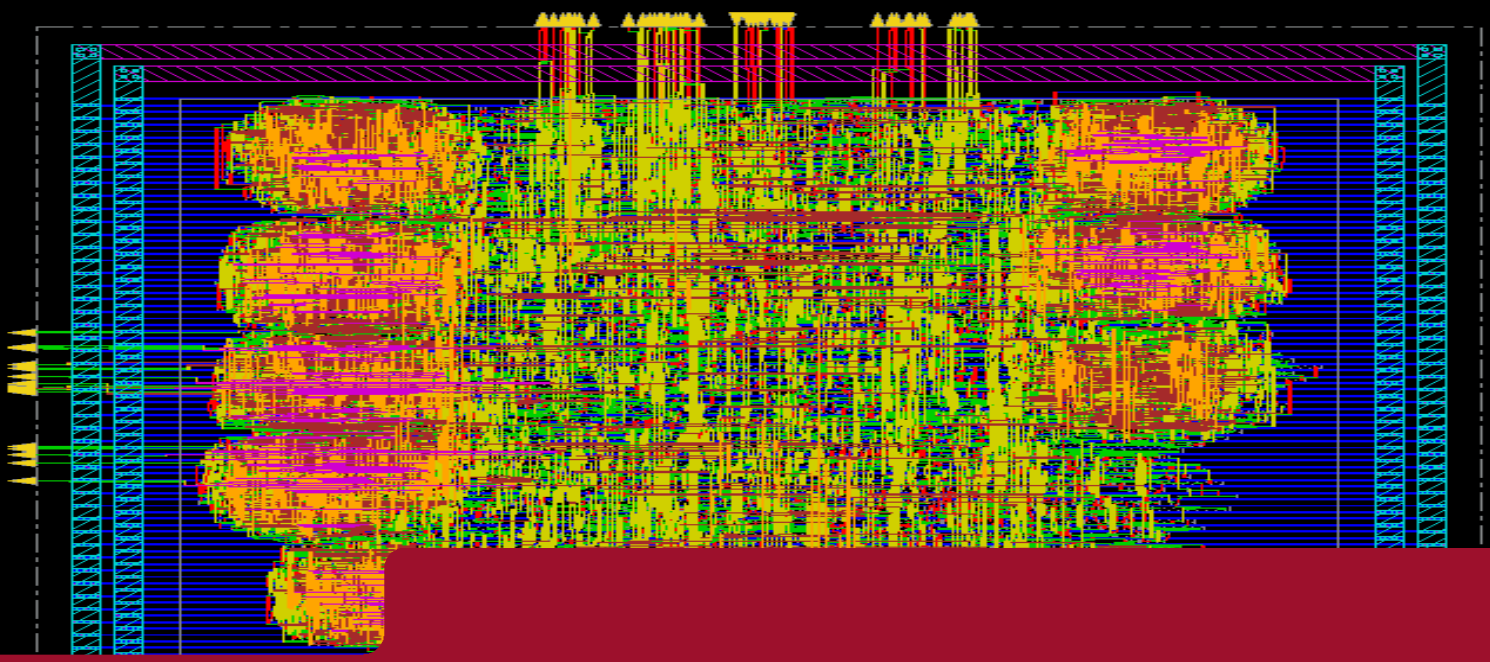
DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, *SWEDEN 2019*

Implementation of 3GPP LTE QPP Interleaver for SiLago

SPANDAN DEY



KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Author

Spandan Dey <spandan@kth.se>
Electronics and Embedded Systems
KTH Royal Institute of Technology

Place for Project

Stockholm, Sweden
Sandviken, Sweden

Examiner

Prof. Dr. Ahmed Hemani <hemani@kth.se>
Electronics and Embedded Systems
KTH Royal Institute of Technology

Supervisor

Dimitrios Stathis <stathis@kth.se>
Electronics and Embedded Systems
KTH Royal Institute of Technology

Acknowledgement

My sincere thanks to Prof. Dr. Ahmed Hemani for giving me this opportunity to work on such an interesting topic.

I would like to specially thank Dimitrios Stathis. This work would have never been possible without his support and guidance.

A special credit goes to Konstantinos Sotiropoulos for being a friend-philosopher-guide.

I would like to thank Sharan Yagneshwar for encouragement and some interesting discussion sessions providing valuable suggestions during the writing of this text. I strongly recommend to read his work titled "Performance Optimization of Signal Processing Algorithms for SIMD Architectures."

Last but not the least, a very special thanks to my wife Shraddha, thank you for providing comfort and motivation when needed the most.

Sandviken, Date: September 29, 2019

Spandan Dey

Abstract

Modern wireless communication systems have seen an increased usage of various channel coding techniques to facilitate improved throughput and latency. Interleavers form an integral part of these coding techniques and play a critical role by making the communication more robust and resilient to noise and other interference. The ever increasing need for higher throughputs and lower latencies has made designers to pursue a more parallel design approach giving rise to parallel adaptations of these encoding/decoding techniques.

A bulk of the modern telecommunication occurs over Wireless Wide Area Network (WWAN), commonly referred to as cellular networks. The 3rd Generation Partnership Project (3GPP), Long Term Evolution (LTE) develops and specifies the standards that are used in cellular communication. Their current most widely used "4G" standard employs Turbo coding techniques and a Quadratic Permutation Polynomial (QPP) interleaver.

Silicon Large Grain Object or SiLago is a Coarse Grain Reconfigurable Fabric facilitating a modular approach towards electronics hardware development. The concept is similar to LEGO bricks, that is to have a library of hardened blocks (similar to Lego bricks) out of which systems of various types and functionalities can be built.

This thesis investigates the state-of-the-art parallel interleavers and parallel interleaving techniques available for the 3GPP LTE QPP interleavers, and implements two interleaver designs, one for Radix 2 and another for Radix 4 decoding techniques. A physical synthesis is carried out in 28nm technology and the results in terms of power and area are reported.

Keywords

Quadratic Permutation Polynomial, QPP, interleaver, SiLago, Address Generation Unit, AGU, 3GPP, LTE, turbo codes, parallel

Referat

Moderna trådlösa kommunikationssystem har sett ökad användning av olika kanaler kodningstekniker för att underlätta förbättrad genomströmning och latens. Interleavers utgör en integrerad del av dessa kodningstekniker och spelar en viktig roll genom att göra kommunikation mer robust och fjädrande för brus och andra störningar. Det ökande behovet av högre genomströmningar och lägre latenser har gjort konstruktörer att driva en mer parallell design tillvägagångssätt som ger upphov till parallella anpassningar av dessa kodningstekniker.

En stor del av modern telekommunikation är via Wireless Wide Area Network (WWAN), vanligen kallad mobilnät. Det Third Generation Partnership Project (3GPP), Long Term Evolution (LTE) utvecklar och specificerar de standarder som används i mobil kommunikation. Deras nuvarande mest använda "4G" standard använder Turbo-kodning tekniker och en Quadratic Permutation Polynomial (QPP) interleaver.

Silicon Large Grain Object eller SiLago är ett grovt kornkonfigurerbart tygstöd ett modulärt tillvägagångssätt för elektronikutveckling. Konceptet är liknande LEGO-tegelstenar, det Ã¤r med ett library av härdade block (liknande Lego-tegelstenar), varav system av olika typer och funktioner kan byggas.

Denna avhandling undersöker de toppmoderna parallella interleaversna och parallellinterfolieringen tekniker som är tillgängliga för 3GPP LTE QPP interleavers, och implementerar tvåinterleavers mönster, en för Radix 2 och en annan för Radix 4-avkodningstekniker. En fysisk syntes utförs i 28nm-teknik och resultaten i kraft och area rapporteras.

Nyckelord

Quadratic Permutation Polynomial, QPP, interleaver, SiLago, Address Generation Unit, AGU, 3GPP, LTE, turbo codes, parallel

Dedicated to my parents...

Contents

Acknowledgement v

Abstract vii

Referat ix

Table of Contents xiii

List of Figures xv

List of Tables xvii

List of Acronyms and Abbreviations xix

1. Introduction 1
- 1.1. Overview . . . 2
- 1.2. Problem Definition . . . 2
- 1.3. Purpose . . . 2
- 1.4. Goals . . . 2
- 1.5. Research Methodology . . . 3
- 1.6. Delimitations . . . 3
- 1.7. Structure of this thesis . . . 3

2. Methodology 5
- 2.1. Research Process . . . 5
- 2.2. Research Design . . . 6
- 2.3. Validity, Reliability, and Generalizability . . . 6
  - 2.3.1. Validity . . . 6
  - 2.3.2. Reliability . . . 7
  - 2.3.3. Generalizability . . . 7

3. Literature Review and Background 9
- 3.1. Information Theory and Coding Theory . . . 9
  - 3.1.1. Forward Error Correction in Wireless Communication . . . 10
  - 3.1.2. Types of ECC . . . 10
  - 3.1.3. Turbo Codes . . . 10
  - 3.1.4. Burst Error and Interleaving . . . 11
- 3.2. Interleavers . . . 12
  - 3.2.1. 3GPP LTE QPP Interleaver . . . 12
  - 3.2.2. Hardware Implementation of 3GPP LTE QPP Interleaver . . . 14
  - 3.2.3. Multimode Interleavers . . . 16
- 3.3. Silicon Large Grain Objects (SiLago) . . . 18
  - 3.3.1. SiLago Architecture . . . 18
  - 3.3.2. SiLago Design Flow . . . 20

3.4. The Research Gap . . . 21

4. Implementation **23**
4.1. Implementation Process . . . 23
4.2. Software (SW) Tools . . . 24
4.3. Implementation of parallel 3GPP LTE QPP Interleavers . . . 26
4.3.1. Circuit Design and Simulation . . . 26
4.3.2. Logic Synthesis . . . 31
4.3.3. Physical Synthesis . . . 32
4.3.4. Post Layout Verification . . . 32
4.4. Summary of Workflow . . . 33
4.5. Validity and Reliability . . . 33

5. Results and Analysis **35**
5.1. Circuit Design and simulations . . . 35
5.1.1. Failure of AM units . . . 35
5.1.2. Failure to reuse $g(x)$ coefficient . . . 36
5.1.3. Questa Sim Logic Simulation . . . 37
5.2. Logic Synthesis . . . 38
5.3. Physical Synthesis . . . 39
5.4. Post Layout Verification . . . 40
5.5. Discussions . . . 41
5.5.1. Parallelization . . . 41
5.5.2. Memory Organization . . . 41
5.5.3. Partitioning in SiLago . . . 42
5.5.4. SiLago Advantage . . . 42

6. Conclusions **43**
6.1. Contributions . . . 43
6.2. Limitations . . . 43
6.3. Future Work . . . 44
6.4. Reflections . . . 44
6.5. Credits . . . 44

Appendices **45**

A. Schematics **45**
A. Circuit diagram of Individual Components . . . 45
B. Schematics Exported from Questa Sim . . . 47

B. Python Scripts **49**
A. Radix2 P8 Test Vector . . . 49
B. Radix4 P4 Test Vector . . . 51

C. TCL Scripts **53**
A. Design Vision . . . 53
B. Innovus . . . 54
C. Physical Design Flow in Innovus . . . 58

Bibliography **59**

Alphabetical Index **63**

List of Figures

2.1. Research process 5

3.1. Block diagram of digital communication system 9
3.2. Turbo Encoder 11
3.3. Turbo Decoder 11
3.4. Hardware for Serial 3GPP LTE QPP Interleaver 15
3.5. Parallel Turbo Decoder Architectures 16
3.6. Some interleaver standards 17
3.7. SiLago Platform Example 18
3.8. SiLago platform instance 19
3.9. SiLago Design Flow 20

4.1. Gajski-Kuhn Y-chart 24
4.2. First Iteration 3GPP LTE QPP Interleaver 26
4.3. Second Iteration 3GPP LTE QPP Interleaver 27
4.4. Third Iteration 3GPP LTE QPP Interleaver 28
4.5. Fourth Iteration 3GPP LTE QPP Interleaver 28
4.6. Radix-2 Parallel 3GPP LTE QPP Interleaver 30
4.7. Radix-4 Parallel 3GPP LTE QPP Interleaver 30
4.8. Thesis WorkFlow 33

5.1. Asghar's proposed design for 3GPP LTE QPP interleavers 36
5.2. Radix-2 Schematic exported from Questa Sim 37
5.3. Radix-4 Schematic exported from Questa Sim 38
5.4. Area distribution among different sub-components 39
5.5. Radix-2 Physical Layout 40
5.6. Radix-4 Physical Layout 40
5.7. Radix-4 Schematic exported from Questa Sim 40
5.8. Design Aspects and trade-offs 42
5.9. Unified Architecture 42

List of Tables

3.1. Turbo code internal interleaver parameters 13

4.1. Logic Synthesis Parameters 32

5.1. Logic Synthesis Results 38
5.2. Physical Synthesis Results 39

List of Acronyms and Abbreviations

3GPP	3rd Generation Partnership Project
AM	Add-Modulo
AM+	Add-Modulo-plus
ASIC	Application Specific Integrated Circuits
DRC	Design Rule Checking
DRRA	Dynamically Reconfigurable Resource Array
ECC	Error Correcting Code
EDA	Electronic Design Automations
FEC	Forward Error Correction
FPMAP	Fully Parallel MAP architecture
FSM	Finite State Machine
GCD	Greatest Common Divisor
HDL	Hardware Description Language
LDPC	Low-Density Parity-Check
LTE	Long-Term-Evolution
NOC	Network on Chip
P-MAP	Pipelined-MAP architecture
PMAP	Parallel MAP architecture
QPP	Quadratic Permutation Polynomial
RSC	Recursive Systematic Convolutional
RTL	Register-transfer level
SiLago	Silicon Large Grain Object
UXMAP	Unrolled XMAP architecture
VHDL	Very-high-speed-integrated-circuit Hardware Description Language
VLSI	Very-large-scale integration

Chapter 1.

1 Introduction

Human kind is undergoing a major transformation primarily fuelled by telecommunication. From Alexander Graham Bell and Charles Sumner Tainter's photophone[1] to the present deep space communication with the most distant man made object in space Voyager-1[2], humans have come a long way in wireless communication. Telecommunication has now become one of the big pillars holding modern society.

A major challenge of wireless communication is the wireless medium itself, which can be noisy. There exists techniques referred to as channel coding which helps mitigate the issue of the effects of noise on wireless communication. Channel coding ensures reliable transmission of information primarily by adding multiple copies of the information stream. In case of information loss during transmission, the original stream of information can be recreated from one or few of the redundant copies. This allows some degree of error correction capability at the receiver end.

With computation capabilities of semiconductors becoming more available, modern communication systems employs advanced coding schemes. There is also a growing trend for fast and voluminous communication in portable (battery operated, or energy harvesting) devices. This has also pushed the need for new coding techniques to be squeezed into semiconductor devices that are high performant as well as energy efficient. These constraints have resulted in highly complex encoding/decoding architectures and delicate designs.

SiLago[3] is a modular approach towards Very-large-scale integration (VLSI) design. Unlike standard cell based design approach of current VLSI design flow, SiLago aims to a more subsystem based design also referred as SiLago blocks. This would simplify system design as complex design blocks, for example the encoder/decoders discussed before, can be reused across multiple designs reducing design efforts significantly and also reduce probability of design errors.

SiLago achieves this reusability of design through a design approach similar to that of LEGO bricks, that is by making the designs of each SiLago block follow similar dimensions and maintain a common interface. This allows blocks to be abuttable, that is to be placed adjacent to each other.

The rest of the chapter is organized as follows. Section 1.1 gives an overview of the areas discussed in this thesis. The problem addressed in this thesis is discussed in Section 1.2. Section 1.3 justifies the relevance and need for this research. The expected deliverable from this work is explained in Section 1.4. There is a brief introduction to the research paradigm in Section 1.5. Section 1.6 states the boundaries that have been set for this research. A summary on how this thesis is organized can be found in Section 1.7.

1.1. Overview

Interleaving/De-interleaving is a key component in the encoding/decoding stages of modern wireless communication systems. Wireless communication systems use a variety of interleaving/de-interleaving techniques. Silicon Large Grain Object (SiLago) is a Coarse Grain Reconfigurable Fabric facilitating a modular approach towards electronics hardware development. It raises the abstraction of hardware development by replacing the concept of standard cell based design to a more higher abstraction to micro-architectural based design. Each of these micro-architectural blocks is referred to as a SiLago block. There exists a variety of such blocks each with different functionalities. Selecting from these varieties and joining them together, one can build systems of various types and functionalities.

At the time of writing this thesis, there does not exist an interleaver/de-interleaver block that efficiently implements a 3GPP LTE QPP interleaver/de-interleaver in SiLago. The main goal of this thesis is to extend the wireless encoding-decoding capabilities of SiLago by adding interleaving/de-interleaving block. This capability should be scalable and can be incorporated into SiLago.

1.2. Problem Definition

An effort was made for an implementable solution through a degree project by Alberto Castella [4] but the implementation was not suitable for SiLago. The implementation was based on the algorithm introduced by Asghar[5]. One of the main challenges in Alberto's thesis was the size of the QPP interleaver. The multi-mode interleaver implementation requires memory to store mathematical coefficients. This memory made the interleaver implementation too large to be able to fit into a few SiLago blocks. Another drawback arose due to the bit level addressing needed for some types of interleavers. Hence there is a need to find an alternative approach for interleaver implementation in SiLago.

1.3. Purpose

Interleavers form a critical part of wireless communication. This thesis will contribute by adding a key functionality which will facilitate development of a range of wireless communications within SiLago.

1.4. Goals

The goals of this thesis work (in order of priority) are:

1. SiLago implementation of 3GPP LTE QPP interleaver block
2. Show the usability of the implementation
3. A project report detailing the process of implementing the above.

1.5. Research Methodology

The research encompasses an existing theory and will contribute by solving a practical problem through a theoretical model. Hence, a constructive research approach will be employed for conducting this research. Lehtiranta *et al.* [6] argues that the main aim of a constructive research is to solve practical problems while producing academically appreciated theoretical contribution. The research process involves the following [6]:

1. selecting a practically relevant problem
2. obtaining a comprehensive understanding of the study area
3. designing one or more applicable solutions to the problem
4. demonstrating the solution's feasibility
5. linking the results back to the theory and demonstrating their practical contribution
6. examining the general usability of the results

1.6. Delimitations

The design space of VLSI implementation is both broad and diverse. The plethora of choices makes developing a generalized solution a challenging task which is beyond the scope of this thesis. Hence, certain limitations were made and listed as follows:

1. No particular preference was made regarding throughput and latency. Hence a more balanced solution is proposed.
2. Memory organization was also left out. To maintain high throughput and low latency, interleavers requires a close coupling with the encoder-decoders. Due to the absence of an existing solution in SiLago to suit this requirement, this was kept out of scope. Some suggestions are made at the end of this thesis.
3. Not much focus was given to the design constraints used in electronic design automation process. The synthesis steps was aligned with SiLago implementation without optimizations.

1.7. Structure of this thesis

The rest of the thesis is organized as follows:

- Chapter 2 describes the research methodology adopted for this research.
- Chapter 3 introduces the key concepts necessary to understand this thesis along with a detailed literature review of some of the notable works in relevant areas.
- Chapter 4 presents the implementation details.
- Chapter 5 discusses the results from the implementation.
- Chapter 6 concludes the work, presents some limitations, suggestion for future work, and contains a short reflection.

Chapter 2.
2 Methodology

Formal methods and techniques play an important role in motivating the end result, and the quality of research. The research encompasses an existing theory and contributes by solving a practical problem through a model implementation. Hence, a constructive research approach was employed for conducting this research.

This chapter is divided into three sections. Section 2.1 discusses the research process undertaken while conducting this research. Section 2.2 discuss the overall research setup serving as a guide towards the ultimate goal of delivering high quality artefacts. This is followed by argument on validity, reliability, and generalizability in Section 2.3.

