9th IEEE International Conference on ASIC, Xiamen, pp. 1102-1105, 2011.

47. Inc. Xilinx, “Vivado Design Suite, Tutorial High-Level Synthesis (UG871)”, 2014.

48. Inc. Xilinx, “SDSoC Environment User Guide (UG1028)”. 2016.

49. S. Roh, K. Cho and K. Chung, "Implementation of an LDPC decoder on a heterogeneous FPGA-CPU platform using SDSoC," IEEE Region 10 Conference (TENCON), Singapore, 2016, pp. 2555-2558, 2016.

50. M. E. Adawy, A. Kamaleldin, H. Mostafa and S. Said, "Performance evaluation of turbo encoder implementation on a heterogeneous FPGA-CPU platform using SDSoC," 2017 Intl Conf on Advanced Control Circuits Systems (ACCS) Systems & 2017 Intl Conf on New Paradigms in Electronics & Information Technology (PEIT), Alexandria, pp. 286-290, 2017.

51. C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1," Proceedings of ICC '93 - IEEE International Conference on Communications, Geneva, Switzerland, pp. 1064-1070 vol.2, 1993.

52. Inc. Xilinx, “ZC702 Evaluation Board for the Zynq-7000 XC7Z020 User Guide”, 2018.

53. T. Kalb and D. Göhringer, "Enabling dynamic and partial reconfiguration in Xilinx SDSoC," 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, pp. 1-7, 2016.

54. Sadek, A., 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 Systems, pp. 1-12, 2017.

Appendix A: Modulation lookup table

Table A.1: 64QAM modulation lookup table

$f_i, f_{i+1}, f_{i+2}, f_{i+3}, f_{i+4}, f_{i+5}$	I	Q
000000	$\frac{1}{\sqrt{42}}$	$\frac{1}{\sqrt{42}}$
000001	$\frac{1}{\sqrt{42}}$	$\frac{3}{\sqrt{42}}$
000010	$\frac{1}{\sqrt{42}}$	$\frac{5}{\sqrt{42}}$
000011	$\frac{1}{\sqrt{42}}$	$\frac{7}{\sqrt{42}}$
000100	$\frac{3}{\sqrt{42}}$	$\frac{1}{\sqrt{42}}$
000101	$\frac{3}{\sqrt{42}}$	$\frac{3}{\sqrt{42}}$
000110	$\frac{3}{\sqrt{42}}$	$\frac{5}{\sqrt{42}}$
000111	$\frac{3}{\sqrt{42}}$	$\frac{7}{\sqrt{42}}$
001000	$\frac{5}{\sqrt{42}}$	$\frac{1}{\sqrt{42}}$
001001	$\frac{5}{\sqrt{42}}$	$\frac{3}{\sqrt{42}}$
001010	$\frac{5}{\sqrt{42}}$	$\frac{5}{\sqrt{42}}$
001011	$\frac{5}{\sqrt{42}}$	$\frac{7}{\sqrt{42}}$
001100	$\frac{7}{\sqrt{42}}$	$\frac{1}{\sqrt{42}}$
001101	$\frac{7}{\sqrt{42}}$	$\frac{3}{\sqrt{42}}$

001110	$\frac{7}{\sqrt{42}}$	$\frac{5}{\sqrt{42}}$
$f_i, f_{i+1}, f_{i+2}, f_{i+3}, f_{i+4}, f_{i+5}$	I	Q
001111	$\frac{7}{\sqrt{42}}$	$\frac{7}{\sqrt{42}}$
010000	$\frac{1}{\sqrt{42}}$	$\frac{-1}{\sqrt{42}}$
010001	$\frac{1}{\sqrt{42}}$	$\frac{-3}{\sqrt{42}}$
-010010	$\frac{1}{\sqrt{42}}$	$\frac{-5}{\sqrt{42}}$
010011	$\frac{1}{\sqrt{42}}$	$\frac{-7}{\sqrt{42}}$
010100	$\frac{3}{\sqrt{42}}$	$\frac{-1}{\sqrt{42}}$
010101	$\frac{3}{\sqrt{42}}$	$\frac{-3}{\sqrt{42}}$
010110	$\frac{3}{\sqrt{42}}$	$\frac{-5}{\sqrt{42}}$
010111	$\frac{3}{\sqrt{42}}$	$\frac{-7}{\sqrt{42}}$
011000	$\frac{5}{\sqrt{42}}$	$\frac{-1}{\sqrt{42}}$
011001	$\frac{5}{\sqrt{42}}$	$\frac{-3}{\sqrt{42}}$
011010	$\frac{5}{\sqrt{42}}$	$\frac{-5}{\sqrt{42}}$
011011	$\frac{5}{\sqrt{42}}$	$\frac{-7}{\sqrt{42}}$
011100	$\frac{7}{\sqrt{42}}$	$\frac{-1}{\sqrt{42}}$
011101	$\frac{7}{\sqrt{42}}$	$\frac{-3}{\sqrt{42}}$
011110	$\frac{7}{\sqrt{42}}$	$\frac{-5}{\sqrt{42}}$