2.1. Research Process

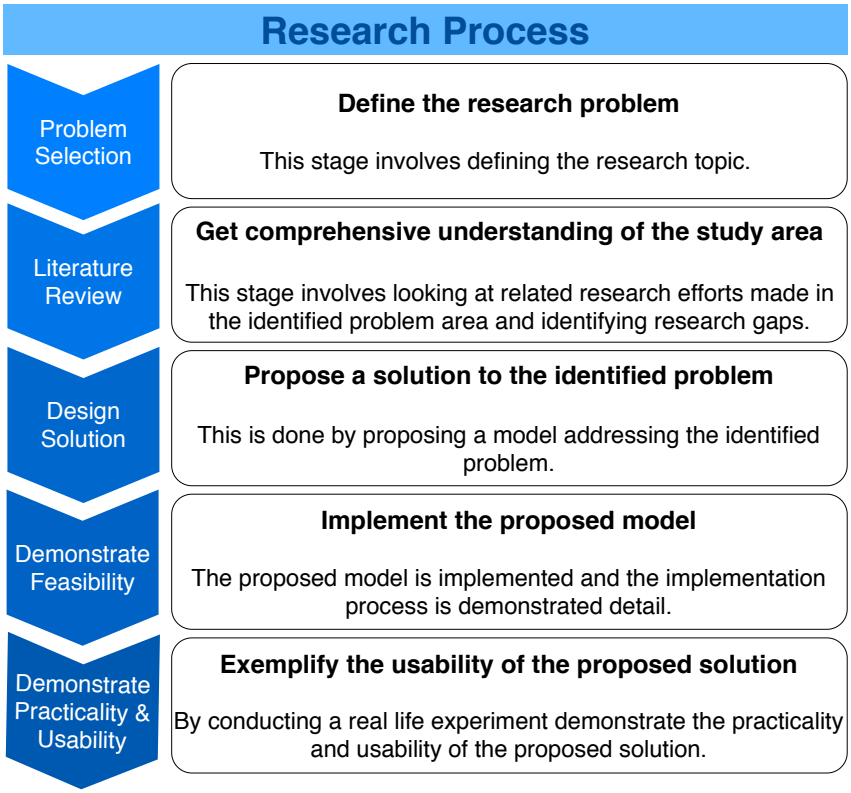


Figure 2.1.: Research process

Lehtiranta *et al.*[6] argues that the main aim of a constructive research is to solve practical problems while producing academically appreciated theoretical contribution. The research process proposed involves the following: (1) selecting a practically relevant problem; (2) obtaining a comprehensive understanding of the study area; (3) designing one or more applicable solutions to the problem; (4) demonstrating the solution's feasibility; (5) linking the results back to the theory and demonstrating their practical contribution; and (6) examining the general usability of the results [6]. Figure 2.1 shows the steps conducted in order to carry out this research.

2.2. Research Design

Selecting a relevant problem was based on qualitative methods, mostly through unstructured interviews with researchers. Apart from interviews, the author of this thesis also participated in discussions and tele-conferences.

A **comprehensive understanding of the research area** was carried out by reviewing researches addressing the identified problem along with a review of some of the approaches adopted by institutions to address the research question. Through this, the research gap was identified to motivate the research need.

Designing an applicable solution to the problem was done through empirical observations of existing research and proposing a new model which addresses the problem area.

The **feasibility of the proposed solution** was done by implementing the proposed model and through simulations. To make the implementation more widely recognizable, industry leading software applications were used.

Finally, the **usability and practicality of the proposed solution** was demonstrated with a practical implementation of the model and comparing it with other research results. The demonstration highlights the practicality and usability of the model as well as serves as a key input for further research.

2.3. Validity, Reliability, and Generalizability

Although this is constructive research, certain parts follows a replication-extension approach. A big weakness of replication-extension research is that it also extends the weaknesses of the original research. So it is very important to identify the weaknesses and improve them. The following subsections will discuss on research validity, reliability and generalizability in more details with respect to the current research.

2.3.1. Validity

Research-extensions, that are primarily measurement driven, suffer from biases that might creep in during replication. There can be multiple ways a given construct can be measured and in cases a complex variable might be measured using one variable. For example, a measurement of throughput could be measured in bits/s, but this value might be significantly affected by operation temperature, operation voltage, etc. This might pose a threat to

the validity of the research. Similarly a single method of measuring a variable might also introduce threats to validity; for example, measuring AC/DC voltage using analogue voltmeter which cannot record transient voltages, etc. Research extensions allow improvement to the weaknesses in the original research.

The demonstration of usability of the model gives an insight into the validity of the proposed model. Construct validity cannot be absolutely asserted unless tested under different settings and contexts, but can be argued as ideal and each replication will strengthen the validity.

This thesis ensures validity through testing under different settings and ensuring exhaustive testing at every stage of research.

2.3.2. Reliability

To consider the artefacts of a research to be reliable, the quality of measurement process and inferencing techniques are important contributors. Also, in a replication experiment, if the original measurement processes suffer from weaknesses, these might propagate and affect the end results.

The demonstration of feasibility forms an implicit check on the reliability of the proposed solution. In the process, care was taken to understand the effects and limitation of existing methods and also the methods that are employed in this research.

2.3.3. Generalizability

One of the main goals of this thesis work is to provide a generic solution. Hence, the solution is built with generalizability and scalability as a prime consideration. The demonstration of usability shows how the proposed solution in generalizable and shows how the methods employed in the current research can be extended and applied to others.

Chapter 3.
3 Literature Review and Background

This chapter introduces some related concepts and terminologies needed to understand this thesis while walking through some of the research conducted in relevant areas.

Section 3.1 begins with a look at the branch of study this research is primarily a part of to help understand the positioning of this work. This is followed by a detailed understanding of interleavers and taking a general look at some of the notable research related to interleaver, with focus on 3GPP LTE QPP interleavers, in Section 3.2. Section 3.3 takes gives a background on the SiLago platform developed at KTH. Finally, Section 3.4 presents a short note on the research gap that this thesis is addressing.

3.1. Information Theory and Coding Theory

Information theory is the mathematical study of coding of information sequences into symbols for its quantification, storage, and communication. It was first proposed by Shannon in his landmark paper"A Mathematical Theory of Communication"[7] where he studies the effects of noise in a communication channel. In the words of Shannon, *"The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point."*

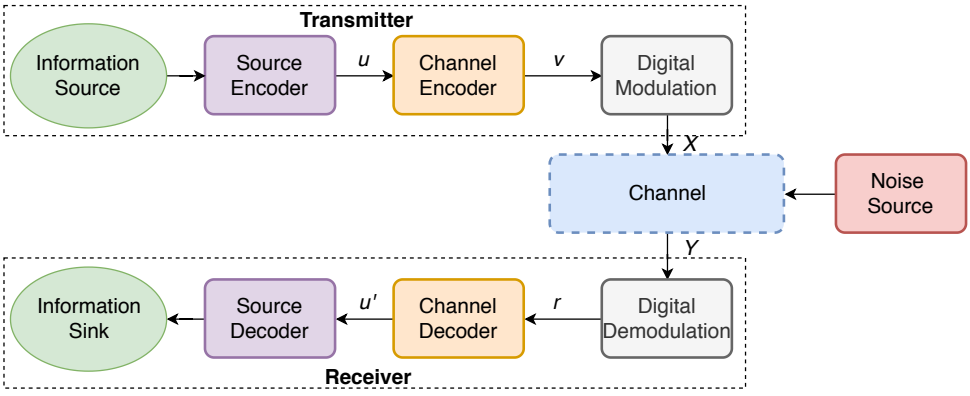


Figure 3.1.: Block diagram of digital communication system

Figure 3.1 depicts a modern digital communication system. Shannon introduced the concept of channel capacity C, which is the maximum amount of information that can be conveyed between the input X and the output Y of a channel. Today this is also referred to as *Shannon Limit*. He proved in his *noisy channel coding theorem* that there exists good

channel coding techniques which can achieve a transmission rate R, with arbitrarily low error probability as long as the information transmitted across the channel is less than the channel capacity C. This has given rise to a new branch of area of study, known today as **Coding Theory** , which is the study of properties of coding techniques.

Various state-of-the-art coding techniques have been developed to address the issue of transmitting information reliably over noisy channels, also referred to as Error Correcting Code (ECC). ECC is a process in which redundancies are added to the original information (vector u) in such a fashion that, in case of minor information corruptions during transmission the original information can be recovered at the receiver (vector u'). Reed-Solomon codes, Turbo codes, Low-Density Parity-Check (LDPC) codes and, more recently Polar codes, are some examples.

3.1.1. Forward Error Correction in Wireless Communication

Wireless transmission of signals is a particularly challenging task. The wireless medium is noisy and susceptible to interference resulting in erroneous transmissions.

For a successful transmission, the receiver has to receive the information transmitted or in the terms used in the Figure 3.1, vector u' should be equal to u. Due to the nature of the noisy channel, vectors v and r may not be the same. For this reason most wireless transmission employs some error correcting techniques. This process of enhancing the chances that information is delivered correct and reliably is known as Forward Error Correction (FEC).

3.1.2. Types of ECC

Primarily there are three main types of ECC, (1) Block codes, (2) Convolutional codes and (3) Compound codes.

Block Codes operate on blocks of data where a K bit input block is converted to a larger N bit output block. These extra $N - K$ bits are also known as parity bits. These parity bits does not contain information but can be used to check the integrity of the received information. Examples of block codes include Reed-Solomon codes, Cyclic Redundancy Check (CRC) codes, etc.

Convolutional Codes on the other hand works on a bit stream and maps an input sequence to an output sequence based on some predetermined mathematical function.

Compound Codes are usually a collection of different types of codes. These types of codes has found widespread usage due to its near Shannon limit error rate performance. Popular examples of compound codes are Turbo Codes , LDPC codes, etc.

3.1.3. Turbo Codes

Turbo Codes, invented by Claude Berrou in 1991[8], was first published in 1993 in the article “Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1”[9]. Turbo code encoders are built by concatenating two Recursive Systematic Convolutional (RSC) codes parallelly and the decoder is constructed using two pipelined soft decoders with feedback loops. Figure 3.2 and 3.3 show the architecture of Turbo Encoder and Turbo Decoder

respectively. It is these feedback loops seen in Figures 3.3 and 3.2 that gives this decoder its characteristic "Turbo" name. The dashed line in Figure 3.3 indicates the feedback loop.

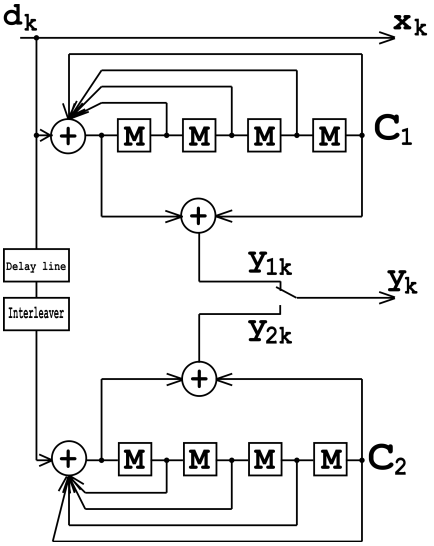


Figure 3.2.: Turbo Encoder

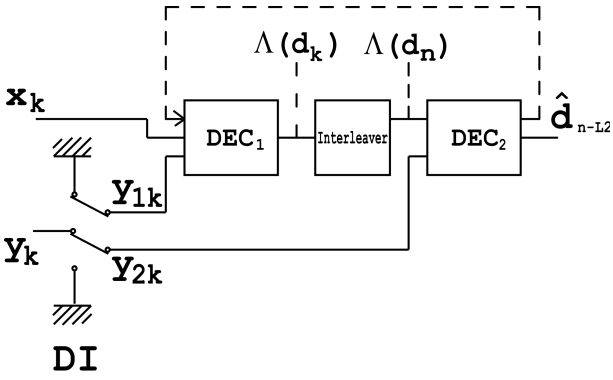


Figure 3.3.: Turbo Decoder

3.1.4. Burst Error and Interleaving

Simply by adding redundancies in information sequences does increase the chances of successful delivery of messages in the communication system. Burst errors are errors in a contiguous section of received information sequence. In wireless communication, bust errors are one of the most common form of interferences. For example, information sequence u is to be transmitted over a channel

$$u = ABC \tag{3.1}$$

Let the ECC create three redundant copies of each symbol as:

$$v = A_1A_2A_3B_1B_2B_3C_1C_2C_3 \tag{3.2}$$

Now this information v is transmitted over a wireless channel. Let there be a burst interference during transmission, lasting three symbol lengths. This results in a stream with errors (represented as X):

$$r = A_1A_2A_3XXXC_1C_2C_3 \tag{3.3}$$

Hence, in-spite of the added redundancies symbol B becomes completely irrecoverable as all copies of it were lost. Or in other words, if the number of errors exceeds the total number of repeated symbols, also known as code rate, then the error correcting algorithm fails to recover the original information sequence.

Now, if the redundant symbols are scrambled within the stream in a random order, for example:

$$v = A_2B_3C_2A_1B_1C_3A_2B_2C_1 \tag{3.4}$$

And the same burst interference occurs, we receive the following sequence.

$$r = A_2B_3C_2XXXA_2B_2C_1 \tag{3.5}$$

Hence, some good copies of all the symbols were received from which, through error correction, the original symbols and information sequence can still be recovered.

This process of scrambling information within the message is known as interleaving and is carried out by interleavers.

3.2. Interleavers

Interleavers are devices that reorder a sequence of symbols $S \rightarrow S'$. In wireless communication, interleaver forms an integral part of the encoder-decoder and can also be seen in figures 3.2 and 3.3. It offers a crucial role in the overall performance of the communication by improving error performance. Also, depending primarily on the requirements of the system using the wireless communication channel, the structure of encoder and decoder can vary vastly. This variation of requirements can significantly affect the design of interleavers. For example, an interleaver for serial decoding will be very different from that implementing a parallel decoding scheme.

At the same time it is a daunting task to accommodate all requirements into one design. Hence, the structure of an interleaver must be considered along with the corresponding encoding or decoding scheme.

For example, for a part of 3rd Generation Partnership Project (3GPP) Long-Term-Evolution (LTE), commonly referred as 4G, Turbo Coding is used along with Quadratic Permutation Polynomial (QPP) interleavers.

3.2.1. 3GPP LTE QPP Interleaver

One of the key requirements for 3GPP was to support data rates of upto 100 Mbps downlink and 50 Mbps uplink along with a low latency of 10 ms. To have high performance turbo codes, the design of the interleaver is critical, specially with s!mall information sequences. Significant research has been conducted in this area, most notably [10–14]. The QPP interleaver used in 3GPP LTE was first proposed by Jing Sun and Takeshita in Jing Sun and Takeshita [15] which in-turn was based on their previous work Takeshita and Costello. QPP interleaver provided a good mix in terms of speed, error correction capabilities and its practicality in efficient hardware implementation.

QPP interleaver for 3GPP LTE, is expressed by the following quadratic form [16]:

$$\Pi\left(i\right) = \left(f_1.i + f_2.i^2\right) \pmod K \tag{3.6}$$

Where block size K is expressed by Equation 3.7, the respective parameters f_1 and f_2 depend are based on the values in Table 3.1, and i ranges between $0 \leq i \leq (K-1)$. A detailed understanding of the rationale behind the choice of f_1 and f_2 coefficient values can be found in [17].

$$K = \begin{cases} 40 + 8f, & 0 \leq f \leq 59 \\ 512 + 16f, & 0 < f \leq 32 \\ 1024 + 32f, & 0 < f \leq 32 \\ 2048 + 64f, & 0 < f \leq 64 \end{cases} \tag{3.7}$$

i	K	f_1	f_2	i	K	f_1	f_2	i	K	f_1	f_2	i	K	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
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

Table 3.1.: Turbo code internal interleaver parameters[1]

[1] From 3GPP TS 36.212 V15.3.0, 2018-09, (Release 15) [16]

3.2.2. Hardware Implementation of 3GPP LTE QPP Interleaver

In its native format Equation 3.6 is not hardware efficient. It involves three multiplications and one modulo division, which are not hardware friendly and can take multiple cycles to compute. But exploiting some algebraic properties of Equation 3.6 the values can be calculated on the fly quite efficiently.

3.2.2.1. Recursive Calculation of $\Pi(i)$

It can be shown that $\Pi(i)$ values can be recursively calculated [18]. In the native format, for every input address i belonging to $0 \leq i \leq (K-1)$ the 3GPP LTE QPP interleaver is defined as:

$$\Pi\left(i\right) = \left(f_1.i + f_2.i^2\right) \pmod K \tag{3.8}$$

Now, for next address $(i+1)$, interleaved address can be calculated as:

$$\begin{aligned}
\Pi\left(i+1\right) &= \left(f_1.\left(i+1\right) + f_2.\left(i+1\right)^2\right) \pmod K \\
&= \left(f_1.i + f_1 + f_2.i^2 + 2.f_2.i + f_2\right) \pmod K \\
&= \left(\left(\left(f_1.i + f_2.i^2\right) \pmod K\right) + \left(\left(f_1 + f_2 + 2.f_2.i\right) \pmod K\right)\right) \pmod K \\
&= \left(\Pi\left(i\right) + \Gamma\left(i\right)\right) \pmod K
\end{aligned} \tag{3.9}$$

Where $\Gamma\left(i\right) = \left(f_1 + f_2 + 2.f_2.i\right) \pmod K$

Similar to $\Pi(i)$, $\Gamma\left(i\right)$ can also be recursively calculated.

$$\begin{aligned}
\Gamma\left(i+1\right) &= \left(f_1 + f_2 + 2.f_2.\left(i+1\right)\right) \pmod K \\
&= \left(\left(\left(f_1 + f_2 + 2.f_2.i\right) \pmod K\right) + \left(2.f_2 \pmod K\right)\right) \pmod K \\
&= \left(\left(\Gamma\left(i\right)\right) + \left(2.f_2 \pmod K\right)\right) \pmod K
\end{aligned} \tag{3.10}$$

These algebraic simplifications of Equation 3.9 and 3.10 make the computation quite hardware efficient. Only $\Gamma\left(0\right)$ and f_2 interleaver addresses can be iteratively generated.

3.2.2.2. Serial 3GPP LTE QPP Interleaver Architecture

Although algebraic simplification makes computation easy, one key aspect is to calculate the modulo. Given that, modular division is successive subtraction until the remainder reaches a non negative value less than the divisor or zero.

Taking an approach proposed by Blakley in “A Computer Algorithm for Calculating the Product AB Modulo M” the modulo can be iteratively calculated. Taking this approach of modulo calculation and the algebraic simplifications, Asghar [5] proposed an efficient hardware architecture shown in Figure 3.4.

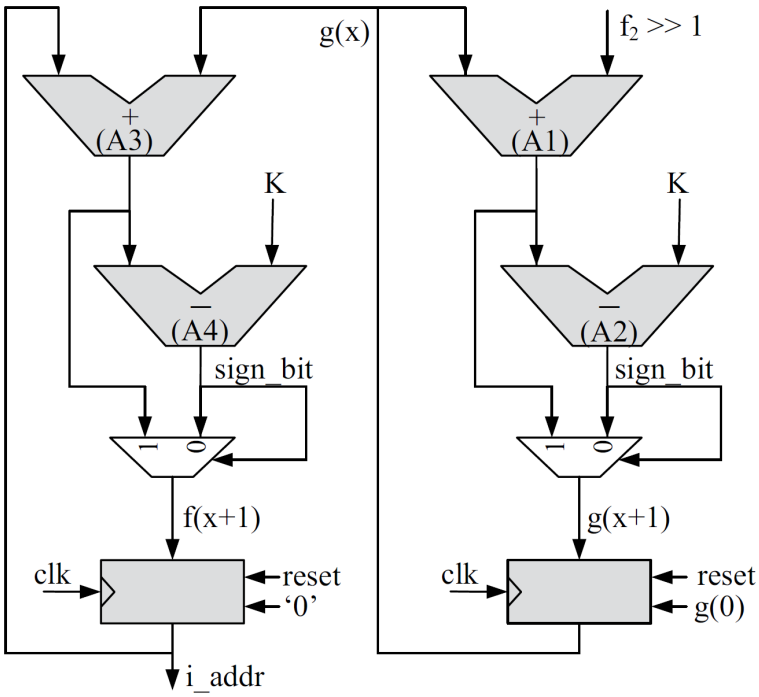


Figure 3.4.: Hardware for Serial 3GPP LTE QPP Interleaver[2]

The right half of the circuit iteratively calculates $\Gamma(i)$, following Equation 3.10 and the left half iteratively calculates $\Pi(i)$, following Equation 3.9. The final interleaved sequence is generated serially at *i_addr*, every clock cycle. The interleaver requires three inputs to start generating interleaved addresses, $\Gamma(0)$, K, and f_2. Computing $\Gamma(0)$ on the fly can be hardware inefficient, hence a look-up table is suggested.

3.2.2.3. Parallel 3GPP LTE QPP Interleaver Architecture

Speed and latency being important constraints, parallelization of design becomes evident. 3GPP LTE QPP interleaver is designed with parallelization in mind. An important feature of this interleaver that allows parallelization is its contention free property [20, 21]. An interleaver, $\Pi(i)$ for $0 \leq i < N$ is considered contention free for a window size W if:

$$\lfloor \Pi(j + tW)/W) \rfloor \neq= \lfloor \Pi(j + vW)/W) \rfloor \tag{3.11}$$

where, $0 \leq j \leq W,\ 0 \leq t,\ t \neq v,\ and\ v < \lfloor N/W \rfloor$.

Utilizing this property, speedup can be achieved by having several parallel decoder[22–26], each operating on a smaller window size W window sizes that are divisors of the block length [27]. Calculating Greatest Common Divisor (GCD) on Equation 3.7:

$$GCD(K) = \begin{cases} 40 + 8f = 8(5 + f) \rightarrow 8, & 0 \leq f \leq 59 \\ 512 + 16f = 16(32 + f) \rightarrow 16, & 0 < f \leq 32 \\ 1024 + 32f = 32(32 + f) \rightarrow 32, & 0 < f \leq 32 \\ 2048 + 64f = 64(32 + f) \rightarrow 64, & 0 < f \leq 64 \end{cases} \tag{3.12}$$

[2] extracted from [5]

Equation 3.12 gives the level of parallelism, with respect to block size K. It can be observed that level of parallelism increases with increasing K. This is an useful feature as it allows higher degree of parallelism for larger block sizes whereby making the process faster. Also, the maximum level of parallelism that 3GPP LTE QPP interleaver achieves across all block sizes is eight (8).

Weithoffer *et al.* [28] in their recent work "25 Years of Turbo Codes: From Mb/s to beyond 100 Gb/s"[28] take a look at some of the state-of-the-art parallel interleavers for Turbo Codes with primary focus on speed and latency. The authors categorize parallel turbo decoder architectures in four categories and some of the notable implementations in each areas.

(a) Parallel MAP architecture (PMAP)[29–32],

(b) Fully Parallel MAP architecture (FPMAP)[33–35],

(c) Pipelined-MAP architecture (P-MAP) also known as XMAP for its characteristic X-shaped decoder architecture.[36–39], and

(d) Unrolled XMAP architecture (UXMAP)[40].

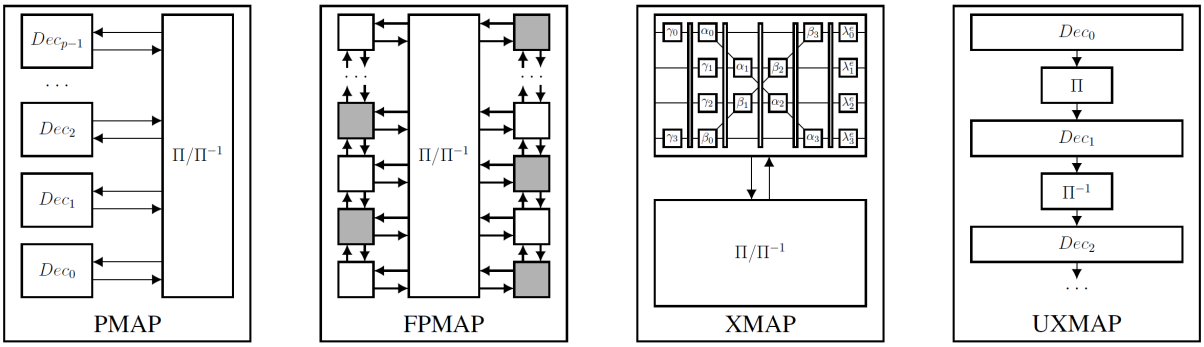


Figure 3.5.: Parallel Turbo Decoder Architectures[3]

Figure 3.5 compares the overall architecture of the four architectures. It can be noted here that the interleaver structure differs significantly amongst different decoding architectures. According to the authors, the architectures that shows most promise with respect to throughput and latency is UXMAP and FPMAP. The implementation details of the interleavers in UXMAP architecture is not quite clear, but the FPMAP interleaver implementation uses a hard-wired interleaver for a fixed block size. This makes the design inflexible, as the interleaver cannot handle multiple block sizes. Some efforts were made in [34] to support different block sizes through transformation of input data. Although the authors managed with some block sizes, they still could not attain all block sizes.

3.2.3. Multimode Interleavers

Apart from QPP interleavers, there also exists multiple standards for interleavers in modern communication systems, each with its own merits and demerits. Figure 3.6 lists some of the interleaver standards used in modern wireless communications.

[3] extracted from [28]

Standard	Interleaver Type	Algorithm / Permutation Methodology
HSPA+	BTC	Multi-Step computation including intra-row permutation computation $S(j)=\left(v\times S(j-1)\right)\%p;\ \ r(i)=T(q(i));$ $U(i,j)=S\left(\left(j\times r(i)\right)\%(p-1)\right);\ qmod(i)=r(i)\%(p-1);$ $RA(i,j)=\left\{RA(i,j-1)+qmod(i)\right\}\%(p-1);$ $I_{i,j}=\left\{C\times r(i)\right\}+U(i,j)$
	Ist, 2nd, and HS-DSCH int.	Standard block interleaving with different column permutations. $\pi(k)=\left(P\left\lfloor\frac{k}{R}\right\rfloor+C\times\left(k\%R\right)\right)\%K_{\pi}$
LTE	QPP for BTC	$I_{(x)}=\left(f_1.x+f_2.x^2\right)\%N$
	Sub-Blk. int.	Standard block interleaving with given column permutations.
WiMAX	Channel interleaver	Two step permutation $M_k=\left(\frac{N}{d}\right)\times\left(k\%d\right)+\left\lfloor\frac{k}{d}\right\rfloor$; and $J_k=s\times\left\lfloor\frac{M_k}{s}\right\rfloor+\left(\left(M_k+N-\left\lfloor d\times\frac{M_k}{N}\right\rfloor\right)\%s\right)$
	CTC interleaver	$I_{(x\%4=0)}=\left(P_0.x+1\right)\%N;\ I_{(x\%4=1)}=\left(P_0.x+1+\frac{N}{2}+P1\right)\%N;$ $I_{(x\%4=2)}=\left(P_0.x+1+P1\right)\%N;I_{(x\%4=3)}=\left(P_0.x+1+\frac{N}{2}+P3\right)\%N$
WLAN	Channel interleaver	Two step permutation $M_k=\left(\frac{N}{d}\right)\times\left(k\%d\right)+\left\lfloor\frac{k}{d}\right\rfloor$; and $J_k=s\times\left\lfloor\frac{M_k}{s}\right\rfloor+\left(\left(M_k+N-\left\lfloor d\times\frac{M_k}{N}\right\rfloor\right)\%s\right)$
802.11n	Ch. Interleaver with Frequency Rotation	Two step permutation as above, with extra frequency interleaving i.e. $R_k=\left[J_k-\left\{\left(\left(\left(i_{ss}-1\right)\times2\right)\%3+3\left\lfloor\frac{i_{ss}-1}{3}\right\rfloor\right)\times N_{ROT}\times N_{BPSC}\right\}\right]\%N$
DVB-H	Outer Conv. interleaver	Permutation defined by depth of first FIFO branch (M) and number of total branches.
	Inner bit interleaver	Six parallel interleavers with different cyclic shift $H_e(w)=\left(w+\Delta\right)\%126$; where $\Delta=0,63,105,42,21$ and 84
	Inner symbol interleaver	$y_{H(q)}=x_q$ for even symbols; $y_q=x_{H(q)}$ for odd symbols; where $H(q)=\left(i\%2\right)\times2^{N_r-1}+\sum_{j=0}^{N_r-2}R_i(j)\times2^j$;
DVB-SH	BTC	$R_c(j)=\left\{R_c(j-1)+Inc(j)\right\}\%32$; and $I(i,j)=\left\{T_{bas}(j)+M_1(i-1,j)\right\}\%C_T$
General Purpose Use	Row and/or Col. Perm. Given	Standard block interleaver with or without row or/and column permutation.

Figure 3.6.: Some interleaver standards[4]

Asghar [5] in his work *Flexible Interleaving Subsystems for FEC in Baseband Processors* showcased an implementation supporting multiple interleaving standards. His research introduced a new approach, primarily using algorithmic transformation and hardware multiplexing and built a multi-mode interleaver with low-latency at a minimum silicon cost.

[4] extracted from [5]

3.3. Silicon Large Grain Objects (SiLago)

SiLago[3] is a Coarse Grain Reconfigurable Fabric facilitating a modular approach towards electronics hardware development. The concept is similar to LEGO bricks i.e. to have a library of hardened blocks (similar to Lego bricks) out of which systems of various types and functionalities can be built. As illustrated in Figure 3.7 SiLago follows a grid based design approach where one grid block is referred to as one SiLago block.

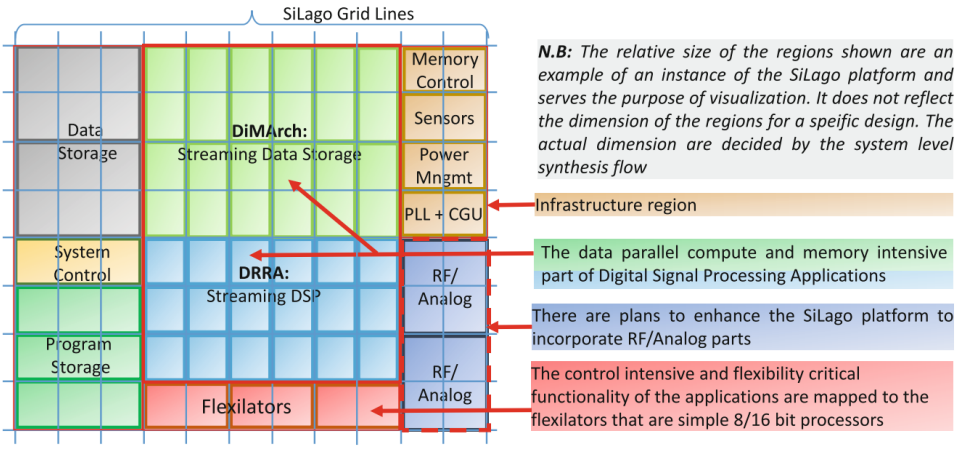


Figure 3.7.: SiLago Platform Example[5]

3.3.1. SiLago Architecture

SiLago blocks replace the concept of standard cell based design to a more higher abstraction micro-architectural based design. A SiLago block is a hardened micro architecture block that is DRC and timing clean and is abuttable (i.e. can be attached to each other, much like LEGO bricks). Based on functionality, SiLago design platform consists primarily of three types of blocks[41]:

1. DRRA - It is a course grain reconfigurable fabric that targets implementing streaming data parallel signal processing functions. It is consists of four sub components
   a) Register File (Reg. File) - It has two read and two write ports to suit the needs of signal processing applications. Each port has its own Address Generation Unit (AGU) *that enables streams of data with spatial and temporal programmability*
   b) Sequencer - A configuration unit designed primarily to handle compile time static signal processing functionalities.
   c) Datapath Unit (DPU) - Quad input processing units to support DSP operations.
   d) Two switch boxes (SWBs) - Allows interconnectivity of register files and DPU between neighbours
2. DiMArch - Distributed memory architecture serves as a scratchpad memory for the DRRA. At its core DiMArch blocks are SRAM banks.

[5] extracted from [41]

3. Flexilators - These are small simple processors that are used for adaptive and control intensive tasks.

To connect one SiLago block to the other there exists two levels of global interconnect:

1. Intra-region - These are interconnects within the DiMArch, DRRA and Flexilators.
2. Inter-region - There are two Network on Chip (NOC) based global interconnects.
   a) High bandwidth circuit switched data NOC - for data movement within regions
   b) Packet switched control NOC - for control, supervision, and configuration

An important feature of SiLago is its ability to create Private Execution Partitions (PREXes). PREXes are dynamic hardware centric custom implementation that enables dynamic customizations to match the requirements of the applications. This allows SiLago to achieve *"ASIC like efficiency"*.

Figure 3.8[41] illustrates an example SiLago instance showing all its sub parts.

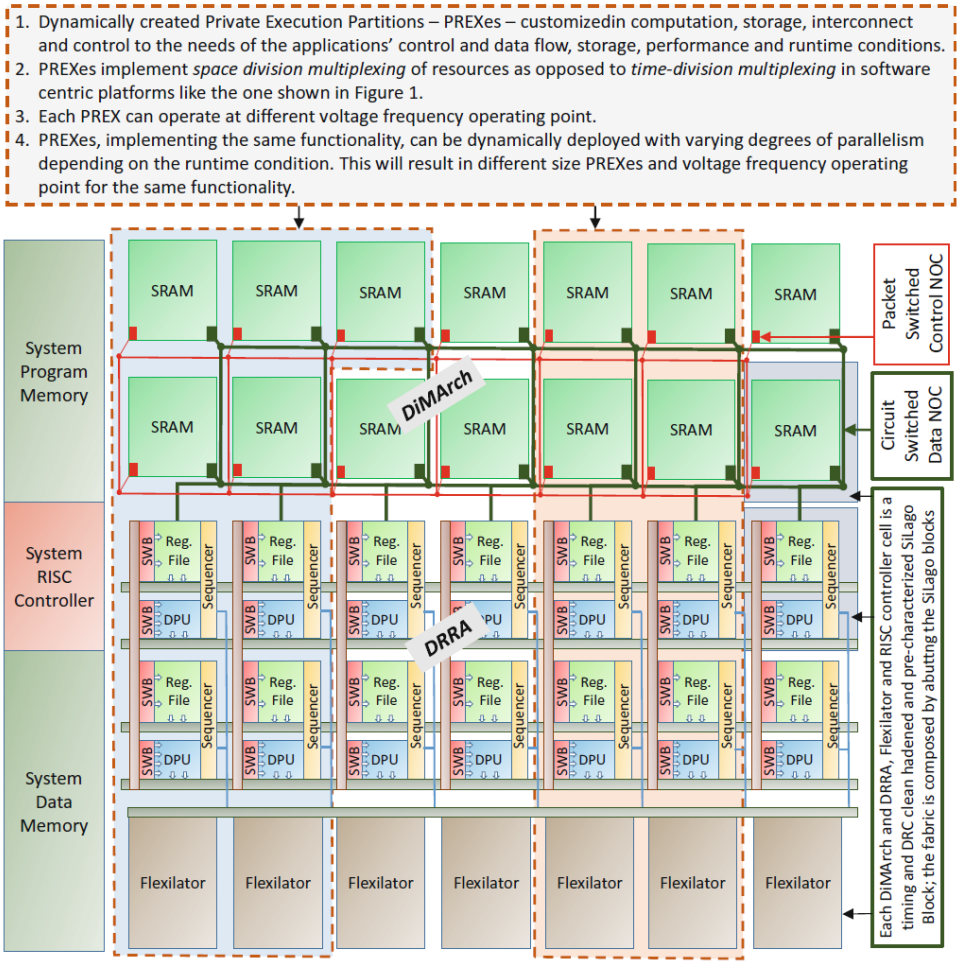


Figure 3.8.: SiLago platform instance with DRRA, DiMArch, flexilators, RISC system controller, and two private execution partitions (PREXes)[6]

[6] extracted from [41]

3.3.2. SiLago Design Flow

SiLago aims to raise the abstraction of design from standard cell to micro-architectural level. SiLago achieves this by hardened, down to physical layout, micro-architectural blocks that are timing and Design Rule Check clean. Each SiLago block comes with a complete set of input and outputs at fixed positions hence allowing them to be abuttable. This approach eliminates the need for logic and physical synthesis.

A SiLago design is primarily a composition of different SiLago blocks. It consists of four stages.

1. **Develpment of SiLago physical platform** - This is similar to conventional standard cell based electronic design automation where a block is designed, verified, implemented and characterized. One major checkpoint is that implementations must follow the abuttable design constraints of SiLago.
2. **Development of Function Implementations (FIMPs) library functions** - A library of the block is developed which can be used by SiLago application level synthesis tool. FIMPs can be composed of multiple SiLago blocks.
3. **Application level synthesis tools explore design space in term of FIMPs** - The SiLago application level synthesis tool, AlgoSil Algorithm-to-Silicon, does a design space exploration based on the FIMP libraries and global constraints from the design.
4. **Hierarchical synthesis script composes a GDSII macro** - Finally the a GDSII macro is generated by abutting selected SiLago blocks. Various synthesis results are also outputted.

Amongst these, stages 1 and 2 requires the maximum engineering effort. In SiLago design flow stages 1 and 2 are one time design efforts and stages 3 and 4 are required for every synthesis effort. This reuse gives SiLago designs it greatest advantage. Figure 3.9 gives a pictorial description of SiLago design flow.

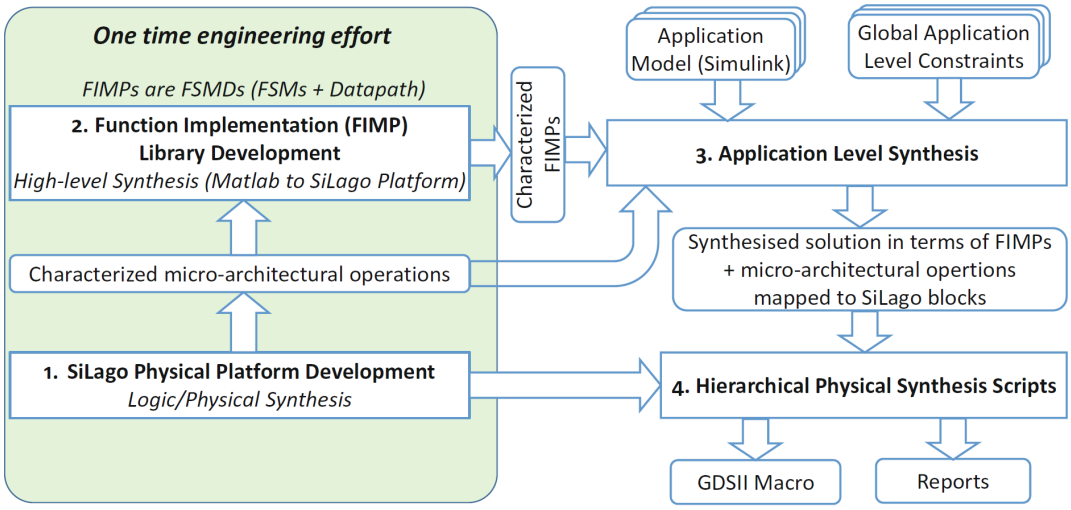


Figure 3.9.: SiLago Design Flow[7]

[7] extracted from [41]

3.4. The Research Gap

Significant research has been conducted in designing and implementing efficient 3GPP LTE QPP interleavers. Some efforts were made in implementing a multi-mode interleaver for SiLago. As the design for this multi-mode interleaver was not made with encoding-decoding techniques and the overall architecture, there still remains a practical gap when it comes to implementable interleavers in SiLago.

Interleaver plays a crucial role in implementing most types of wireless communication. 3GPP LTE (4G) being one of the most common wireless infrastructure, implementation of an interleaver will play a crucial role to fill this gap and create opportunity for research in implementation of encoder-decoders for the same technology.

To the best of author's knowledge, there does not exist a similar implementation of 3GPP LTE QPP interleaver suitable for SiLago, as of writing this thesis. Hence, the principal focus of this thesis was to investigate and implement QPP interleavers for 3GPP LTE communication, suitable for incorporating in SiLago.

Chapter 4. Implementation

This chapter elaborates the implementation of two variants of 3GPP LTE QPP interleavers. Section 4.1 gives an overview of the implementation process. Section 4.2 discusses the choices of tools used to aid the implementation process. Section 4.3 elaborate on two types of interleaver implementations. Finally this chapter concludes with a discussion on the Validity and Reliability of the said implementation in Section 4.5.

4.1. Implementation Process

Since design space exploration process can be extensive in Application Specific Integrated Circuits (ASIC) design, it is extremely important to take a systematic approach towards it. Two ways of conquering such a large design space is by (1) Partitioning or (2) Abstraction. *Partitioning* and hierarchy hides details whereas *Abstraction* lacks details and refines itself incrementally in steps. In case of large designs the abstraction approach is taken primarily because it conducts a stepwise exploration of the design space.

The various perspectives of ASIC design flow can be expressed graphically by the well known Gajski-Kuhn Y-chart, introduced by Daniel Gajski and Robert Kuhn, illustrated in Figure 4.1. The graph is a methodological representation of three domains of ASIC design on three axes and the concentric circles represent abstraction levels. Synthesis is the process of refining from a more abstract specification to a more detailed specification.