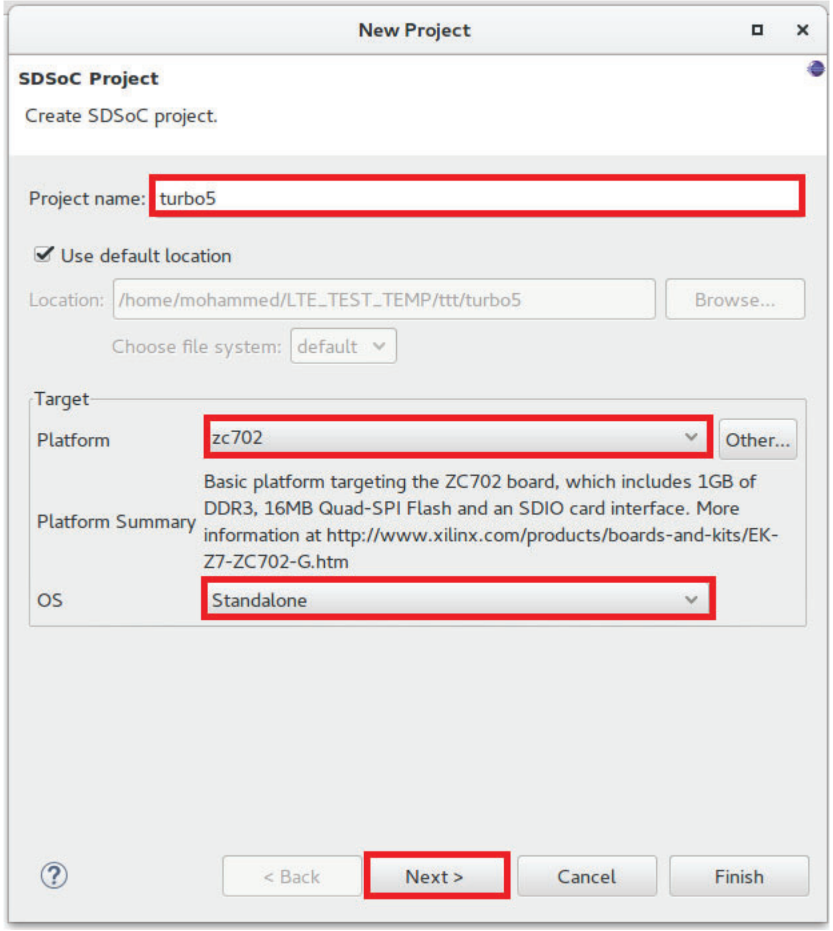
$f_i, f_{i+1}, f_{i+2}, f_{i+3}, f_{i+4}, f_{i+5}$	***I***	***Q***
011111	$\frac{7}{\sqrt{42}}$	$\frac{-7}{\sqrt{42}}$
100000	$\frac{-1}{\sqrt{42}}$	$\frac{1}{\sqrt{42}}$
100001	$\frac{-1}{\sqrt{42}}$	$\frac{3}{\sqrt{42}}$
100010	$\frac{-1}{\sqrt{42}}$	$\frac{5}{\sqrt{42}}$
100011	$\frac{-1}{\sqrt{42}}$	$\frac{7}{\sqrt{42}}$
100100	$\frac{-3}{\sqrt{42}}$	$\frac{1}{\sqrt{42}}$
100101	$\frac{-3}{\sqrt{42}}$	$\frac{3}{\sqrt{42}}$
100110	$\frac{-3}{\sqrt{42}}$	$\frac{5}{\sqrt{42}}$
100111	$\frac{-3}{\sqrt{42}}$	$\frac{7}{\sqrt{42}}$
101000	$\frac{-5}{\sqrt{42}}$	$\frac{1}{\sqrt{42}}$
101001	$\frac{-5}{\sqrt{42}}$	$\frac{3}{\sqrt{42}}$
101010	$\frac{-5}{\sqrt{42}}$	$\frac{5}{\sqrt{42}}$
101011	$\frac{-5}{\sqrt{42}}$	$\frac{7}{\sqrt{42}}$
101100	$\frac{-7}{\sqrt{42}}$	$\frac{1}{\sqrt{42}}$
101101	$\frac{-7}{\sqrt{42}}$	$\frac{3}{\sqrt{42}}$
101110	$\frac{-7}{\sqrt{42}}$	$\frac{5}{\sqrt{42}}$
101111	$\frac{-7}{\sqrt{42}}$	$\frac{7}{\sqrt{42}}$

$f_i, f_{i+1}, f_{i+2}, f_{i+3}, f_{i+4}, f_{i+5}$	I	Q
110000	$\frac{-1}{\sqrt{42}}$	$\frac{-1}{\sqrt{42}}$
110001	$\frac{-1}{\sqrt{42}}$	$\frac{-3}{\sqrt{42}}$
110010	$\frac{-1}{\sqrt{42}}$	$\frac{-5}{\sqrt{42}}$
110011	$\frac{-1}{\sqrt{42}}$	$\frac{-7}{\sqrt{42}}$
110100	$\frac{-3}{\sqrt{42}}$	$\frac{-1}{\sqrt{42}}$
110101	$\frac{-3}{\sqrt{42}}$	$\frac{-3}{\sqrt{42}}$
110110	$\frac{-3}{\sqrt{42}}$	$\frac{-5}{\sqrt{42}}$
110111	$\frac{-3}{\sqrt{42}}$	$\frac{-7}{\sqrt{42}}$
111000	$\frac{-5}{\sqrt{42}}$	$\frac{-1}{\sqrt{42}}$
111001	$\frac{-5}{\sqrt{42}}$	$\frac{-3}{\sqrt{42}}$
111010	$\frac{-5}{\sqrt{42}}$	$\frac{-5}{\sqrt{42}}$
111011	$\frac{-5}{\sqrt{42}}$	$\frac{-7}{\sqrt{42}}$
111100	$\frac{-7}{\sqrt{42}}$	$\frac{-1}{\sqrt{42}}$
111101	$\frac{-7}{\sqrt{42}}$	$\frac{3}{\sqrt{42}}$
111110	$\frac{-7}{\sqrt{42}}$	$\frac{-5}{\sqrt{42}}$
111111	$\frac{-7}{\sqrt{42}}$	$\frac{-7}{\sqrt{42}}$

Appendix B: Implement SDSOC project manually steps

The required steps to implement the SDSOC project manually are illustrated as follows:

1. Open SDSoC tool. The Workspace Launcher window will appear.
2. Select “**File**” → “**New**” → “**SDSoc Project**”. New Project GUI window will appear as shown in figure B.1. Enter project name, for example “**turboX**”, select “**ZC702**” as a platform (depending on the evaluation kit used), select “**Standrad**”**,** and click “**Next**”**.**



SDSoC Project

Create SDSoC project.