This thesis takes the Y-chart approach. The markings **A** through **D** gives an overview of the ASIC implementation path followed by this work.

- **A** - The interleaver was realized in the structural domain with the help of a logic design and circuit simulations tool that has the flexibility of expressing circuits at various levels of structural abstraction.
- **B** - The logic design tool generates a Register-transfer level (RTL) description of the circuit through an export feature. This is only a translation process between two domains and does not involve optimizations or any synthesis.
- **C** - This edge indicates Logic Synthesis, where RTL is synthesized into gate level netlist. Refinement of design is guided by a set of parameters and constraints.
- **D** - This edge indicates Physical Synthesis, where the design is implemented in hardware through further refinement and optimizations. The refinement and optimizations are governed by a set of parameters and constraints.

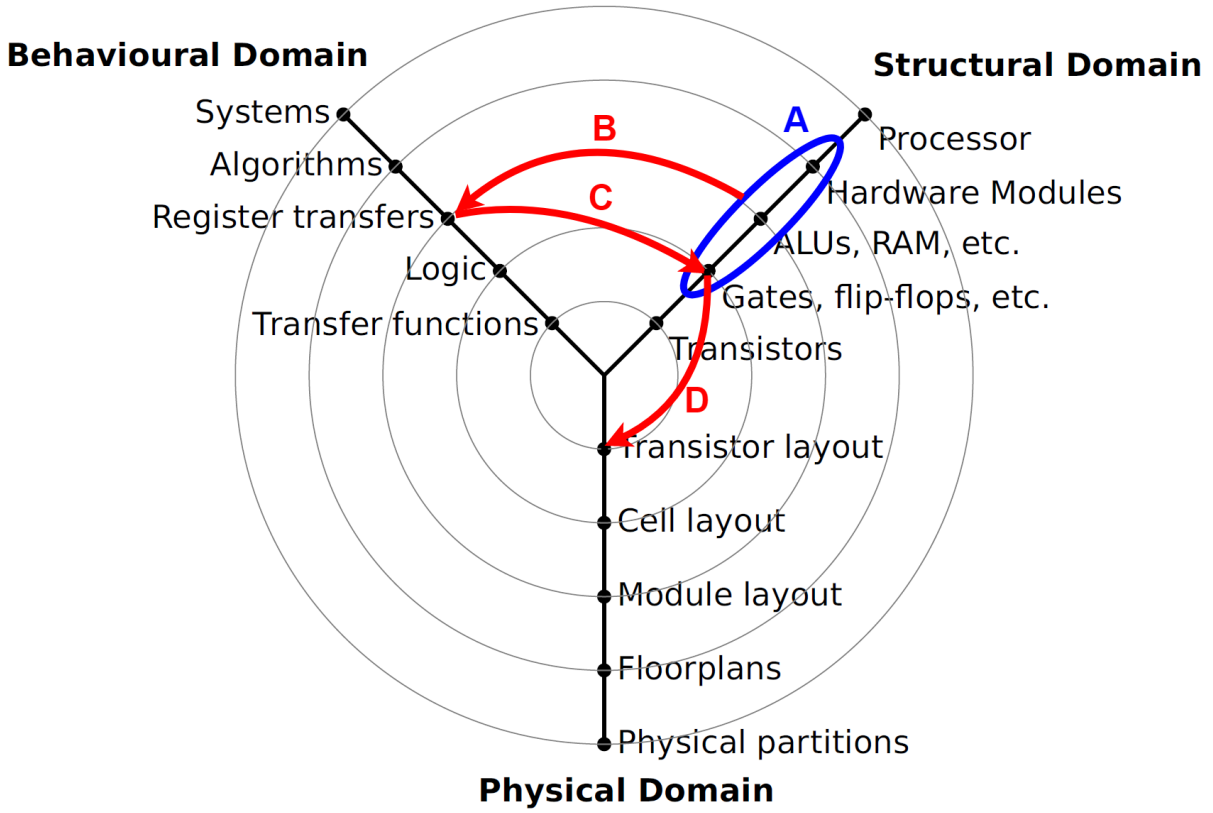


Figure 4.1.: Gajski-Kuhn Y-chart

4.2. Software (SW) Tools

Before discussing the implementations, it is worthwhile to mention the steps and tools used at each step. Choice of the appropriate tools can significantly help and simplify the design space exploration process, right from algorithmic level to its implementation. For ASIC development it is important to think in terms of hardware. Hence, the choice of tools was primarily motivated by this factor. The author would also like to mention that, except for one tool, *Digital*, the author does not endorse the tools mentioned in the subsequent text. The mentions are purely indicative, so that it makes it easier for the reader to reproduce this work.

Reference Management - *Mendeley* was the principal tool used for managing documents and literature review. Features such as "doi search" and "copy as bibtex" helps in getting correct information for citation. Mendeley also supports annotation which helps in note taking and highlighting important sections. And, all of this works cross platform, across multiple devices, through its cloud sync feature, all documents and annotations remains synchronized across all instances.

Logic Design and circuit simulations - *Digital*, a graphical digital logic designer and circuit simulator[42], was used. It is an open source project primarily contributed by Helmut Neemann and is redistributed under GNU General Public License v3.0. At the core, Digital carries the ethos of *Logisim* and its subsequent iteration *Logisim-evolution*. This gives Digital a familiar looking intuitive interface and usability, whilst adding significant new features that simplify digital circuit design and its testing.

Of all the tools used in this thesis, the author found this tool to be one of the most productive tool. In the opinion of the author, the tool has great potential and the author encourages readers to participate in the efforts to enhance this tool further. Some of the main features of Digital includes:

- **Circuit simulation with test vectors** allows extensive and exhaustive testing through simple truth table like test vectors.
- **Component Library** includes most of the necessary or commonly used circuit components.
- **Custom Components** can be created and be reused to create a hierarchical design.
- **VHDL or Verilog Component Description** is possible (with GHDL or Icarus Verilog compilers), which allows describing complex custom components in Hardware Description Language (HDL) or Verilog.
- **VHDL or Verilog Export** feature allows exporting the designed circuit in synthesizable code along with test benches which can be used by HDL simulators to make further cycle accurate simulations.

Test vector generation - *Python* programming language was used to generate the test vectors which were subsequently used in Digital to validate the various circuits. It allowed generating large (50k lines) test vectors, in multiple different configurations, relatively simple, quick and easy to manage.

HDL Simulation - Primarily *ModelSim and Questa Sim* were used. The main purpose of this step was to analyse and test the exported VHDL with the testbenches. This forms an additional verification in the workflow. Initial, smaller designs were simulated on a Laptop PC using ModelSim PE Student Edition and later more larger designs and simulations were carried out on a more powerful Desktop PC using Questa Sim. Although not conducted in this thesis work, much advanced verification can be conducted using psl-language and other features in Questa Sim.

Logic Synthesis - *Synopsys Design Compiler* was used for logic synthesis. The main objective of this exercise was to evaluate two architecture (Radix-2 and Radix-4), and prepare the files necessary for physical synthesis.

Physical Synthesis - *Cadence Innovus Implementation System* was used for the final physical synthesis.

Post Layout Verification - *Questa Sim* was used for the verification of the final design.

Notable Mentions Apart from the tools mentioned above, some other tools that came handy while conducting this research.

- KTHB - For access to research journals.
- TexStudio - For writing this report
- MSExcel - For creating coefficients and manual testing
- Wolfram - For polynomial congruency solving and others.
- Google - The author would like to specially acknowledge the search engine for helping in searching through the enormous human knowledge base with such ease.

4.3. Implementation of parallel 3GPP LTE QPP Interleavers

Based on the implementation process shown in Figure 4.1 the implementation will be discussed in three stages (1) Circuit Design and Simulation (marked A & B in Figure 4.1), (2) Logic Synthesis (C in Figure 4.1), and (3) Physical Synthesis (D in Figure 4.1).

4.3.1. Circuit Design and Simulation

Circuit design was conducted in stages and multiple iterations. Each stage involved testing for functionality before moving to the next iteration.

4.3.1.1. First Iteration - Realizing 3GPP LTE QPP Interleaver

Here much focus was given to understand the workings of 3GPP LTE QPP interleaver. This stage also focused on familiarization with the tools and its limitations. Figure 4.2 portrays the design. This design is not exportable as multipliers and divider units were used. In other words this implementation is a direct application of the mathematical Equation 3.6. To test the circuit small test vectors were constructed for multiple block sizes. The test vectors were hand crafted principally using an Excel Spreadsheet.

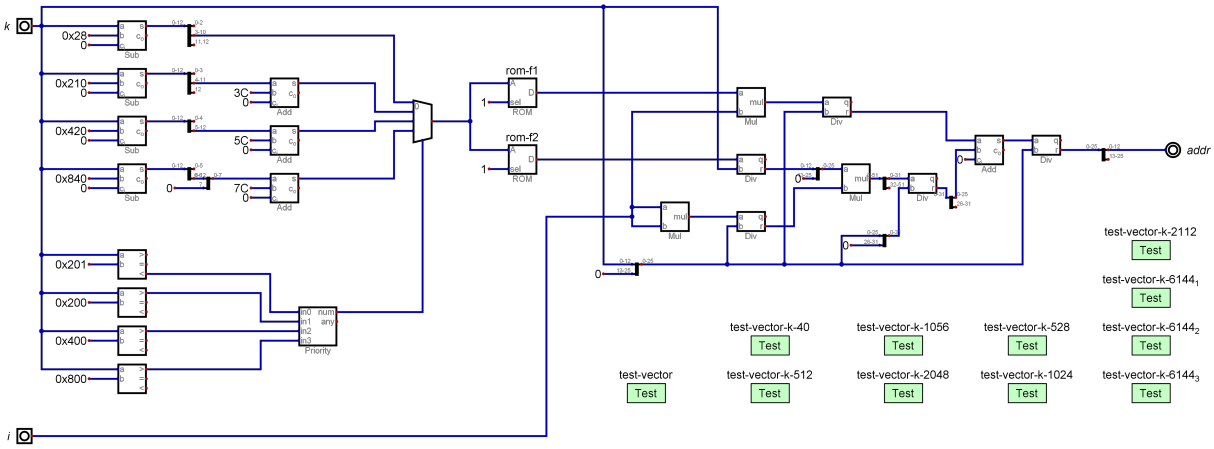


Figure 4.2.: First Iteration 3GPP LTE QPP Interleaver

The main learnings for this implementation is listed as follows:

- **Coefficient generation** - This is the circuit shown on the left hand of figure 4.2 4.2. The circuit implements the Equation 3.6 and realizes an efficient way for lookup.
- **Bit growth** - On the right half it can be observed that the bus width was increased from 13 bits to 26 bits. This was to accommodate bit growth due to multiplication, specially due to coefficient $f_2 i^2$ in Equation 3.6.
- **Test vector** - While testing with large test vectors, certain limitations were identified. This resulted in downloading, modifying and recompiling the application from the source. This exercise did fix the issue temporarily and was reported to the creator and maintainer of the application, who recognized the issue and identified it as a bug in the software to be fixed in future releases.

4.3.1.2. Second Iteration - Implementing HW optimization

From the first iteration the costly nature of multiplication-division was clear. Also in literature it was seen that this is possible with a much simpler recursive calculation approach as discussed in Section 3.2.2.1. The Add-Modulo (AM) architecture was implemented along with a counter for read address generation. The design also included a RAM for storage of coefficients and later retrieval.

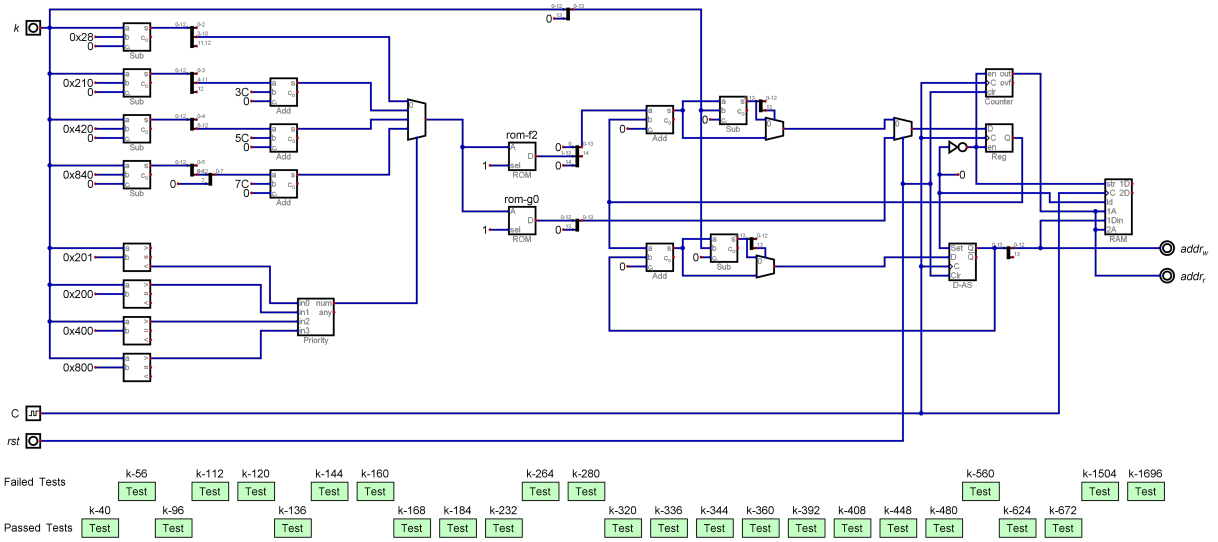


Figure 4.3.: Second Iteration 3GPP LTE QPP Interleaver

The main learnings for this implementation is listed as follows:

- **Roll over of read address generation** - With continued application of clock pulses, the recursive calculation of interleaved addresses automatically rolls over and restarts. On the other hand the read address generation counter does not reset resulting in memory leak in the RAM.
- **Limitation of AM units** - A major weakness was identified through the exhaustive testing of the AM units. With certain values of block sizes (K's) and bit address lead to erroneous results (indicated in the upper row of test cases, *Failed Tests*). Further investigation on this exposes the limitation of the proposed recursive address generation.

4.3.1.3. Third Iteration - Correcting design weakness

The AM units were modified to correct the weakness identified in the previous design iteration and referred to as Add-Modulo-plus (AM+). Also some control and synchronization signals were added. A more hierarchical approach was chosen and coefficient generation was moved into a separate component. Registers for storing coefficients were added as a precursor to future parallelization. The resulting interleaver circuit is shown in Figure 4.4

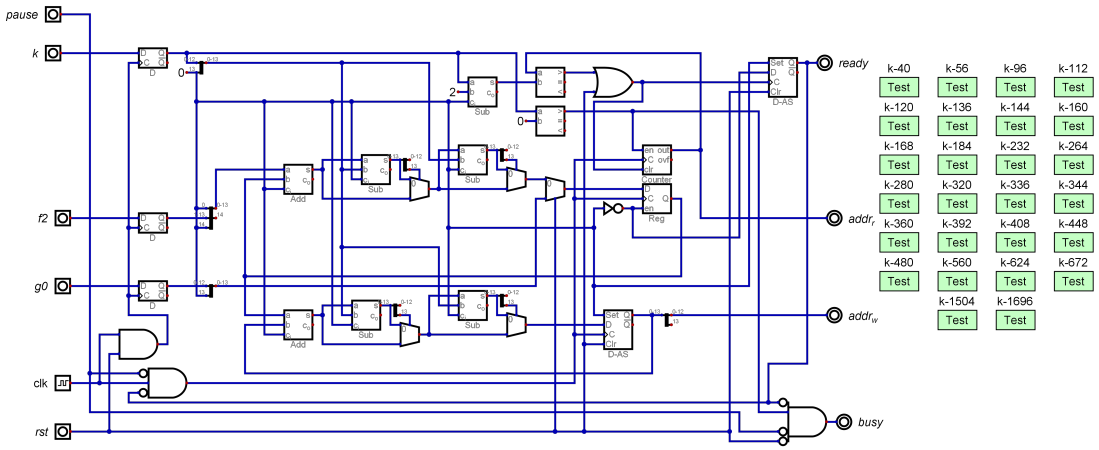


Figure 4.4.: Third Iteration 3GPP LTE QPP Interleaver

4.3.1.4. Fourth Iteration - Parallelizing

This first effort of parallelization is an aggregation of seven individual interleavers sharing one coefficient generator. To suit the needs of SiLago and maximize bus bandwidth, with the help of a custom FIFO buffers, generated addresses are aggregated into a large 256 bit word. Each interleaver can be configured to generate addresses for different block sizes.

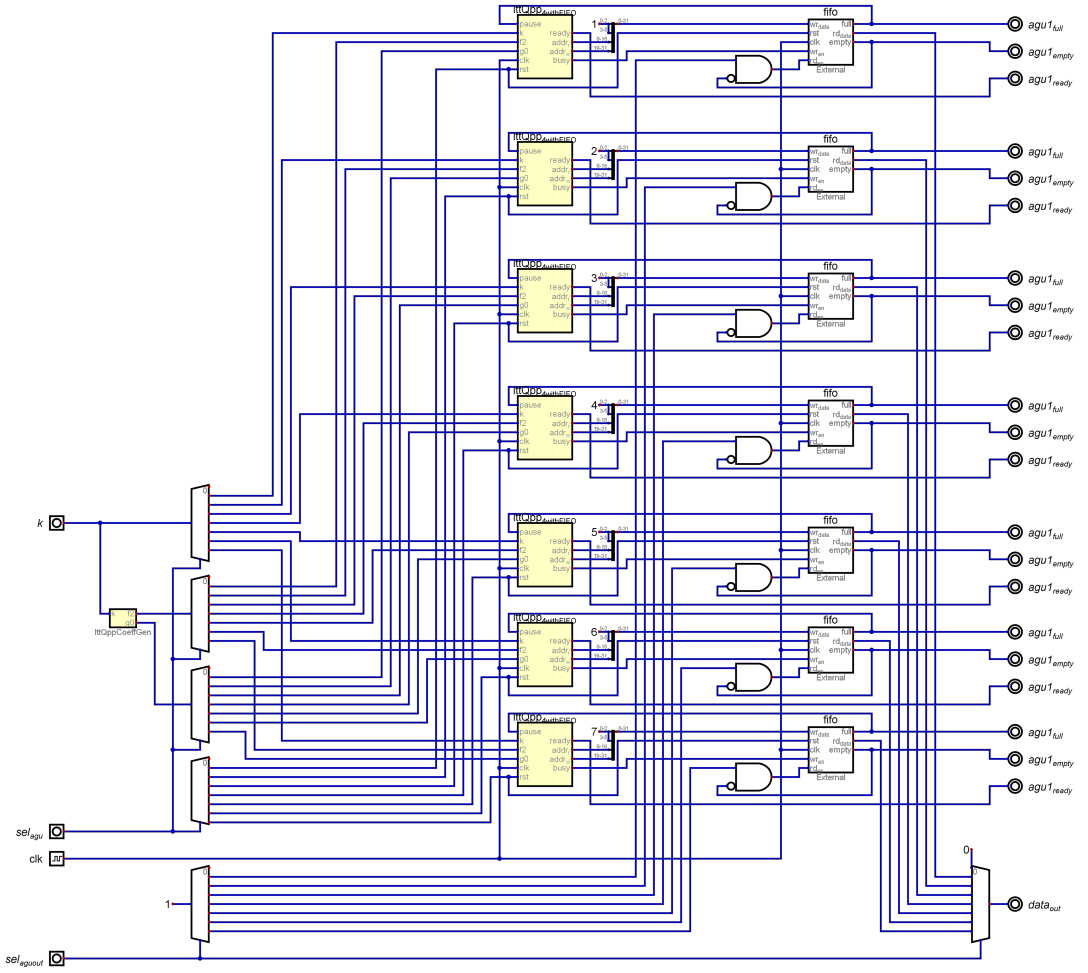


Figure 4.5.: Fourth Iteration 3GPP LTE QPP Interleaver

The main learnings for this implementation is listed as follows:

- **Latency** - Although this architecture can achieve higher throughput but for each block it would still require a long time, specially for large block sizes. This is not desirable.

- **Control FSM** - Such architecture also complicates the control FSM and results in more precise timing for write and read.

4.3.1.5. Fifth Iteration - Radix-2 parallel

To address latency and a simpler design, the radix-2 parallel architecture as discussed by Asghar in his thesis *Flexible Interleaving Subsystems for FEC in Baseband Processors*[5], was adopted. Such parallelism requires additional memory for coefficient generation. The coefficient generation was modified to suit the requirements, as depicted in Figure A.1. Also, a more hierarchical design approach was taken, the detailed circuit can be found in Appendix A.

Exhaustive testing of this design was conducted. The test vectors were generated using a Python script. The Python script can be found in Appendix B

Following the Asghar design, during initial testing another big weakness identified was the reuse of $g(x)$ values in parallel interleavers. This was addressed by adding an additional $g(x)$ stage. The algorithm followed for Radix-2 address generation is given in Algorithm 1. Figure 4.6 gives the final overall architecture of implementation. This design was exported as VHDL for further analysis.

Algorithm 1: Address Generation Algorithm for Radix-2 Parallel 8

Initialization :

$N_{SISO} = 8$;
$K_{sub\_blk} = K/N_{SISO}$;
$g_{even}(0) = \{f_1 + f_2\} \mod K$;
$g_{odd}(0) = \{f_1 + f_2 + 2f_2K/N_{SISO}\} \mod K$;
$I_1(0) = 0$;
for $x = 2 \rightarrow N_{SISO}$ **do**
    $I_x(0) = \{\} \mod K$;
end

Execution :

for $j = 1 \rightarrow (K_{sub\_blk} \div 2) - 1$ **do**
    $g_{even}(j) = \{g_{even}(j-1) + 2f_2\} \mod K$;
    $g_{odd}(j) = \{g_{odd}(j-1) + 2f_2\} \mod K$;
    for $x = 1 \rightarrow N_{SISO} \div 2$ **do**
        $I_{2x-2}(j) = \{I_{2x-2}(j-1) + g_{even}(j)\} \mod K$;
        $I_{2x-1}(j) = \{I_{2x-1}(j-1) + g_{odd}(j)\} \mod K$;
    end
end

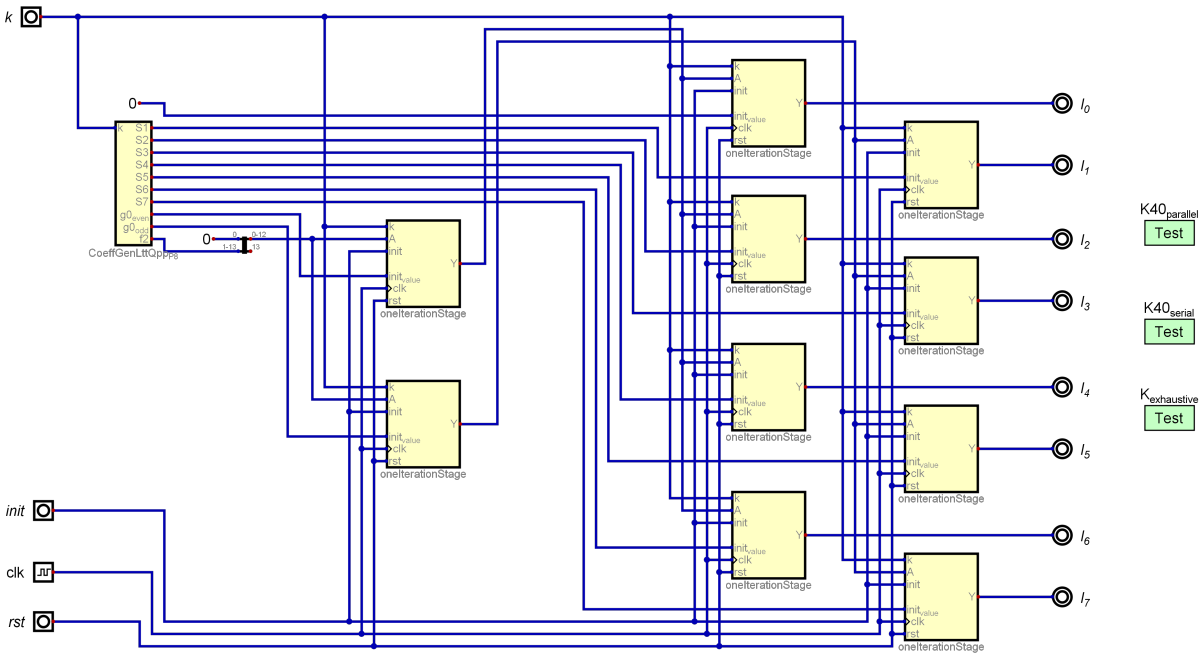


Figure 4.6.: Radix-2 Parallel 3GPP LTE QPP Interleaver

4.3.1.6. Sixth Iteration - Radix-4 parallel

With most of the design weaknesses identified and rectified in Radix-2 parallel implementation, the radix-4 implementation was relatively simpler. The major difference being the circuit design, the rest of the implementation remains the same as that for Radix-2 implementation. The AM units in the original design did have the weakness discussed in Section 4.3.1.2 where it failed to generate addresses for a few block sizes. The design was recreated with the modified AM+ units and tested exhaustively with a single large test vector. The algorithm followed for Radix-4 address generation is given in Algorithm 2. Figure 4.7 shows implementation of Asghar proposed design for a Radix-4 parallel interleaver. This design was exported as VHDL for further processing.

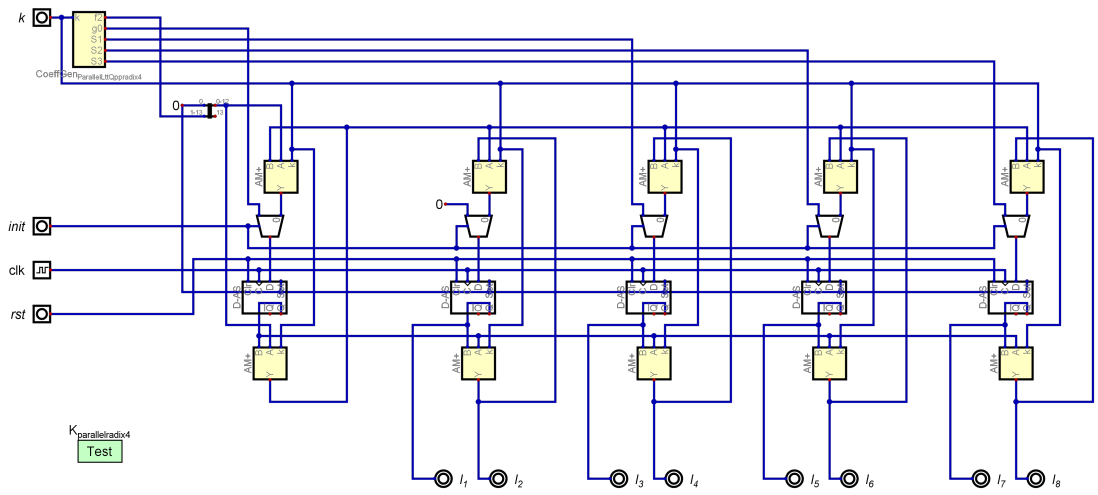


Figure 4.7.: Radix-4 Parallel 3GPP LTE QPP Interleaver

Algorithm 2: Address Generation Algorithm for Radix-4 Parallel 4

Initialization :

$K_{sub\_blk} = K/N_{SISO}$;
$g(0) = \{f_1 + f_2\} \mod K$;
$I_1(0) = 0$;
$I_2(0) = g(0)$;
for $x = 2 \rightarrow N_{SISO}$ **do**
  $I_{2x-1}(0) = S_{x-1} = \{S_{x-2} + S'_{x-1}\} \mod K$;
  $I_{2x}(0) = \{I_{2x-1}(0) + g(0)\} \mod K$;
end

Execution :

for $j = 1 \rightarrow (K_{sub\_blk} \div 2) - 1$ **do**
  $g(2j-1) = \{g(2j-2) + 2f_2\} \mod K$;
  $g(2j) = \{g(2j-1) + 2f_2\} \mod K$;
  for $x = 1 \rightarrow N_{SISO}$ **do**
    $I_{2x-1}(j) = \{I_{2x-1}(j-1) + g(2j-1)\} \mod K$;
    $I_{2x}(j) = \{I_{2x}(j-1) + g(2j)\} \mod K$;
  end
end

Where, $S(x) = \left(f_1.x.K_{sub_block} + f_2.(x.K_{sub_block})^2\right) \mod K$, and, $S'(x) = f_1.K_{sub_block} + (2x-1)\, f_2.(K_{sub_block})^2$

4.3.1.7. Questa Sim Simulation

The exported VHDL contained one source for the design and the test vectors were exported as testbenches. The circuit was analysed compiled and simulated in Questa Sim.

Although the VHDL code was auto generated, certain statements had to modified to successfully compile and simulate the design. On making the changes the design could be simulated successfully. A schematic view was generated and compared with the original design. It should be noted that the schematic exported from Questa Sim is reconstructed from the VHDL source files. Hence, this stage serves as an additional step for validation of the RTL expression of the circuit.

4.3.2. Logic Synthesis

The modified VHDL was imported in Synopsys's Design vision and synthesized in 28nm technology. Parameters for synthesis and constraints were primarily based on the parameters and constraints used for synthesizing SiLago. Table 4.1 summarizes the parameters and Synopsys Design Constraints used for 28nm logic synthesis. TCL script used for logic synthesis can be found in Appendix C. Two parameters were reported for further analysis (1) Area, and (2) Power.

Settings	Parameters
Library Objects	
target_library	(project specific)
symbol_library	tcbn90g.sdb
synthetic_library	standard.sldb dw_foundation.sldb
link_library	(project specific)
Design Environment Objects	
set_operating_conditions	ss0p855v125c
set_wire_load_model	segmented
set_driving_cell	-lib_cell BUFFD6BWP30P140 [remove_from_collection [all_inputs] clk]
set_load	-pin_load 0.001 [all_outputs]
Design Optimisation Constraints	
create_clock	-name "clk" -period 5 -waveform { 0 2.5 } { clk }
set_clock_gating_style	-sequential_cell latch
set_clock_uncertainty	0.1 [get_clocks clk]
set_clock_gating_style	-sequential_cell latch
set_input_delay	-clock clk 0.1 [remove_from_collection [all_inputs] clk]
set_output_delay	-clock clk 0.1 [all_outputs]

Table 4.1.: Logic Synthesis Parameters

4.3.3. Physical Synthesis

Physical synthesis was done in 28nm. The SiLago system is envisioned to be utilizing modern state-of-the-art silicon technology. In the post Dennard era, benefits from scaling down comes at a cost, that is mostly power. This results in the phenomenon commonly known as "Dark Silicon", i.e. all the circuit cannot be powered at the same time. This was discussed extensively in *The Dark Side of Silicon*[41] and how SiLago mitigates this issue. Hence the decision to do the final physical synthesis in 28nm architecture was made.

The 28nm technology offers eight metal layers out of which the two topmost layers were used for the power and ground lines. Parameters for synthesis and constraints were primarily based on the parameters and constraints used for synthesizing SiLago. These parameters can be found in the TCL script used for logic synthesis given in Appendix C. The steps followed for physical synthesis in Innovus is summarised in Appendix C.

4.3.4. Post Layout Verification

Verilog NetLists generated from physical synthesis were saved. These NetLists along with the technology library and the testbenches (used before logic synthesis), were imported into Questa Sim. Exhaustive simulations were conducted with these testbenches along with a more realistic clock to verify the operation of the circuit.

4.4. Summary of Workflow

Figure 4.8 illustrates the thesis workflow, that is from knowledge through implementation and verification. The figure indicates the steps undertaken from left to right, some of the tools used at each step and at the bottom indicating artefacts generated at each step.

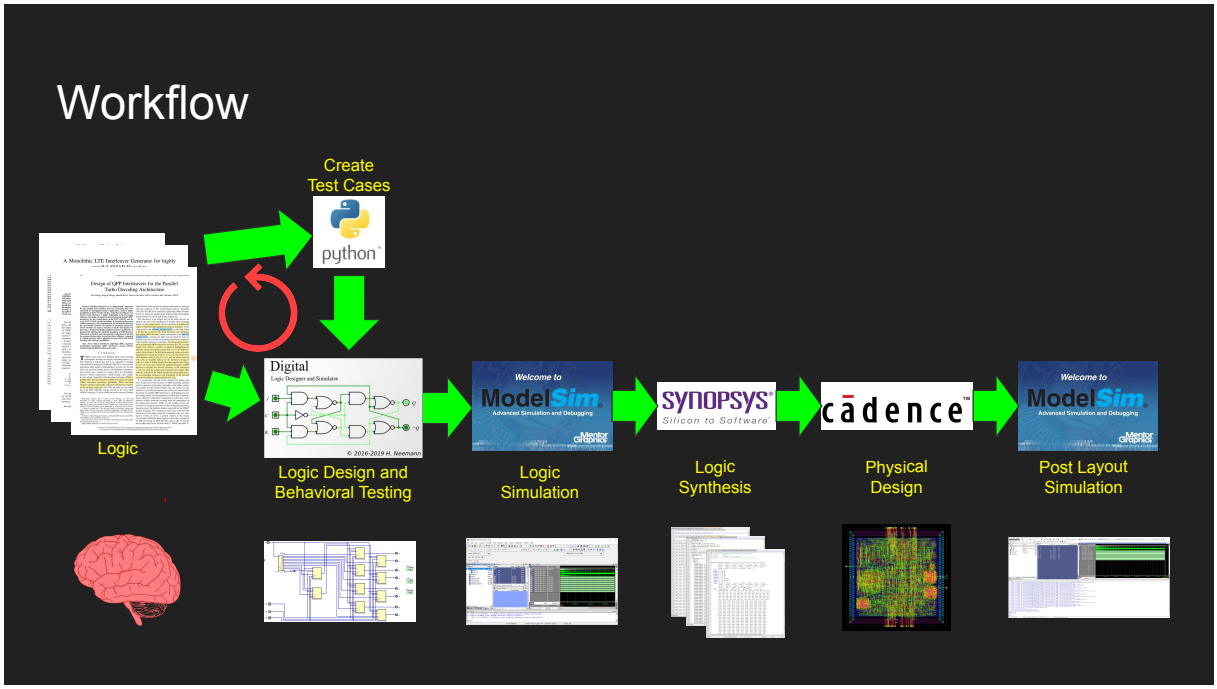


Figure 4.8.: Thesis WorkFlow[1]

4.5. Validity and Reliability

It is discussed in Section 2.3 that multiple measures were taken to ensure validity and reliability of the design. Exhaustive circuit simulation plays a critical part in circuit design. It helped identifying key design flaws of the existing design discussed in literature. This has resulted in a very robust design that can successfully compute addresses for every possible combinations.

The reliability of the design was doubly verified with two types of simulations. The first using the test vector feature while circuit design followed by simulations in ModelSim and Questa Sim.

Use of reliable tools with a good proven track record for synthesis is also a major contributor towards a reliable design.

[1] All product names, logos, and brands are property of their respective owners and are for identification purposes only.

Chapter 5.
5 Results and Analysis

Multiple simulations were conducted at different iteration stages of the implementation, as discussed in the previous chapter 4. This chapter presents the notable results obtained from the experiments and provides analysis for them.

The chapter is organized in five sections. The first section 5.1 reflects on the result from the initial phase of the implementation. This is followed by Logic and Physical Synthesis results in Sections 5.2 and 5.3 respectively. Section 5.4 lists some of the observations made during post layout verification. Finally it concludes with a general discussion on interpretation of the results in Section 5.5.

5.1. Circuit Design and simulations

Each design iteration in circuit designing was thoroughly tested. Two major weaknesses were identified through these tests. (1) failure of AM units for certain block sizes, and, (2) reuse of $g(x)$ values in Radix-2 parallel interleaver with parallelism of 8.

5.1.1. Failure of AM units

The principle idea behind each iteration stage in the design is to compute the following in an efficient way:

$$Y = (A + B) \mod C \tag{5.1}$$

The modulo can be replaced by addition and subtraction by solving the linear congruence and Equation 5.1 can be rewritten as:

$$Y = A + B - nC, \; where \; n \in \mathbb{N} \tag{5.2}$$

Further exploiting mathematical properties of 3GPP LTE QPP interleaver coefficients. It is observed that in most cases the value of n in Equation 5.2 ranges between 0 and 1. This simplifies the equation to take the form:

$$Y = \begin{cases} A + B & n = 0 \\ A + B - C & n = 1 \end{cases} \tag{5.3}$$

Extensive testing of the design revealed that there exists certain combinations where Equation 5.3 does not hold true. As an example, for block size 160, the recursive 3GPP LTE

QPP interleaver takes the following form:

$$\begin{aligned} f(x) &= \left(21x + 120x^2\right) \mod 160, \ where\ 0 \leq x \leq 159 \\ g(x) &= (21 + 2 \times 120x + 120) \mod 160, \ where\ 0 \leq x \leq 159 \\ f(x+1) &= (f(x) + g(x)) \mod 160, \ where\ 0 \leq x \leq 158 \\ g(x+1) &= (g(x) + 2 \times 120) \mod 160, \ where\ 0 \leq x \leq 158 \end{aligned}$$

Applying the simplification of Equation 5.3:

$$g(x+1) = \begin{cases} (g(x) + 2 \times 120), \ where\ 0 \leq x \leq 158 & if\ g(x+1) < 160 \\ (g(x) + 2 \times 120) - 160, \ where\ 0 \leq x \leq 158 & if\ g(x+1) \geq 160 \end{cases}$$

$$f(x+1) = \begin{cases} (f(x) + g(x)), \ where\ 0 \leq x \leq 158 & if\ f(x+1) < 160 \\ (f(x) + g(x)) - 160, \ where\ 0 \leq x \leq 158 & if\ f(x+1) \geq 160 \end{cases}$$

Solving $f(x)$ and $g(x)$ iteratively for two iterations yields the following:

$f(0) = 0$

$g(0) = 141$

$f(1) = f(0) + g(0) = 141$

$g(1) = g(0) + 2 \times 120 = 381, \ which\ is \geq 160, \ hence\ 381 - 160 = 221$

$f(2) = f(1) + g(1) = 141 + 221 = 362, \ which\ is \geq 160, \ hence\ 362 - 160 = 202$

This value of $f(2)$ is incorrect.

To overcome this weakness, Equation 5.3 was expanded to solve n values up till 2. The resulting equation can be expressed as follows:

$$Y = \begin{cases} A + B & n = 0 \\ A + B - C & n = 1 \\ A + B - C - C & n = 2 \end{cases} \tag{5.4}$$

5.1.2. Failure to reuse $g(x)$ coefficient

Figure 5.1 shows proposed design of Asghar for a Radix-2 parallel interleaver with parallelism of 8.

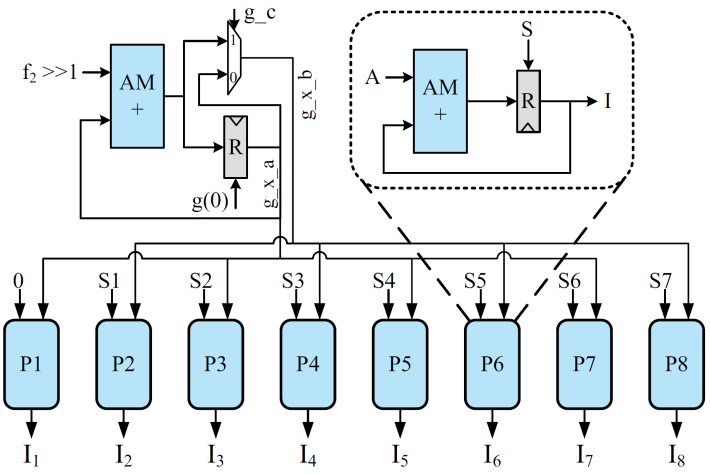


Figure 5.1.: Asghar's proposed design for 3GPP LTE QPP interleavers[1]

This design failed to generate correct coefficients for g_c (referred as $g(x)$ in this thesis) for significant number of block sizes (216, 368, 464, 496, 504, 592, 624, 656, 688, 752, 784, 816, 848, 880, 912, 928, 944, 976, 1056, 2112, 4160, 4288, 4416, 4544, 4672, 4736, 4800, 4864, 4928, 4992, 5056, 5120, 5184, 5248, 5312, 5376, 5440, 5504, 5568, 5632, 5696, 5760, 5824, 5888, 5952, 6016 and 6080).

It was observed that, in the failed cases, odd and even addresses uses different set of $g(x)$ values. The author believes that, in the design proposed in Figure 5.1, the even and odd addresses can be programmed to use different $g(x)$ values. This was identified to some extent by Asghar. However it is only limited to $g(x)$ value of the previous iteration, which is not enough.