Project name: turbo5

☑ Use default location

Location: /home/mohammed/LTE_TEST_TEMP/ttt/turbo5

Choose file system: default

Target

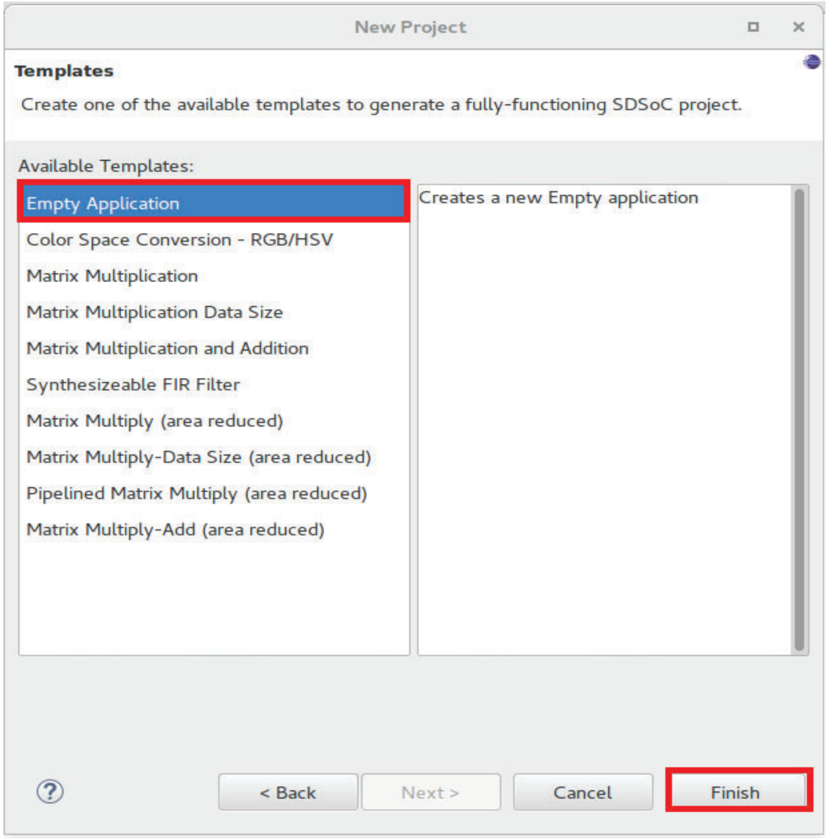
Platform: zc702

Platform Summary: Basic platform targeting the ZC702 board, which includes 1GB of DDR3, 16MB Quad-SPI Flash and an SDIO card interface. More information at http://www.xilinx.com/products/boards-and-kits/EK-Z7-ZC702-G.htm

OS: Standalone

Figure B.1: Project naming and platform selection and OS selection

3. Select “**Empty Application**” to create a new empty project and then Click “**Finish**” as shown in figure B.2.



Create one of the available templates to generate a fully-functioning SDSoC project.

Available Templates:
Empty Application
Color Space Conversion - RGB/HSV
Matrix Multiplication
Matrix Multiplication Data Size
Matrix Multiplication and Addition
Synthesizeable FIR Filter
Matrix Multiply (area reduced)
Matrix Multiply-Data Size (area reduced)
Pipelined Matrix Multiply (area reduced)
Matrix Multiply-Add (area reduced)

Creates a new Empty application

< Back
Next >
Cancel
Finish

Figure B.2: Empty project Selection

4. A new directory is generated at the location of project path and Project window will appear as shown in figure B.3. The new directory name is same as project name typed in step 1. Copy the C/C++ source files into the following directory inside project "<project path>" → "**turboX**" → "**src**"

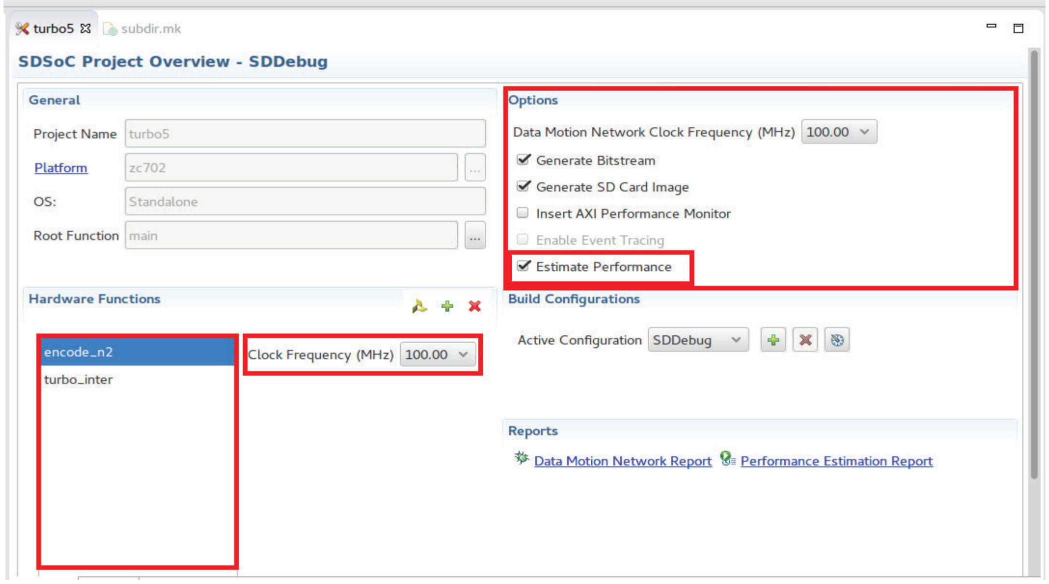
The "Project Explorer" window located in the left side, under "turboX" project directory, the "src" directory contains the source files which were copied from in the previous steps.

In the right-side pane, in "Option" window select generate bitstream, generate SD CARD image and estimate performance. In Hardware function, click in plus icon and select the required functions to be implemented as a hardware acceleration function. Also select the operating clock frequency for hardware functions.

Note that: -

a) Estimate performance mode is used to get the estimated cycles for hardware acceleration and to get estimated values for resource utilization.

b) The hardware functions are selected depending on the selected function.



SDSoC Project Overview - SDDebug

General

Project Name: turbo5

Platform: zc702

OS: Standalone

Root Function: main

Options

Data Motion Network Clock Frequency (MHz): 100.00

☑ Generate Bitstream

☑ Generate SD Card Image

☐ Insert AXI Performance Monitor

☐ Enable Event Tracing

☑ Estimate Performance

Hardware Functions

encode_n2

turbo_inter

Clock Frequency (MHz): 100.00

Build Configurations

Active Configuration: SDDebug

Reports

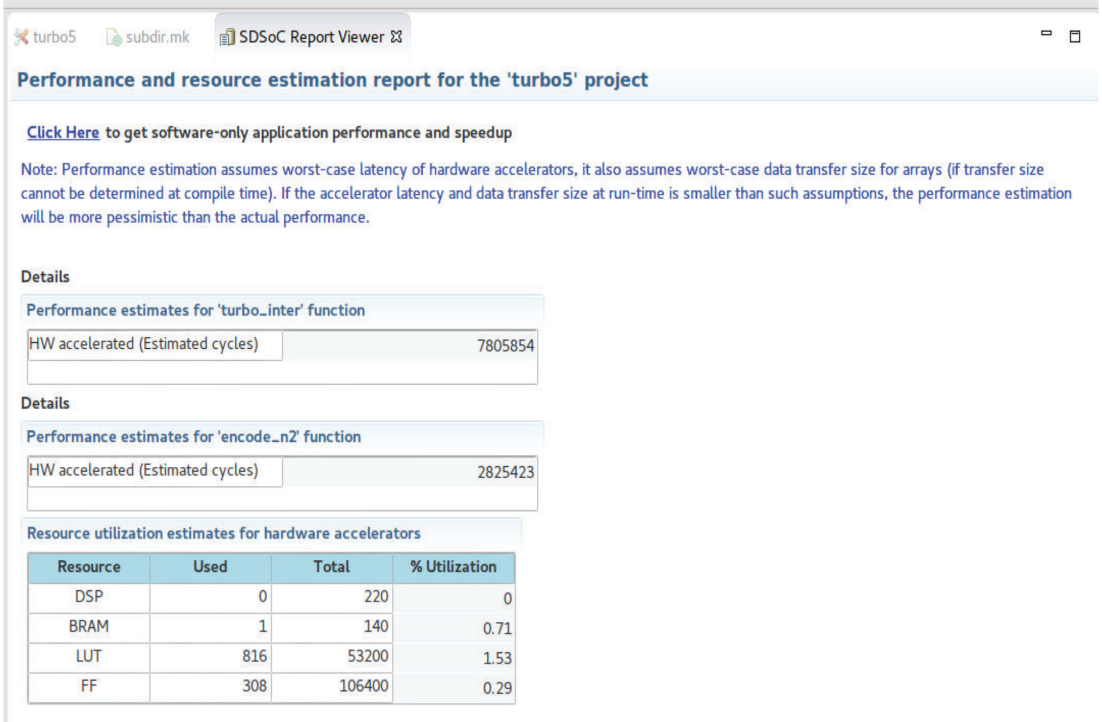
Data Motion Network Report

Performance Estimation Report

Figure B.3: project window in estimated performance mode

5. From "**Project**" → select "**Build ALL**", the process of building project will start.

6. After finish building project, the estimated performance results appear as shown in figure B4. The output results of the estimated performance mode generated by the SDSoC tool are the estimated hardware acceleration and the resource utilization for hardware acceleration function.



Performance and resource estimation report for the 'turbo5' project

Click Here to get software-only application performance and speedup

Note: Performance estimation assumes worst-case latency of hardware accelerators, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the accelerator latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Details

Performance estimates for 'turbo_inter' function

HW accelerated (Estimated cycles)	7805854

Details

Performance estimates for 'encode_n2' function

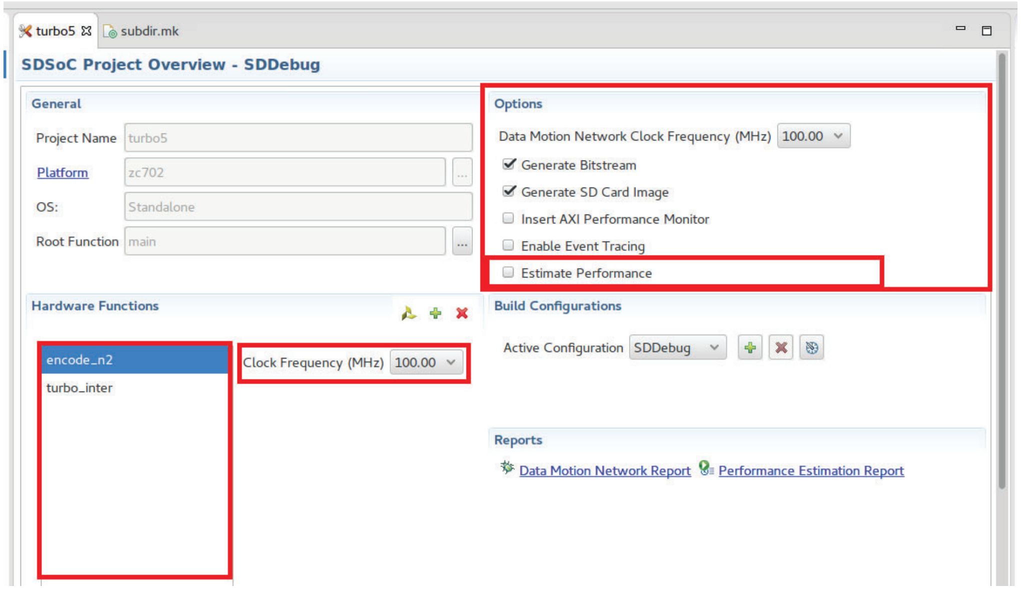
HW accelerated (Estimated cycles)	2825423