The solution to this was identified as to use two separate stages to calculate $g(x)$ coefficients for even and odd addresses. For even addresses it is initialized with $g_{even}(0) = \{f_1 + f_2\} \mod K$, and for odd it is initialized with $g_{odd}(0) = \{f_1 + f_2 + 2f_2K/N_{SISO}\} \mod K$.

5.1.3. Questa Sim Logic Simulation

Figure 5.2 & 5.3 are schematics exported from Questa Sim, (For schematics of the individual sub components refer to Appendix A). On comparing Figures 5.2 with 4.6 and Figures 5.3 with 4.7, it seen that same circuits were reconstructed from VHDL source.

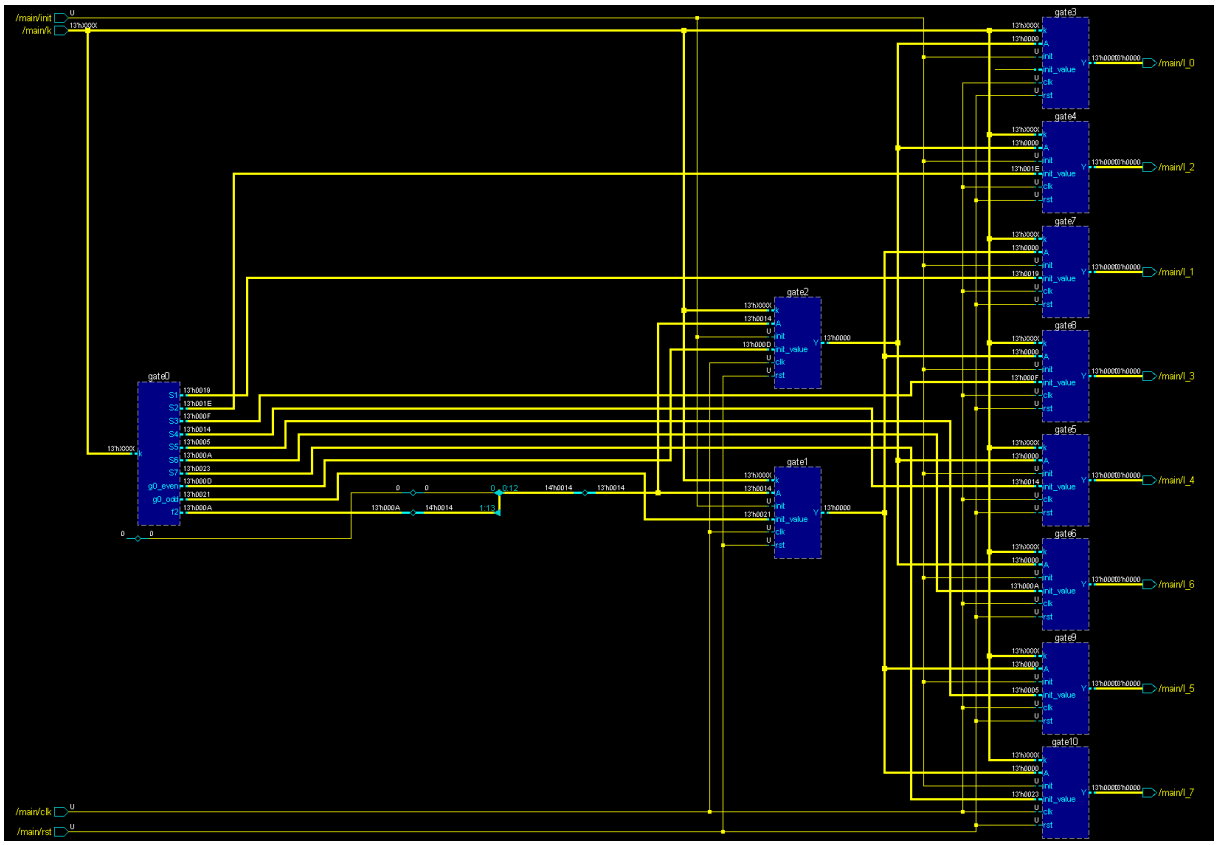


Figure 5.2.: Radix-2 Schematic exported from Questa Sim

[1] Extracted from pg 130 of *Flexible Interleaving Subsystems for FEC in Baseband Processors*[5]

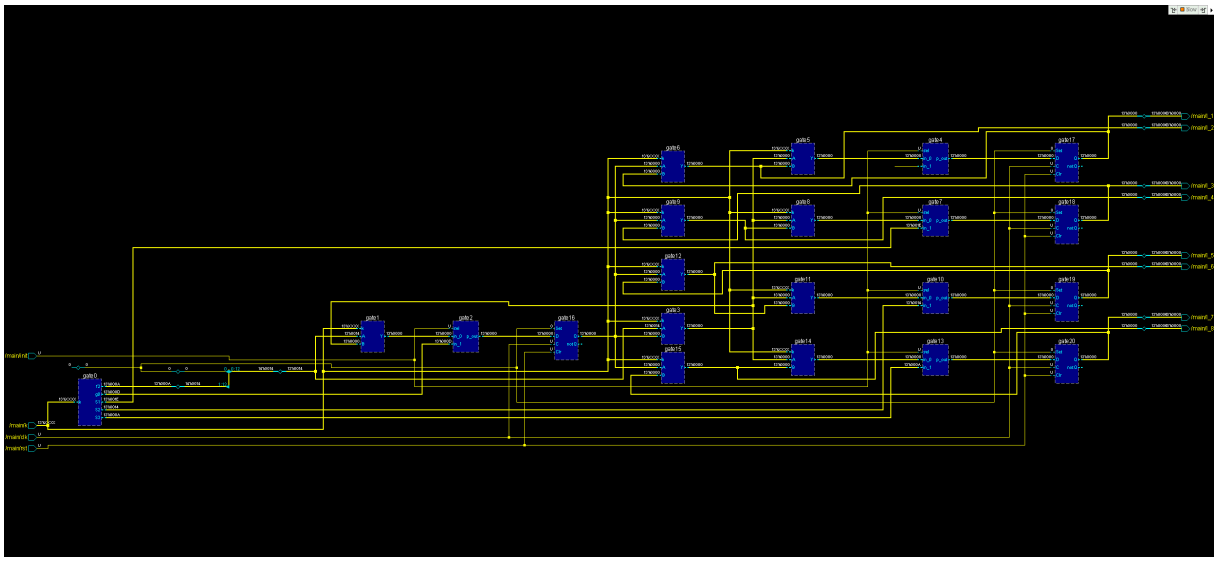


Figure 5.3.: Radix-4 Schematic exported from Questa Sim

The successful execution of the testbenches in Questa Sim and a successful recreation of circuit from source code serves as a verification step of the VHDL export feature of *Digital*. Apart from a few minor compiler warnings the export feature seems very reliable.

5.2. Logic Synthesis

Table 5.1 summaries the area and power report from the Synopsys' Design Compiler. It is important to note that Radix-2 contains a larger look-up table than Radix-4 implementation. Hence, it was expected that the Radix-2 implementation would be larger and more power consuming. It was observed that, contrary to the expectations the Radix-2 implementation occupies a smaller area and consumes less power than Radix-4 implementation.

Multiple warnings were noted relating to open or unconnected ports. This was found to be the unused output ports of design components like comparators, adders, etc. Also some small amount of slack was noted for Radix-2 implementation. This was ignored for the time as such small amounts of slack can be optimized and/or compensated for in Physical Synthesis.

Observed Parameter	Radix-2	Radix-4
Total Cell Area	**4188.870015** um^2	**4431.545997** um^2
Combinational area:	3697.848004 um^2	4183.829992 um^2
Buf/Inv area:	411.642004 um^2	688.590001 um^2
Noncombinational area:	491.022011 um^2	247.716005 um^2
Total Power	**285.9 uW**	**307.2 uW**
Cell Internal Power	218.5090 uW	210.3547 uW
Net Switching Power	61.4271 uW	89.9725 uW
Cell Leakage Power	5.9872 uW	6.8826 uW

Table 5.1.: Logic Synthesis Results

5.3. Physical Synthesis

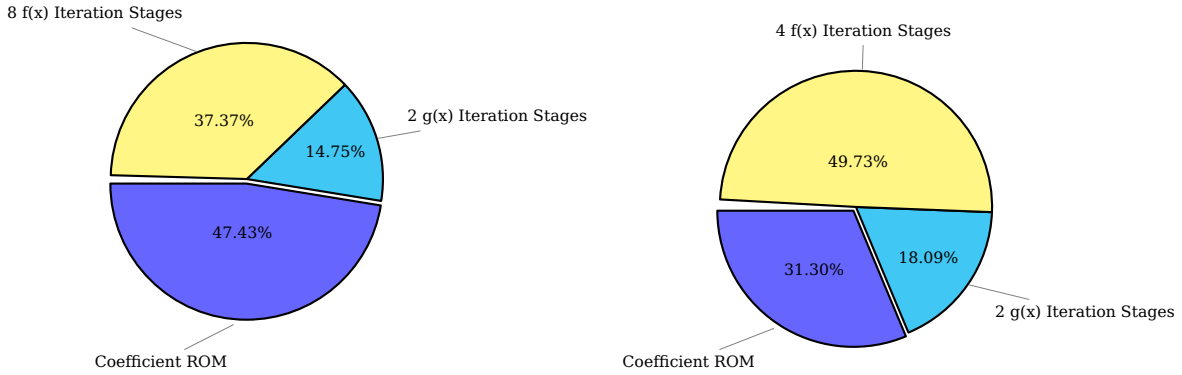
Table 5.1 summaries the final floor area and power consumption from Cadence's Innovus. Similar to the observations made in Logic Synthesis, the Radix-2 implementation seems to be more efficient both in terms of area and significantly efficient in terms of power consumption.

Comparing the values in Table 5.2 with that of Table 5.1, the area estimation of Radix-2 implementation was relatively close but that of Radix-4 was significantly different. Whereas the estimation of power consumption was found to differ by almost a degree of magnitude. This difference highlights the limitations of the current Electronic Design Automations (EDA) tools, stressing that at early stages of EDA these values are indicative only.

Observed Parameter	**Radix-2**	**Radix-4**
Total Cell Area	**4157.244** um^2	**6395.76** um^2
Coefficient ROM	1971.774 um^2	2001.636 um^2
Total Power	**1.04696833 mW**	**2.61775758 mW**
Total Internal Power	0.62014208 mW	1.49331923 mW
Total Switching Power	0.42073040 mW	1.11369284 mW
Total Leakage Power	0.00609584 mW	0.01074551 mW

Table 5.2.: Physical Synthesis Results

Figures 5.4(a) and 5.4(b) shows the percentage share of area for different parts of the design for Radix 2 and Radix 4 implementations respectively. In general, the larger area in the Radix-4 design can be attributed to additional memory elements required for the computation. Also, effort was not made to optimize the design for power.



(a) Radix-2 (b) Radix-4

Figure 5.4.: Area distribution among different sub-components

Figure 5.5 and 5.6 shows the final layout of Radix-2 and Radix-4 implementations respectively. The nine look-up coefficient-ROMs in the Radix-2 implementation can be identified easily as the nine circular blobs on the left and right edges. Similarly the four look-up coefficient-ROMs of Radix-4 implementation can be observed as the 3 distinct blobs and one a more sparse blob on the left. The Radix-2 implementation also seems to be more dense. This might be due to the stricter slack requirements of the design, which could have resulted in shorter wires.

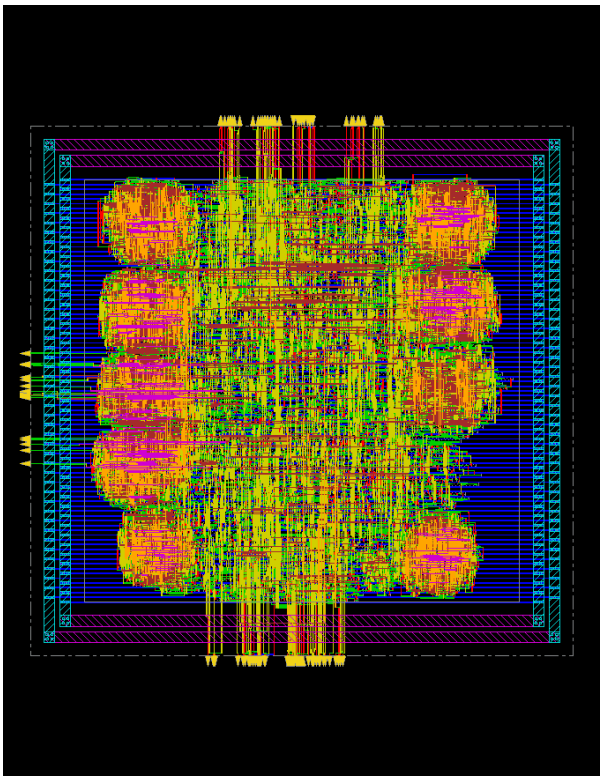


Figure 5.5.: Radix-2 Physical Layout

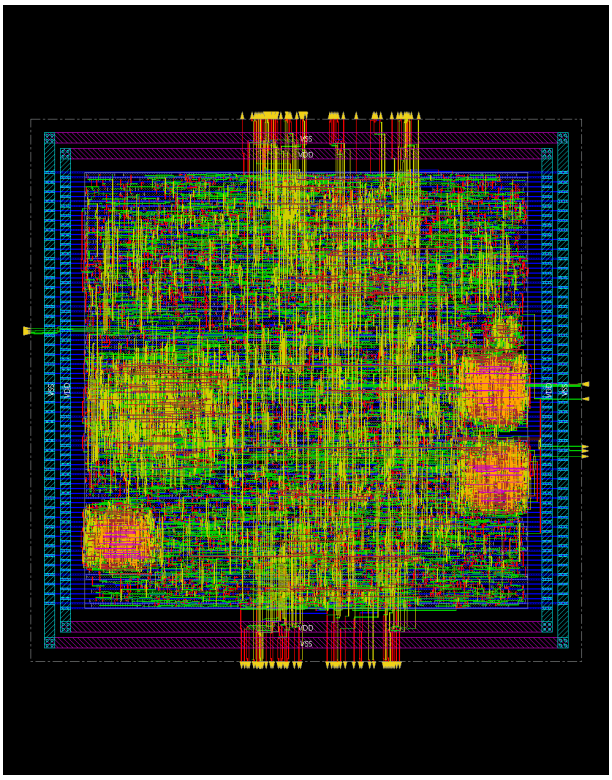


Figure 5.6.: Radix-4 Physical Layout

5.4. Post Layout Verification

Figure 5.7 is a screen shot from the post layout verification in Questa Sim. Simulations were performed with a more realistic clock and no skew could be observed. At the bottom of Figure 5.7 multiple warnings can be observed. This is due to unconnected (unused) output port, for example $\bar{Q}$ in the D-Flip Flips, Carry outputs (c_o) in Add/Subtract units of Iteration Stages, etc. Refer to Appendix A for details. This issue can be addressed simply by using custom components in the schematic design phase of implementation.

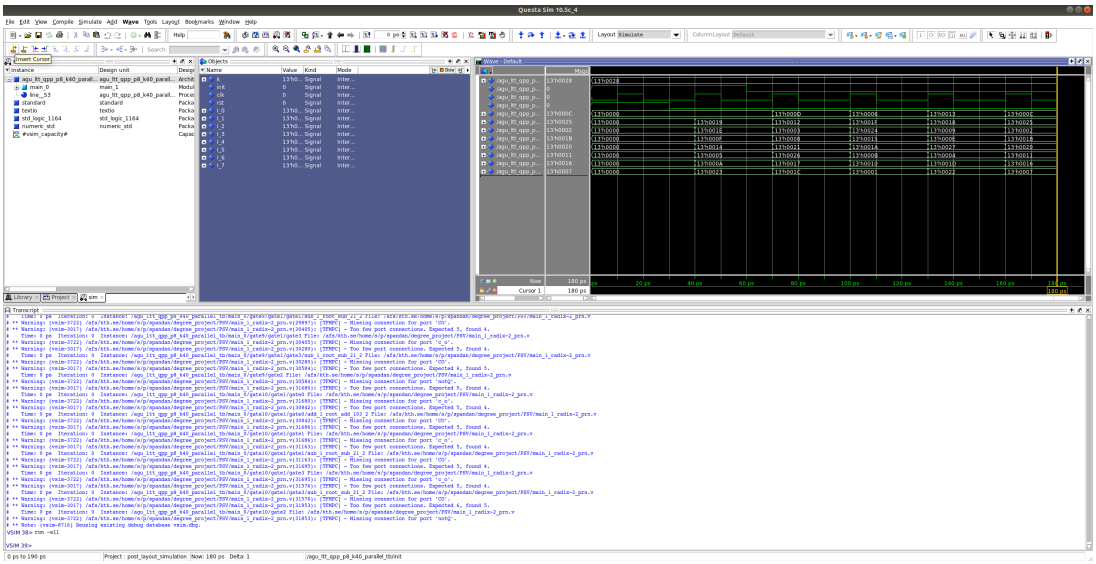


Figure 5.7.: Radix-4 Schematic exported from Questa Sim

5.5. Discussions

Comparing the Radix-2 and Radix-4 implementations it can be clearly seen that Radix-2 has significant advantage with respect to area and power, inspite the lookup table being significantly larger in Radix-2. This shows the optimization capabilities of EDA tools.

5.5.1. Parallelization

Discussed in Section 3.2.2.3, an important property of 3GPP LTE QPP interleaver is its contention free properties. This facilitates parallelization where each decoder can work on smaller window sizes. 3GPP LTE QPP interleavers are parallelizable upto eight across all block sizes and even more when considering larger block sizes (See Equation 3.12). But, with increasing parallelism the size and complexity of the coefficient ROM also increases. This is by far the one of the important trade-off while doing design space exploration.

Significant amount of effort during implementation was spent in exploring several parallelization architectures. Finally two primary approaches were investigated and were presented in Sections 4.3.1.4 and 4.3.1.5.

Section 4.3.1.4 is an aggregation seven serial interleavers which shares a common coefficient ROM. The major benefit of this architecture is that each serial interleavers can asynchronously handle different block sizes, allowing multiple information packets to be simultaneously processed. Also, by sharing a common coefficient ROM, the memory requirement of this design is low. This design performs well when processing information of different sizes. On the other hand the major drawback is that, for large block sizes this design performs poorly as essentially each one of them are a serial interleaver. It is this poor performance with large block sizes that lead to the second design.

Section 4.3.1.5 addresses the above issue and can process a single block of information eight times faster than the previous design. A important trade-off of this design is the size of coefficient ROM which is more than twice the size of the previous design (3055 Bytes instead of 1222 Bytes).

5.5.2. Memory Organization

This is by far one of the biggest design decision that must be taken before working further with these interleavers. Two types of memory organization for input data discussed in literature for these parallel interleavers are (1) Row major[43], and (2) Column major[44].

Depending on memory organization scheme, cost of handling bit addresses could vary significantly. For example, in case of Row major it is easier to fill the input data buffers, but is computationally expensive to read interleaved addresses. Whereas in case of Column major, it takes longer to fill the input buffer but is extremely easy to read interleaved addresses (split upper and lower part of addresses to read row and column addresses, respectively).

Apart from these two, there are also other schemes for moving data, for example, [35] proposes using a Beneš network for interleaving. [34] proposes a bypass mechanism where one hardwired interleaver is reused for interleaving different block sizes by using a mapping scheme. Each such approach comes with their trade-offs depending on which a memory architecture needs to be chosen.

5.5.3. Partitioning in SiLago

Another major architectural decision is the overall architecture of the encoder-decoder units, partitioning and placement of its individual components (input/output buffers, encoder/decoder, interleavers/de-interleavers, etc). Figure 5.8 [34] depicts some of the design aspects and design tradeoffs while designing encoder/decoders. The main areas of differentiation in design lies in area, energy, latency and throughput.

For example, to optimise area and increase throughput the concepts of Sections 4.3.1.4 and 4.3.1.5 can be easily integrated to form a solution. Figure 5.9 shows an architectural overview of the proposed design where one SiLago block can be designed to hold the coefficient ROM and the control logic whereas the surrounding blocks hold the interleaver iteration stages. This structure can be further expanded to accommodate decoder as well.