Resource utilization estimates for hardware accelerators

Resource	Used	Total	% Utilization
DSP	0	220	0
BRAM	1	140	0.71
LUT	816	53200	1.53
FF	308	106400	0.29

Figure B.4: Estimated performance results

7. To get the complete performance metrics for SDSoC implementation, repeat from step 1 to step 5, but in step 5, then de-select the “**Estimated Performance**” checkpoint shown in figure B.5 then, from “**Project**” → select “**Build ALL**”, the process of building project starts as shown in figure B.4.



SDSoC Project Overview - SDDebug

General

Project Name: turbo5
Platform: zc702
OS: Standalone
Root Function: main

Options

Data Motion Network Clock Frequency (MHz): 100.00
☑ Generate Bitstream
☑ Generate SD Card Image
☐ Insert AXI Performance Monitor
☐ Enable Event Tracing
☐ Estimate Performance

Hardware Functions

encode_n2
turbo_inter

Clock Frequency (MHz): 100.00

Build Configurations

Active Configuration: SDDebug

Reports

Figure B.5: Window project in non-estimated performance mode

8. After finish building the project, “**SDDebug**” directory is created inside the project path. The SDSoC directory structure inside the project path is as shown in figurer B.6.

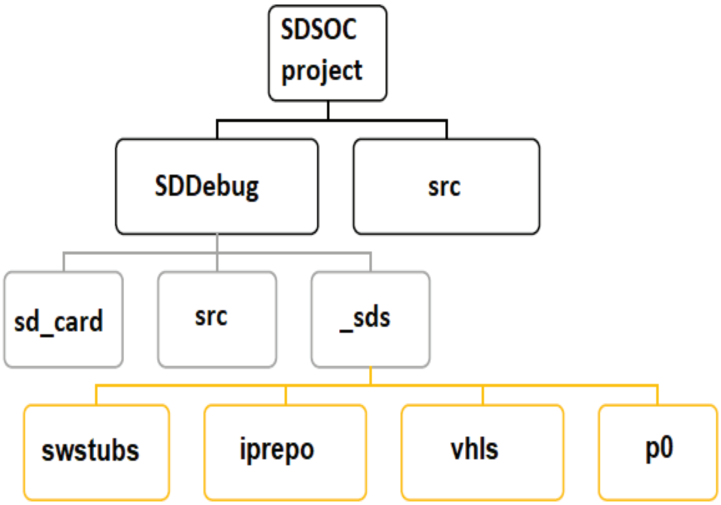


Figure B.6: SDOSC directory structure

The content of each directory is the following:

- The “**swstubs**” directory contains source file to handle data motion and source file used to communicate between different hardware acceleration block.
- The “**iprepo**” directory is generated by Vivado HLS and contain a sub-directory for each hardware function.
- The “**vhls**” directory is generated by Vivado HLS and contains the complete HLS implementation for each function.
- The “**p0**” directory consists of “ipi” sub-directory that contains The Vivado project (synthesis and implementation and results) and the generated bit file. The project file located inside the “ipi” directory and has an extension of xpr.

Appendix C: SDSOC shell script

The required steps to implement project using SDSoC shell script are shown as follows: -

1) Create directory "**lte_tx**" which is a top project path directory. Copy the "**src**" directory, the "**info.txt**" and "**SDSOC.sh**" to the "**lte_tx**" directory.

2) The "**src**" directory contains all the C/C++ source files to implement the LTE PDSCH transmitter/receiver using SDSoC tool. The "**SDSOC.sh**" is a developed automated script file used to execute SDSOC tool multiple times to generate all required solution sets.

   The "**info.txt**" is file describes the block function names that implemented as a hardware acceleration block. For example, the following list shows the "**info.txt**" of LTE PDSCH transmitter.

```
RE_mapper_dl_siso:lte_RE_mapper_dl_siso.c
modulator:lte_modulator.c
scrambler:lte_scrambler.c
rate_matching:lte_ratematching.c
interleaver:lte_interleaver.c
tx_turbo:tx_turbo.c
tx_seg_turboenc:lte_segment.c
tx_crc:tx_crc.c
```

3) Open a terminal window and execute the script using "**./SDSOC.sh**" command.

4) Select either C project of C++ project (the current LTE project is C project)

5) Select type of Xilinx evaluation board either a default ZC706 evaluation board or one of the following evaluation board shown in table C.1.

Table C.1: Supported SDSoC evaluation board

ID	Type of evaluation board
1	ZC702
2	ZC706
3	Zed
4	Zybo
5	ZCu102_es1
6	ZCu102_es2

6) Select the operating system of platform either the default standalone operating system or one of the operating system supported by sdsoc tool shown in table C.2.

Table C.2: Supported SDSoC operating system

ID	Type operating system
1	Linux
2	Standalone

Select the clock frequency used for synthesis and implementation hardware accelerated function. The table C.3 shows supported frequencies by the SDSOC tool.

Table C.3: synthesis frequency supported by SDSoC tool

ID	HLS synthesis frequency (MHz)
0	166.666672
1	142.857132
2	100.000000
3	200.000000

7) Select the data motion operating frequency, the frequency depend in selection of evaluation kit as shown in table C.4.

Table C.4: Data motion operating frequency

Platform	ID	data motion operating frequency
ZC702	0	166
	1	142
	2	100
	3	200
ZC706	0	166
	1	142
	2	100
	3	200
Zed	0	166
	1	142
	2	100
	3	200
Zybo	0	25
	1	100
	2	125
	3	50
ZCu102_e1	0	100
	1	150
	2	200
	3	300
ZCu102_e2	0	100
	1	150
	2	200
	3	300

8) The script generates all solution sets. Also, the script generates "design_space.rpt" file that contains the performance metrics results

List C.1: SDSoC.sh

```bash
#!/bin/bash

#Functions definition
create_makefile() {
	echo 'APPSOURCES = ' $ls*.$ext>Makefile
	echo 'EXECUTABLE = out.elf'>>Makefile
	echo 'CC = '$CC' ' $SDSFLAGS>>Makefile
	echo 'CFLAGS = -O3 -c'>>Makefile
	echo 'CFLAGS += -MMD -MP -MF"$(@:%.o=%.d)"'>>Makefile
	echo 'LFLAGS = -O3 -lm'>>Makefile
	echo 'OBJECTS := $(APPSOURCES:.'$ext'=.o)'>>Makefile
	echo 'DEPS := $(OBJECTS:.o=.d)'>>Makefile
	echo '.PHONY: all clean ultraclean'>>Makefile
	echo 'all: ${EXECUTABLE}'>>Makefile
	echo '${EXECUTABLE}: ${OBJECTS}'>>Makefile
	echo $'\t${CC} ${LFLAGS} $^ -o $@'>>Makefile
	echo '-include ${DEPS}'>>Makefile
	echo '%.o: %.'$ext>>Makefile
	echo $'\t${CC} ${CFLAGS} $^ -o $@'>>Makefile
	echo 'clean:'>>Makefile
	echo $'\t${RM} ${EXECUTABLE} ${OBJECTS} *.d'>>Makefile
	echo 'ultraclean: clean'>>Makefile
	echo $'\t${RM} ${EXECUTABLE}.bit'>>Makefile
	echo $'\t${RM} -rf _sds sd_card'>>Makefile
return  0
}

#Ask if C or C++ project
#Makefile format changes according project type
echo $'C or C++ project?\n1)C project\n2)C++ project?'
read choice
	if [ $choice = 1 ]
	then
		ext='c'
		CC='sdscc'
	elif [ $choice = 2 ]
	then
		ext='cpp'
		CC='sds++'
	fi
echo
"##################################################################"
```

```
#Choose OS
echo $'Default operating system is standalone\nchange OS ? [y/n]'
read choice
      if [ $choice = y ]
      then
            echo $'Available operating systems are:
linux,freertos,standalone\nchoosen OS: '
            read choice
            OS=$choice
      elif [ $choice = n ]
      then
            OS='standalone'
      fi
echo
"################################################################"


#choose HW synthesis clock frequency
echo $'Choose HW synthesis clock\navaliable CLK_IDs are
0,1,2,3\nclock frequency values differ based on platform\nfor more
information check Xilinx documents'
echo 'Choosen CLK_ID: '
read choice
CLK_ID=$choice
echo
"#################################################################"

#choose data network clock frequency
echo $'Choose data network clock\navaliable CLK_IDs are
0,1,2,3\ndata network clock frequency values differ based on
platform\nfor more information check Xilinx documents'
echo 'Choosen DMCCLK_ID: '
read choice
DMCCLK_ID=$choice
echo
"#################################################################"

#Create different solutions
#list C/C++ functions which will be compiled in both modes (HW and
SW) in array
#get number of functions from number of lines in file
function_count=$(cat info.txt |wc -l)
i=0
while [[ $i -lt $function_count ]]
do
      func[ $i ]=$(head -"$(echo "$i+1"|bc)" info.txt|tail -1 |cut
-d ':' -f 1)
      ((i++))
done
```

```
#then compute the number of C/C++ funcions in current directory and
the number of HW/SW combinations and loop on combinations
combination_num="$(echo  2^$function_count |bc)"
i=0
while [ $i -lt $combination_num ]
do
      #make directory for solution and create Makefile
      mkdir ./solution_$i
      #convert solution number to binary to determine how each
C/C++ is compiled (HW/SW)
      tmp="$(echo "obase=2;$i"|bc)"
      function_mode="$(echo $(printf "%0"$(echo $function_count)"d"
$tmp))"
            #loop on functions to generate HardWare options field
in the sdscc/sds++ command synopsis
            j=0
            HW_options_str=" "
            while [ $j -lt $function_count ]
            do
                  #HW_options are passed to sdscc/sds++ compiler in
the Makefile
                  if [[ ${function_mode:$j:1} = 1 ]];
                  then
                        file_name=$(cat info.txt|grep -w
${func[$j]} |cut -d':' -f 2)
                        HW_options_str=$HW_options_str' -sds-hw
'${func[$j]}' '${file_name}' -clkid '$CLK_ID' -sds-end '
                  fi

            (( j++ ))
            done

      #Two design flows will be followed, performance estimation
flow and traditional flow
      #performance estimation flow is used for latency estimation
and speed up
      #traditional flow is used to create SDCard files and get
area,power info.
      if [ $i -lt 10 ]
      then
            line='################## solution '$i'
######################'
      elif [ $i -lt 100 ]
      then
            line='################## solution '$i'
#####################'
      else
            line='################## solution '$i'
####################'
      fi

      echo '####################################################'
      echo $line
      echo '####################################################'
      #************************************************************
      #2)Traditional flow
      mkdir ./solution_$i/reg_flow
      cp -r src/* ./solution_$i/reg_flow
      cd ./solution_$i/reg_flow && touch Makefile
      SDSFLAGS=' -sds-pf '$platform' '$HW_options_str' -target-os
'$OS' -dmclkid '$DMCCLK_ID
      create_makefile
      make -f Makefile

#Generate design space exploration report
touch design_space.rpt
echo 'This file is generated by SDSoC script'>design_space.rpt
echo 'Author: Mahmoud M.Kishky'>>design_space.rpt
echo "+-------+-------------------+--------------+--------------+---
----------+">>design_space.rpt
printf "| %5s | %16s | %12s | %12s | %11s | \n"   'Sol#'   'Slice
Logic'     'DSPs'     'Latency'     'Power(W)' >>design_space.rpt
echo "+-------+-------------------+--------------+--------------+---
----------+">>design_space.rpt


i=1
while [ $i -lt $combination_num ]
do
      #power Calculation
      total_power="$(cat
./solution_$i/reg_flow/_sds/p0/ipi/*.runs/impl_1/*_power_routed.rpt
|grep Total\ On-Chip\ Power -w |cut -d'|' -f 3  )"
      pow[$i]="$(echo "($total_power)"|bc)"

      #Area Calculation
      slice_luts[$i]="$(cat
./solution_$i/reg_flow/_sds/p0/ipi/*.runs/impl_1/*_power_routed.rpt
|grep Slice\ Logic -w |cut -d'|' -f 4  )"
      DSP[$i]="$(cat
./solution_$i/reg_flow/_sds/p0/ipi/*.runs/impl_1/*_utilization_plac
ed.rpt|grep DSPs -w |cut -d'|' -f 3  )"

      #write in perfromance report file
      printf "| %5s | %16s | %12s | %12s | %11s | \n" $i
${slice_luts[$i]}  ${DSP[$i]}  ${latency[$i]}
${pow[$i]}>>design_space.rpt

      #write in perfromance report file
      printf "| %5s | %16s | %12s | %12s | %11s | \n" $i
${slice_luts[$i]}  ${DSP[$i]}  ${latency[$i]}
${pow[$i]}>>design_space.rpt
((i++))
done
echo "+-------+-------------------+--------------+--------------+---
----------+">>design_space.rpt
```