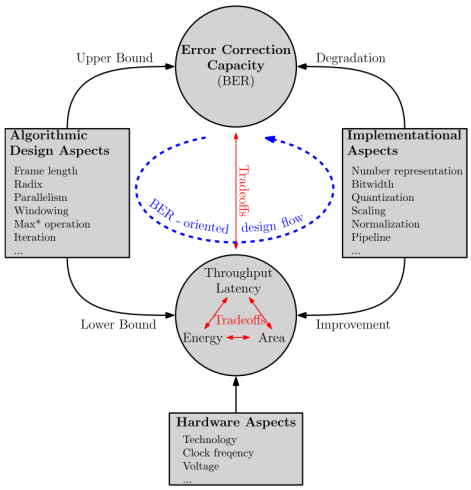


Figure 5.8.: Design Aspects and trade-offs[2]

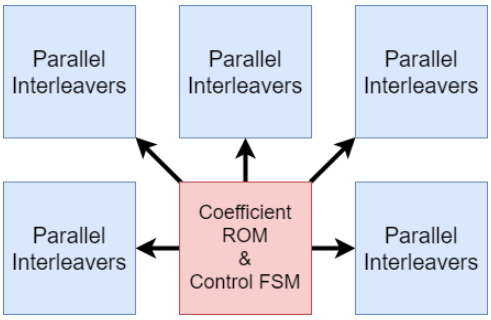


Figure 5.9.: Unified Architecture

5.5.4. SiLago Advantage

An important observation was the differences in area and power at different stages of EDA flow. This is a common scenario and arises due to the abstract nature of EDA flow, that is at earlier stages design constraints are not known, although such issues can be mitigated by using complex models and constraints.

On the contrary SiLago does face a similar issue. As each block is hardened, timing and DRC clear, a very accurate estimation of area and power can be obtained at an earlier stage of EDA flow. This early estimation can be instrumental in facilitating time and energy efficient design space exploration as it offers a close to reality model for consideration.

[2] extracted from [34]

Chapter 6. Conclusions

This chapter discusses the outcome of this thesis and is organized into four sections. Section 6.1 presents the major contribution of this thesis work. Section 6.2 discusses some of the limitations of the designed AGU. Section 6.3 lists some the of future developments that can carried out based on this work. Section 6.4 takes a step back and reflects on this thesis from a social relevance perspective.

6.1. Contributions

The primary contribution of this thesis work is:

- **Implementation of Radix-2 3GPP LTE QPP interleaver**
- **Implementation of Radix-4 3GPP LTE QPP interleaver**

Apart from the two above mentioned major contributions, this thesis also highlights:

- Challenges faced in interleaver implementations. For example, coefficient generation for parallel operations, etc.
- Design approach required to integrate implemented interleavers with decoders.
- Bit level data movement. Concepts of aggregation and transport using FIFO buffers.
- A highly automated design workflow for VLSI design.

Also, while working with Digital, a bug was identified in case of large test vectors. The bug was discussed with the maintainer of the open source software and would be fixed in future releases of the application.

6.2. Limitations

One of the major limitation of the current implementation is that it does not include memory organization. Memory organization depends on the encoding-decoding scheme used, hence this was intentionally left out.

The current Radix-2 implementation is only optimized for parallel-8 operations and does not incorporate hardware reuse features for serial and for parallelism of 2 and 4. In other words, for serial and parallelism of 2 and 4, the interleaver is underutilized. Optimizations offered from the EDA tools were also not explored.

6.3. Future Work

Being a critical part of encoding-decoding scheme in communications, this thesis paves the way for many future developments. The following points discuss some possible areas of future work.

- **Memory organization**
  This thesis focused primarily on an optimized implementation of a generic 3GPP QPP interleaver. Depending on decoding scheme different memory organizations are needed. This is by far one of the major missing piece when it comes to implementing the 3GPP QPP decoder.
- **Coefficient generation**
  To avoid complex calculations, the coefficients required for parallel address generation are currently stored in a ROM. This consumes both area and energy. A thorough investigation on this can result in area and energy savings.
- **Error handling**
  The current implementation does not include error handling features. This was left out mainly to keep the interleaver design simple to fulfil area and energy constraints of SiLago cells.
- **Compare various implementation**
  This work made two implementations of 3GPP QPP interleaver, (Radix-2 with parallelism of 1, 2, 4 or 8, and Radix-4 with parallelism on 4). There also exists other ways of implementing similar and/or more parallel architectures which can be compared in area, energy, throughput, latency, etc.

6.4. Reflections

In the opinion of the author, human kind is undergoing a major transformation primarily fuelled by telecommunication. Being able to contribute in this transformation and understanding the complexities and efforts put in realizing this transformation in itself is the most important take away from this thesis work.

Apart from human communication, machine-to-machine communication is bringing about a socio-economic transformation in the world of manufacturing, partly fuelled by Industry 4.0 (the fourth industrial revolution). The major catalyst for this transformation is "connectivity and interactions among parts, machines, and humans"[45].

The author is glad that the results from this work can be used to fulfil such requirements and be a part in shaping the future.

6.5. Credits

This thesis was conducted at Electronics and Embedded Systems (ESY) department of KTH Royal Institute of Technology in Stockholm under the supervision of Dimitrios Stathis and the guidance of Prof. Dr. Ahmed Hemani. The author is extremely grateful for the supervision, guidance and also administrative support received from KTH.

Appendix A.

A Schematics

A. Circuit diagram of Individual Components

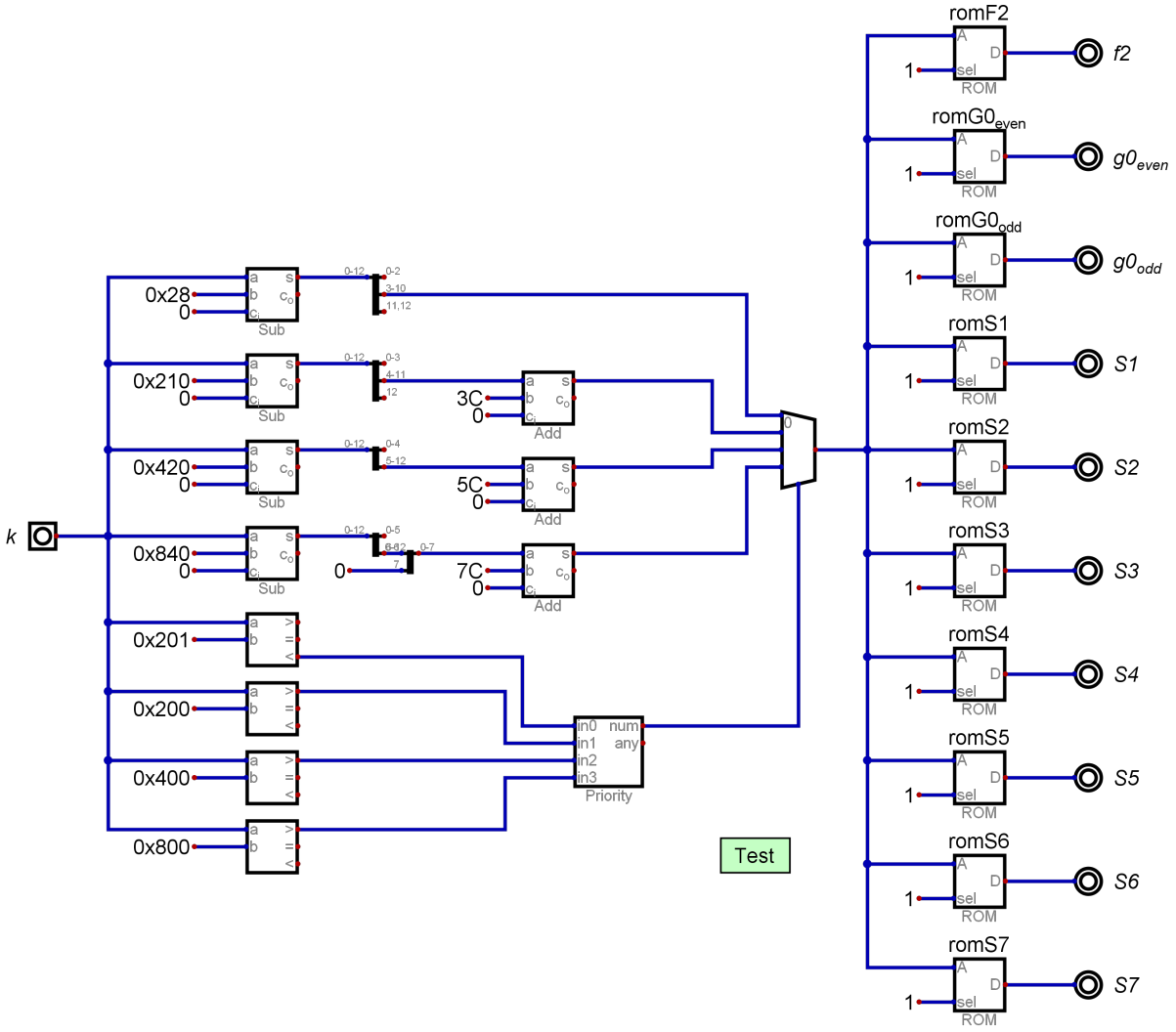


Figure A.1.: Radix2 Parallel 3GPP QPP Interleaver Coefficient Generator

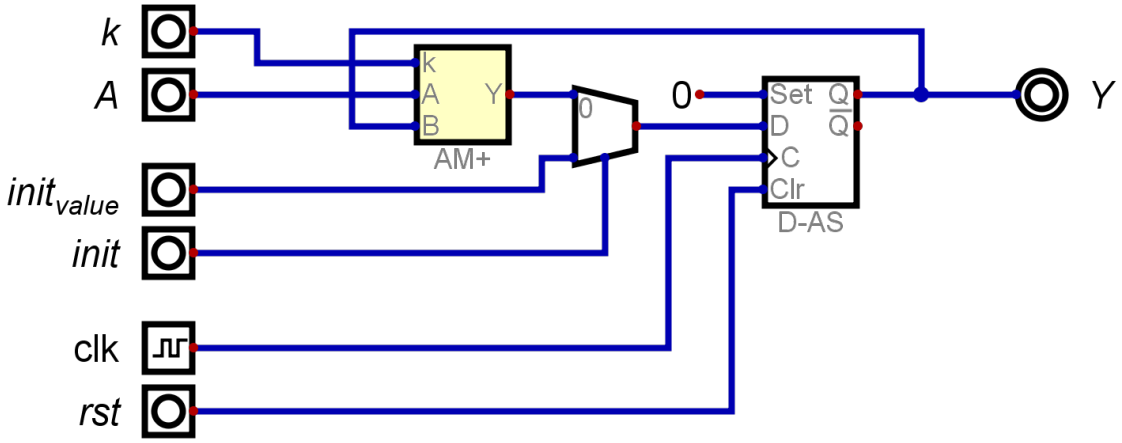


Figure A.2.: Radix2 Parallel 3GPP QPP Interleaver One Iteration Stage

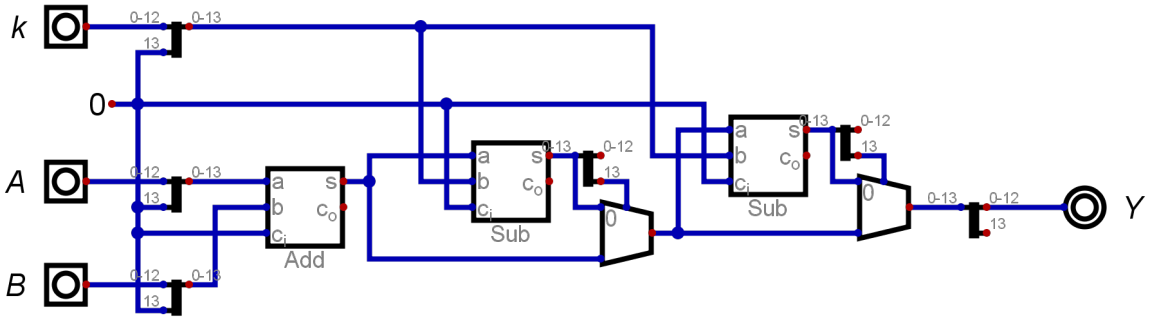


Figure A.3.: Radix2 Parallel 3GPP QPP Interleaver Add-Modulo-Plus Units

B. Schematics Exported from Questa Sim

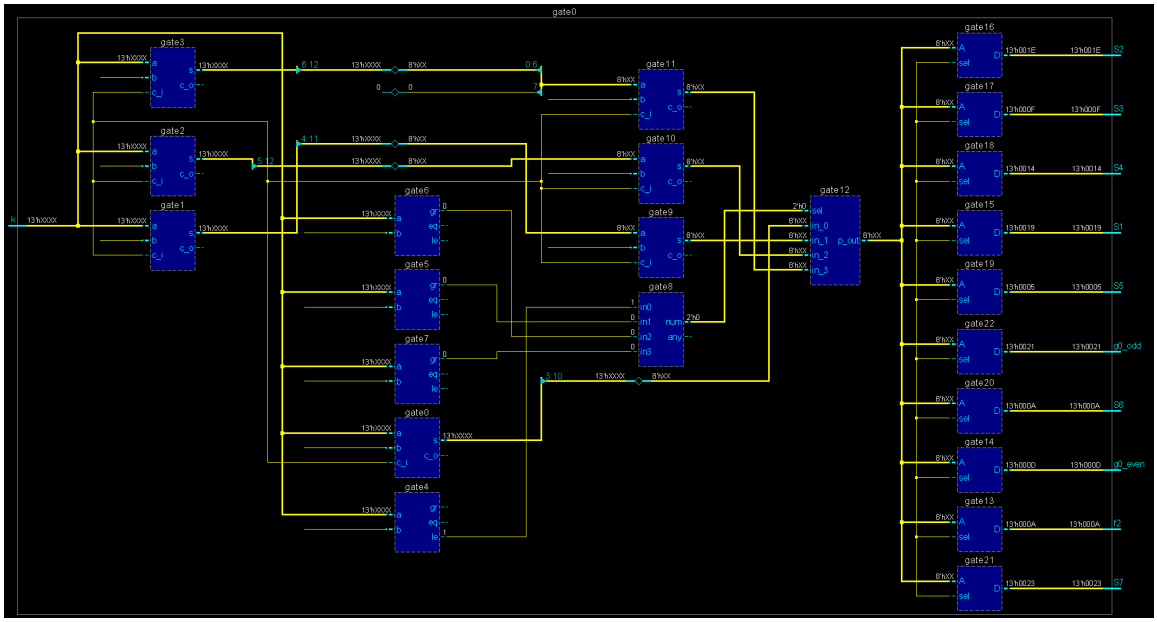


Figure A.4.: Questa Sim schematics of Radix2 Parallel 3GPP QPP Interleaver Coefficient Generator

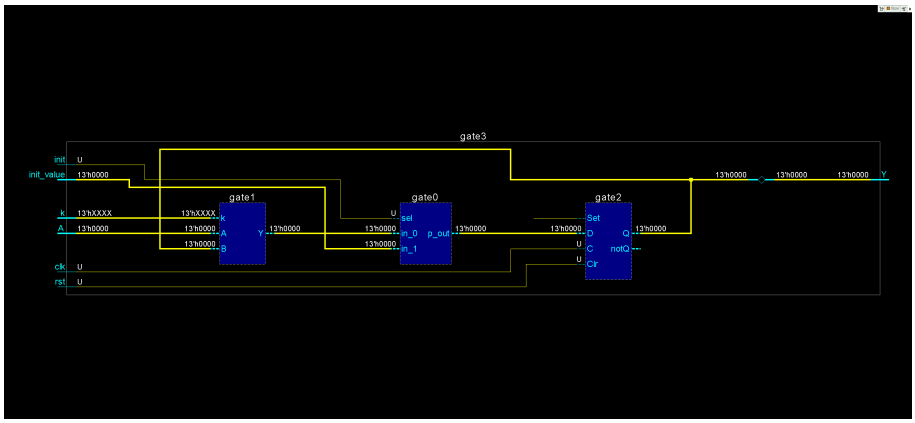


Figure A.5.: Questa Sim schematics of Radix2 Parallel 3GPP QPP Interleaver One Iteration Stage

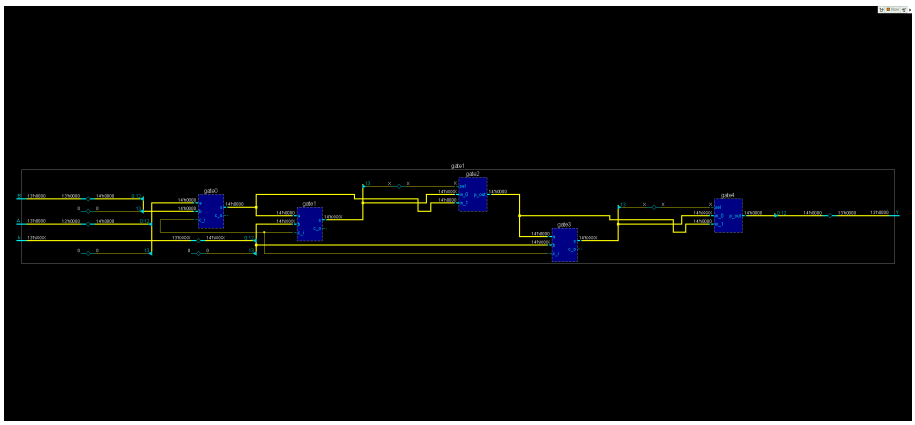


Figure A.6.: Questa Sim schematics of Radix2 Parallel 3GPP QPP Interleaver Add Modulo Plus Units

B Appendix B.
Python Scripts

A. Radix2 P8 Test Vector

```
#!/usr/bin/python3

# Open saved file
from openpyxl import load_workbook
wb = load_workbook("LTT_QPP.xlsx")

# Select sheet "Tabelle"
ws = wb['Tabelle']

k = ws['B']
f1 = ws['C']
f2 = ws['D']
P = 8

filename = "test_vector_for_parallel8.txt"
text_file = open(filename, "w")
print("#test vector for parallel 8 ", file=text_file)
print("clk\trst\tk\tinit\tI_1\tI_2\tI_3\tI_4\tI_5\tI_6\tI_7\tI_8", file=text_file)

for index in range(1, len(k)):
    block_size = k[index].value
    print("For block size: {0}".format(block_size))
    # Reset
    print("C\t1\t{0}\t0\t0\t0\t0\t0\t0\t0\t0".format(block_size), file=text_file)
    print("C\t0\t{0}\t1\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}"
          .format(block_size,
                  int(((f1[index].value * (block_size / P * 0)) + (
                              f2[index].value * (block_size / P * 0) * (block_size / P * 0))) %
                              ↪  block_size),
                  int(((f1[index].value * (block_size / P * 1)) + (
                              f2[index].value * (block_size / P * 1) * (block_size / P * 1))) %
                              ↪  block_size),
                  int(((f1[index].value * (block_size / P * 2)) + (
                              f2[index].value * (block_size / P * 2) * (block_size / P * 2))) %
                              ↪  block_size),
                  int(((f1[index].value * (block_size / P * 3)) + (
                              f2[index].value * (block_size / P * 3) * (block_size / P * 3))) %
                              ↪  block_size),
                  int(((f1[index].value * (block_size / P * 4)) + (
                              f2[index].value * (block_size / P * 4) * (block_size / P * 4))) %
                              ↪  block_size),
                  int(((f1[index].value * (block_size / P * 5)) + (
```

```
                                f2[index].value * (block_size / P * 5) * (block_size / P * 5))) %
                                ↪  block_size),
                        int(((f1[index].value * (block_size / P * 6)) + (
                                f2[index].value * (block_size / P * 6) * (block_size / P * 6))) %
                                ↪  block_size),
                        int(((f1[index].value * (block_size / P * 7)) + (
                                f2[index].value * (block_size / P * 7) * (block_size / P * 7))) %
                                ↪  block_size)),
                  file=text_file)

    for iteration in range(1, int(block_size/P)):
        # print("value of bit {0}".format(iteration))
        print("C\t0\t{0}\t0\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}"
              .format(block_size,
                      int(((f1[index].value * ((block_size / P * 0) + iteration)) +
                           (f2[index].value * ((block_size / P * 0) + iteration) *
                                               ((block_size / P * 0) + iteration))) % block_size),

                      int(((f1[index].value * ((block_size / P * 1) + iteration)) +
                           (f2[index].value * ((block_size / P * 1) + iteration) *
                                               ((block_size / P * 1) + iteration))) % block_size),

                      int(((f1[index].value * ((block_size / P * 2) + iteration)) +
                           (f2[index].value * ((block_size / P * 2) + iteration) *
                                               ((block_size / P * 2) + iteration))) % block_size),

                      int(((f1[index].value * ((block_size / P * 3) + iteration)) +
                           (f2[index].value * ((block_size / P * 3) + iteration) *
                                               ((block_size / P * 3) + iteration))) % block_size),

                      int(((f1[index].value * ((block_size / P * 4) + iteration)) +
                           (f2[index].value * ((block_size / P * 4) + iteration) *
                                               ((block_size / P * 4) + iteration))) % block_size),

                      int(((f1[index].value * ((block_size / P * 5) + iteration)) +
                           (f2[index].value * ((block_size / P * 5) + iteration) *
                                               ((block_size / P * 5) + iteration))) % block_size),

                      int(((f1[index].value * ((block_size / P * 6) + iteration)) +
                           (f2[index].value * ((block_size / P * 6) + iteration) *
                                               ((block_size / P * 6) + iteration))) % block_size),

                      int(((f1[index].value * ((block_size / P * 7) + iteration)) +
                           (f2[index].value * ((block_size / P * 7) + iteration) *
                                               ((block_size / P * 7) + iteration))) % block_size)),
              file=text_file)

text_file.close()
```