List C.2: cost_calc.sh

```
#!/bin/bash
#Author: Mahmoud M.Kishky

touch cost.rpt
echo "+-------+-------------------------+">>cost.rpt
printf "| %5s | %23s |\n"   'Sol#'     'cost'>>cost.rpt
echo "+-------+-------------------------+">>cost.rpt
sol_count=$(cat valid_solution.rpt |wc -l)

area_tar=$(cat cons.txt|grep area_target|cut -d ':' -f 2)
pow_tar=$(cat cons.txt|grep power_target|cut -d ':' -f 2)
lat_tar=$(cat cons.txt|grep latency_target|cut -d ':' -f 2)
aw=$(cat cons.txt|grep area_weight|cut -d ':' -f 2)
pw=$(cat cons.txt|grep power_weight|cut -d ':' -f 2)
lw=$(cat cons.txt|grep latency_weight|cut -d ':' -f 2)

i=0
while [[ $i -lt $sol_count ]]
do

      sol_num=$(head -"$(echo "$i+1"|bc)" valid_solution.rpt|tail -
1 |cut -d ':' -f 1)
      power="$(cat
./validSol/solution_$sol_num/_sds/p0/ipi/*.runs/impl_1/*_power_rout
ed.rpt|grep Total\ On-Chip\ Power -w |cut -d'|' -f 3  )"
      area="$(cat
./validSol/solution_$sol_num/_sds/p0/ipi/*.runs/impl_1/*_power_rout
ed.rpt|grep Slice\ Logic -w |cut -d'|' -f 4  )"
      latency="$(cat
./perfEst/solution_$sol_num/_sds/est/console_out.log |grep
Estimated\ hardware\ latency -w |cut -d'=' -f 2  )"

      t1=$(echo "($area-$area_tar)/$area_tar" | bc -l)
      t2=$(echo "($power-$pow_tar)/$pow_tar" | bc -l)
      t3=$(echo "($latency-$lat_tar)/$lat_tar" | bc -l)
      cost[$i]=$(echo "$aw*$t1+$pw*$t2+$lw*$t3" | bc -l)
      #write in perfromance report file
      printf "| %5s | %23s |\n" $sol_num ${cost[$i]} >>cost.rpt
((i++))
done
echo "+-------+-------------------------+">>cost.rpt

#sort according to cost
#print least cost solution/s

IFS=$'\n'
least_cost="$(echo "${cost[*]}" | sort -n |head -1)"
echo "least cost solution/s:">>cost.rpt
cat cost.rpt |grep $least_cost -w|cut -d'|' -f 2 |tr '\n'
','>>cost.rpt
echo $'\n'>>cost.rpt
```

الملخص

صناعة الأنظمة على رقاقة (SoC) تخلق الكثير من التحديات لمطورين مثل هذه الأنظمة.
تتدرج التحديات من التعقيد الوظيفي و التصميم و المجهود اللازم من أجل التأكد من وظيفة الرقاقة و التصميم على قيود مقاييس الأداء مثل الطاقة. بالإضافة إلى ذلك, التجانس الصعب بين الأدوات و المكونات مما يؤدي إلى الإحتياج إلى دورة طويلة و معقدة من أجل التنفيذ.

طورت شركة (Xilinx) الأداة المسماة (SDSoC) , وهي بيئة متكاملة متخصصة لتصميم الأنظمة على رقاقة (SoC) على منصة متكاملة مبنية على منصة متكاملة بين المعالج و مصفوفة البوابات القابلة للبرمجة. تقدم هذه الأداة آلية جديدة لبناء الأنظمة على رقاقة على منصة متكاملة مبنية على منصة تتكون من المعالج و مصفوفة البوابات القابلة للبرمجة بالإضافة إلى أنها تدمج أدوات متعددة و هذا يجعل عملية التصميم أسرع و أكثر مرونة.

في هذه الأطروحة , تم توضيح طريقة عمل الأداة المسماة (SDSoC) بالإضافة إلى ذلك تم تصميم طريقة عمل مقترحة لإدخال تقنية جديدة من أجل تشغيل الأداة لتصميم النظام على رقاقة (SoC) وفق قيود مسبقة على مقاييس الأداء مثل الطاقة و المساحة.
تم تصميم نظام الأرسال و الإستقبال ل شبكة الجيل الرابع (LTE) كتطبيق مباشر على طريقة العمل الجديدة التي تم تصميمها بالإضافة إلى اختيار المنصة التي تحقق أفضل قيود مقاييس الأداء.

مهندس: محمد أحمد عبدالحميد العدوي

تاريخ الميلاد: 1988\09\21

الجنسية: مصري

تاريخ التسجيل: 2012\10\01

تاريخ المنح: 2019\....\.....

القسم: هندسة الإلكترونيات و الإتصالات الكهربية

الدرجة: ماجستير العلوم

المشرفون:

ا.د. أحمد حسين محمد خليل

د. حسن مصطفى حسن مصطفى

الممتحنون:

أ.د. أحمد حسين محمد خليل مشرف رئيسي
أستاذ في قسم الإلكترونيات بكلية الهندسة – جامعة القاهرة

د. حسن مصطفى حسن مصطفى مشرف
استاذ مساعد في قسم الالكترونيات بكلية الهندسة – جامعة القاهرة

أ.د. أمين محمد نصار الممتحن الداخلي
أستاذ في قسم الإلكترونيات بكلية الهندسة – جامعة القاهرة

أ.د. عمرو طلعت عبدالحميد الممتحن الخارجي
أستاذ مساعد بقسمى الإلكترونيات و الشبكات – الجامعة الألمانية بالقاهرة

عنوان الرسالة:

تصميم سيناريو ذاتي يعتمد على متغيرات مقاييس الأداء من أجل تمثيل فعال ل LTE PDSCH باستخدام برنامج SDSoC

الكلمات الدالة:

SDSoC; Xilinx; SoC; LTE; PDSCH; FPGA

ملخص الرسالة:

طورت شركة (Xilinx) الأداة المسماة (SDSoC) وهي بيئة متكاملة متخصصة لتصميم الأنظمة على رقاقة (SoC) على منصة متكاملة مبنية على منصة متكاملة بين المعالج و مصفوفة البوابات القابلة للبرمجة .تقدم هذه الأداة آلية جديدة لبناء الأنظمة على رقاقة على منصة متكاملة مبنية على منصة تتكون من المعالج و مصفوفة البوابات القابلة للبرمجة .في هذه الأطروحة , تم توضيح طريقة عمل الأداة المسماة (SDSoC) بالإضافة إلى ذلك تم تصميم طريقة عمل مقترحة لإدخال تقنية جديدة من أجل تشغيل الأداة لتصميم النظام على رقاقة (SoC) وفق قيود مسبقة على مقاييس الأداء مثل الطاقة و المساحة. تم تصميم نظام الأرسال و الإستقبال ل شبكة الجيل الرابع كتطبيق مباشر على طريقة العمل الجديدة التي تم تصميمها بالإضافة إلى أختيار المنصة التي تحقق أفضل قيود مقاييس الأداء.

تصميم سيناريو ذاتي يعتمد على متغيرات مقاييس الأداء من أجل تمثيل فعال ل LTE PDSCH باستخدام برنامج SDSoC

اعداد

محمد أحمد عبدالحميد العدوي

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة
ماجستير العلوم
في
هندسة الإلكترونيات و الاتصالات الكهربية

يعتمد من لجنة الممتحنين:

الاستاذ الدكتور: أحمد حسين محمد خليل مشرف رئيسى
أستاذ في قسم الإلكترونيات بكلية الهندسة – جامعة القاهرة

الدكتور: حسن مصطفى حسن مصطفى مشرف
استاذ مساعد في قسم الالكترونيات بكلية الهندسة – جامعة القاهرة

الاستاذ الدكتور: أمين محمد نصار الممتحن الداخلي
أستاذ في قسم الإلكترونيات بكلية الهندسة – جامعة القاهرة

الاستاذ الدكتور: عمرو طلعت عبدالحميد الممتحن الخارجي
أستاذ مساعد بقسمى الإلكترونيات و الشبكات – الجامعة الألمانية بالقاهرة

كلية الهندسة - جامعة القاهرة
الجيزة - جمهورية مصر العربية
2019

تصميم سيناريو ذاتي يعتمد على متغيرات مقاييس الأداء من أجل تمثيل فعال ل LTE PDSCH باستخدام برنامج SDSoC

اعداد

محمد أحمد عبدالحميد العدوي

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة
ماجستير العلوم
في
هندسة الإلكترونيات و الاتصالات الكهربية

تحت اشراف

أ.د. أحمد حسين محمد خليل
استاذ في قسم الالكترونيات
والاتصالات الكهربية
كلية الهندسة - جامعة القاهرة

د. حسن مصطفى حسن مصطفى
استاذ مساعد في قسم الالكترونيات
والاتصالات الكهربية
كلية الهندسة - جامعة القاهرة

كليـة الهندسـة - جامعـة القاهـرة
الجيـزة - جمهوريـة مصـر العربيـة
2019

تصميم سيناريو ذاتي يعتمد على متغيرات مقاييس الأداء من أجل تمثيل فعال ل LTE PDSCH باستخدام برنامج SDSoC

اعداد

محمد أحمد عبدالحميد العدوي

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة
ماجستير العلوم
في
هندسة الإلكترونيات و الاتصالات الكهربية

كلية الهندسة - جامعة القاهرة
الجيزة - جمهورية مصر العربية
2019