B. Radix4 P4 Test Vector

```
#!/usr/bin/python3

# Open saved file
from openpyxl import load_workbook
wb = load_workbook("LTT_QPP.xlsx")

# Select sheet "Tabelle"
ws = wb['Tabelle']

k = ws['B']
f1 = ws['C']
f2 = ws['D']
P = 4

filename = "test_vector_for_parallel_radix4.txt"
text_file = open(filename, "w")
print("#test vector for parallel radix 4 ", file=text_file)
print("clk\trst\tk\tinit\tI_1\tI_2\tI_3\tI_4\tI_5\tI_6\tI_7\tI_8", file=text_file)

for index in range(1, len(k)):
    block_size = k[index].value
    print("For block size: {0}".format(block_size))
    # Reset
    print("C\t1\t{0}\t0\t0\t0\t0\t0\t0\t0\t0".format(block_size), file=text_file)
    print("C\t0\t{0}\t1\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}"
          .format(block_size,
                  int(((f1[index].value * (block_size / P * 0)) + (
                        f2[index].value * (block_size / P * 0) * (block_size / P * 0))) % block_size),
                  int(((f1[index].value * ((block_size / P * 0) + 1)) + (
                        f2[index].value * ((block_size / P * 0) + 1) * ((block_size / P * 0) + 1))) %
                        ↪  block_size),

                  int(((f1[index].value * (block_size / P * 1)) + (
                        f2[index].value * (block_size / P * 1) * (block_size / P * 1))) % block_size),
                  int(((f1[index].value * ((block_size / P * 1) + 1)) + (
                        f2[index].value * ((block_size / P * 1) + 1) * ((block_size / P * 1) + 1))) %
                        ↪  block_size),

                  int(((f1[index].value * (block_size / P * 2)) + (
                        f2[index].value * (block_size / P * 2) * (block_size / P * 2))) % block_size),
                  int(((f1[index].value * ((block_size / P * 2) + 1)) + (
                        f2[index].value * ((block_size / P * 2) + 1) * ((block_size / P * 2) + 1))) %
                        ↪  block_size),

                  int(((f1[index].value * (block_size / P * 3)) + (
                        f2[index].value * (block_size / P * 3) * (block_size / P * 3))) % block_size),
                  int(((f1[index].value * ((block_size / P * 3) + 1)) + (
                        f2[index].value * ((block_size / P * 3) + 1) * ((block_size / P * 3) + 1))) %
                        ↪  block_size)),

                  file=text_file)

    for iteration in range(1, int(block_size/P)):
        # print("value of bit {0}".format(iteration))
        print("C\t0\t{0}\t0\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}"
              .format(block_size,
                      int(((f1[index].value * ((block_size / P * 0) + (2 * iteration))) +
                           (f2[index].value * ((block_size / P * 0) + (2 * iteration)) *
```

```
55                                          ((block_size / P * 0) + (2 * iteration)))) % block_size),
56
57                      int(((f1[index].value * ((block_size / P * 0) + 1 + (2 * iteration))) +
58                           (f2[index].value * ((block_size / P * 0) + 1 + (2 * iteration)) *
59                                          ((block_size / P * 0) + 1 + (2 * iteration)))) %
                                            ↪  block_size),
60
61                      int(((f1[index].value * ((block_size / P * 1) + (2 * iteration))) +
62                           (f2[index].value * ((block_size / P * 1) + (2 * iteration)) *
63                                          ((block_size / P * 1) + (2 * iteration)))) % block_size),
64
65                      int(((f1[index].value * ((block_size / P * 1) + 1 + (2 * iteration))) +
66                           (f2[index].value * ((block_size / P * 1) + 1 + (2 * iteration)) *
67                                          ((block_size / P * 1) + 1 + (2 * iteration)))) %
                                            ↪  block_size),
68
69                      int(((f1[index].value * ((block_size / P * 2) + (2 * iteration))) +
70                           (f2[index].value * ((block_size / P * 2) + (2 * iteration)) *
71                                          ((block_size / P * 2) + (2 * iteration)))) % block_size),
72
73                      int(((f1[index].value * ((block_size / P * 2) + 1 + (2 * iteration))) +
74                           (f2[index].value * ((block_size / P * 2) + 1 + (2 * iteration)) *
75                                          ((block_size / P * 2) + 1 + (2 * iteration)))) %
                                            ↪  block_size),
76
77                      int(((f1[index].value * ((block_size / P * 3) + (2 * iteration))) +
78                           (f2[index].value * ((block_size / P * 3) + (2 * iteration)) *
79                                          ((block_size / P * 3) + (2 * iteration)))) % block_size),
80
81                      int(((f1[index].value * ((block_size / P * 3) + 1 + (2 * iteration))) +
82                           (f2[index].value * ((block_size / P * 3) + 1 + (2 * iteration)) *
83                                          ((block_size / P * 3) + 1 + (2 * iteration)))) %
                                            ↪  block_size)),
84
85              file=text_file)
86
87  text_file.close()
```

Appendix C.

TCL Scripts

A. Design Vision

```
set designer "Spandan Dey"
set company "KTH"
set SynopsysHome "/synopsys/syn_M-2016.12"
set outfname "AGU_LTT_QPP_Radix2_P8"
set version "1.1"
####### Set Directary #########
set SYNDIR ../exe
set OUTDIR ../output
set SYNDB  ../exe
set RPTDIR ../reports
######Enviroment############
define_design_lib work -path $SYNDIR/work
set write_name_nets_same_as_ports "true"

# Define the libraries and search path
#set target_library
set search_path "$search_path /synopsys/syn_M-2016.12/libraries/syn"
set synlib "/synopsys/syn_M-2016.12/libraries/syn/dw_foundation.sldb"
set link_path "$search_path"
set synthetic_library "$synthetic_library $synlib /synopsys/syn_M-2016.12/libraries/syn/standard.sldb"
set link_library "* $target_library $synthetic_library"

define_design_lib WORK -path ./SYN/WORK
set hdlin_use_syn_shifters true
set_clock_gating_style -sequential_cell latch

######Read Design#############
read_file -format vhdl {"/VHDL/AGU_LTT_QPP_Radix2_P8.vhdl"}
elaborate main -architecture Behavioral -library DEFAULT

######Timing Exceptions#######
#set_load_unit -picofarads
set_operating_conditions ss0p855v125c
set_wire_load_mode segmented
set_false_path -from [get_port rst]

########Set Clock#############
create_clock -name "clk" -period 5 -waveform { 0 2.5  }  { clk }

set_load -pin_load 0.001 [all_outputs]
set_input_delay -clock clk  0.1  [remove_from_collection [all_inputs] clk]
set_output_delay -clock clk  0.1  [all_outputs]
```

```
set_driving_cell -lib_cell BUFFD6BWP30P140 [remove_from_collection [all_inputs] clk]
set_clock_uncertainty  0.1  [get_clocks clk]
#######Compile Option############
compile -map_effort high -exact_map -boundary_optimization -gate_clock
#######Report####################
report_timing > $RPTDIR/timing_$outfname$version.rpt
report_area > $RPTDIR/area_$outfname$version.rpt
report_power -analysis_effort medium > $RPTDIR/power_$outfname$version.rpt
#######Result####################
#write -format db -hier -o $SYNDB/$outfname$version.db # no longer supported
write -format verilog -hier -o $OUTDIR/$outfname$version.v
write_sdf -version 2.1 $OUTDIR/$outfname$version.sdf
write_sdc $OUTDIR/$outfname$version.sdc
write -hierarchy -format ddc -output $OUTDIR/$outfname$version.ddc
```

B. Innovus

Initialize Design Scripts

```
set conf_qxconf_file {NULL}
set conf_qxlib_file {NULL}
set defHierChar {/}
set init_gnd_net {VSS}
set init_lef_file $LEF_FILE
set init_mmmc_file $MMMMC_FILE
set init_pwr_net {VDD}
set init_top_cell $TOP_NAME
set init_verilog $NETLIST_FILE
set lsgOCPGainMult 1.000000

# do not unify the design (we want to have master clone partitions)
set init_design_uniquify 1
init_design
setPreference MinFPModuleSize 0
getIoFlowFlag
setIoFlowFlag 0
setDrawView fplan
fit

#Set Process node
setDesignMode -process 28

#Set Global nets
clearGlobalNets
globalNetConnect VDD -type pgpin -pin VDD -inst * -module {}
globalNetConnect VSS -type pgpin -pin VSS -inst * -module {}

setAnalysisMode -analysisType onChipVariation -cppr both
saveDesign ${save_dir}/${out_name}_${version}_loaded.enc -verilog -def
```

Place and Route Scripts

```
#place design + prelimenar clock tree
setPlaceMode -reset
setPlaceMode -congEffort auto -timingDriven 1 -clkGateAware 1 -powerDriven 0 -ignoreScan 1 -reorderScan 1 -ignoreSpare 0 -placeIOPins 1 -moduleAwareSpare 0 -preserveRouting 0 -rmAffectedRouting 0 -checkRoute 0 -swapEEQ 0 -place_global_fast_cts true -place_global_uniform_density true

add_ndr -width {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.1 M7 0.1 M8 0.8 AP 4.0 } -spacing {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.1 M7 0.1 M8 0.8 AP 4.0 } -name CTS_2W2S
add_ndr -width {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.1 M7 0.1 M8 0.8 AP 4.0 } -spacing {M1 0.05 M2 0.05 M3 0.05 M4 0.05 M5 0.05 M6 0.05 M7 0.05 M8 0.4 AP 2.0 } -name CTS_2W1S

# New route types
create_route_type -name CTS_route_top_trunk -non_default_rule CTS_2W2S -top_preferred_layer M4 -bottom_preferred_layer M3 -shield_net VSS -preferred_routing_layer_effort high
create_route_type -name CTS_route_leaves -non_default_rule CTS_2W1S -top_preferred_layer M4 -bottom_preferred_layer M3 -preferred_routing_layer_effort high

set_ccopt_property -net_type leaf -route_type CTS_route_leaves
set_ccopt_property -net_type trunk -route_type CTS_route_top_trunk
set_ccopt_property -net_type top -route_type CTS_route_top_trunk

set_ccopt_property -cts_buffer_cells {CKBD0BWP30P140 CKBD12BWP30P140 CKBD16BWP30P140 CKBD1BWP30P140 CKBD20BWP30P140 CKBD24BWP30P140 CKBD2BWP30P140 CKBD3BWP30P140 CKBD4BWP30P140 CKBD6BWP30P140 CKBD8BWP30P140 DCCKBD12BWP30P140 DCCKBD16BWP30P140 DCCKBD20BWP30P140 DCCKBD4BWP30P140 DCCKBD8BWP30P140}
set_ccopt_property -cts_inverter_cells {CKND0BWP30P140 CKND12BWP30P140 CKND16BWP30P140 CKND1BWP30P140 CKND20BWP30P140 CKND24BWP30P140 CKND2BWP30P140 CKND3BWP30P140 CKND4BWP30P140 CKND6BWP30P140 CKND8BWP30P140 DCCKND12BWP30P140 DCCKND16BWP30P140 DCCKND20BWP30P140 DCCKND4BWP30P140 DCCKND8BWP30P140}
set_ccopt_property -clock_gating_cells {CKLNQD12BWP30P140 CKLNQD16BWP30P140 CKLNQD1BWP30P140 CKLNQD20BWP30P140 CKLNQD24BWP30P140 CKLNQD2BWP30P140 CKLNQD3BWP30P140 CKLNQD4BWP30P140 CKLNQD6BWP30P140 CKLNQD8BWP30P140 CKLNQOPTMAD16BWP30P140 CKLNQOPTMAD1BWP30P140 CKLNQOPTMAD2BWP30P140 CKLNQOPTMAD4BWP30P140 CKLNQOPTMAD8BWP30P140 \
                                                                                                    CKLHQD12BWP30P140 CKLHQD16BWP30P140 CKLHQD1BWP30P140 CKLHQD20BWP30P140 CKLHQD24BWP30P140 CKLHQD2BWP30P140 CKLHQD3BWP30P140 CKLHQD4BWP30P140 CKLHQD6BWP30P140 CKLHQD8BWP30P140}
set_ccopt_property use_inverters auto
set_ccopt_property target_skew 50ps
set_ccopt_property target_max_trans 130ps
create_ccopt_clock_tree_spec -file ccopt_place.spec
source ccopt_place.spec
setOptMode -reset
setOptMode -powerEffort none -effort high -usefulSkew true -usefulSkewPreCTS true -usefulSkewPostRoute true -usefulSkewCCOpt standard
place_opt_design

#optDesign
setOptMode -reset
setOptMode -powerEffort none -effort high -usefulSkew true -usefulSkewCCOpt medium
optDesign -preCTS

```

```
#save design
saveDesign ${save_dir}/${out_name}_${version}_preCTS.enc -verilog -def
```

Clock Tree Synthesis Scripts

```
# Set Max Routing layer
setNanoRouteMode -quiet -routeTopRoutingLayer 7

set_ccopt_property -net_type leaf -route_type CTS_route_leaves
set_ccopt_property -net_type trunk -route_type CTS_route_top_trunk
set_ccopt_property -net_type top -route_type CTS_route_top_trunk
set_ccopt_property -cts_buffer_cells {CKBD0BWP30P140 CKBD12BWP30P140 CKBD16BWP30P140 CKBD1BWP30P140
    ↪  CKBD20BWP30P140 CKBD24BWP30P140 CKBD2BWP30P140 CKBD3BWP30P140 CKBD4BWP30P140 CKBD6BWP30P140
    ↪  CKBD8BWP30P140 DCCKBD12BWP30P140 DCCKBD16BWP30P140 DCCKBD20BWP30P140 DCCKBD4BWP30P140
    ↪  DCCKBD8BWP30P140}
set_ccopt_property -cts_inverter_cells {CKND0BWP30P140 CKND12BWP30P140 CKND16BWP30P140 CKND1BWP30P140
    ↪  CKND20BWP30P140 CKND24BWP30P140 CKND2BWP30P140 CKND3BWP30P140 CKND4BWP30P140 CKND6BWP30P140
    ↪  CKND8BWP30P140 DCCKND12BWP30P140 DCCKND16BWP30P140 DCCKND20BWP30P140 DCCKND4BWP30P140
    ↪  DCCKND8BWP30P140}
set_ccopt_property -clock_gating_cells {CKLNQD12BWP30P140 CKLNQD16BWP30P140 CKLNQD1BWP30P140
    ↪  CKLNQD20BWP30P140 CKLNQD24BWP30P140 CKLNQD2BWP30P140 CKLNQD3BWP30P140 CKLNQD4BWP30P140
    ↪  CKLNQD6BWP30P140 CKLNQD8BWP30P140 CKLNQOPTMAD16BWP30P140 CKLNQOPTMAD1BWP30P140 CKLNQOPTMAD2BWP30P140
    ↪  CKLNQOPTMAD4BWP30P140 CKLNQOPTMAD8BWP30P140 \
                                                                    CKLHQD12BWP30P140
                                                                    ↪  CKLHQD16BWP30P140
                                                                    ↪  CKLHQD1BWP30P140
                                                                    ↪  CKLHQD20BWP30P140
                                                                    ↪  CKLHQD24BWP30P140
                                                                    ↪  CKLHQD2BWP30P140
                                                                    ↪  CKLHQD3BWP30P140
                                                                    ↪  CKLHQD4BWP30P140
                                                                    ↪  CKLHQD6BWP30P140
                                                                    ↪  CKLHQD8BWP30P140}
set_ccopt_property use_inverters auto
set_ccopt_property target_skew 180ps
set_ccopt_property target_max_trans 130ps
create_ccopt_clock_tree_spec -file ccopt.spec
source ccopt.spec
ccopt_design
saveDesign ${save_dir}/${out_name}_${version}_postCTS.enc -verilog -def
```

Post Clock Tree Synthesis Optimizations Scripts

```
setAnalysisMode -cppr both -clockGatingCheck 1 -timeBorrowing 1 \
-useOutputPinCap 1 -sequentialConstProp 1 -timingSelfLoopsNoSkew 0 \
-enableMultipleDriveNet 1 -clkSrcPath 1 -warn 1 -usefulSkew 1 \
-analysisType onChipVariation -skew true -clockPropagation sdcControl -log 1
update_analysis_view -name AV_WC_RCWORST -constraint_mode CM_POST
update_analysis_view -name AV_BC_RCBEST  -constraint_mode CM_POST
update_analysis_view -name AV_TC_RCTYP   -constraint_mode CM_POST
```

```
set_analysis_view -update_timing
set_interactive_constraint_modes {CM_POST}
set_propagated_clock [all_clocks]
```

Final Route Scripts

```
#Detail route design
setNanoRouteMode -quiet -timingEngine {}
setNanoRouteMode -quiet -routeWithTimingDriven 1
setNanoRouteMode -quiet -routeWithSiPostRouteFix 0
setNanoRouteMode -quiet -drouteStartIteration default
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven true
setNanoRouteMode -quiet -routeWithSiDriven false
routeDesign -globalDetail
saveDesign ${save_dir}/${out_name}_${version}_postROUTE.enc -verilog -def
```

Save Design Scripts

```
saveDesign ${save_dir}/${out_name}_${version}_prn.enc -verilog -def

saveNetlist ${output_dir}/${out_name}_${version}_prn.v
streamOut ${output_dir}/${out_name}_${version}_prn.gds -mapFile streamOut.map -libName DesignLib -units
↪  2000 -mode ALL -uniquifyCellNames -outputMacros
global dbgLefDefOutVersion
set dbgLefDefOutVersion 5.8
defOut -floorplan -netlist -routing -withShield -allLayers ${output_dir}/${out_name}_${version}_prn.def
set dbgLefDefOutVersion 5.8
extractRC
all_hold_analysis_views
all_setup_analysis_views
write_sdf -ideal_clock_network ${output_dir}/${out_name}_${version}_prn.sdf -abstracted_model
↪  -recompute_parallel_arcs -recompute_delay_calc -typ_view AV_TC_RCTYP -max_view AV_WC_RCWORST
↪  -min_view AV_BC_RCBEST

# new command
write_lef_abstract ${output_dir}/${out_name}_${version}_prn.lef -5.8 -portForEachStripePin -stripePin
↪  -PGpinLayers {7 8} -extractBlockPGPinLayers {7 8}
```

C. Physical Design Flow in Innovus

- Create folder structure - "dc", "exe", "save", "output", "reports", "scripts"
- Copy all files (sdc, ddc, v) from logic synthesis output to "dc" folder
- Copy *.sdc file and rename *_post.sdc
- Modify clock uncertainty in *_post.sdc to "0.01"
- Copy all scripts to "scripts" folder
- Modify "global_variables.tcl"
  - TOP_NAME -> main entity
  - Set version -> to required version
  - Set NETLIST_FILE -> point to *.v in dc
  - Set SDC_FILES -> point to *.sdc in dc
  - Set SDC_FILES_POSTCTS -> point to *_post.sdc in dc
- Move to exe directory and open Innovus

  > source ../scripts/global_variables.tcl

  > source ../scripts/myInitDesign.tcl

  > Floorplanning

  > Adding Power Rings

  > Special Routing of Power Grid

  > source ../scripts/palce.tcl

  > delete_ccopt_clock_tree_spec

  > timeDesign -preCTS

  > source ../scripts/ccopt_2W2S.tcl

  > source ../scripts/update_constraint_POSTCTS.tcl

  > timeDesign -postCTS

  > optDesign -postCTS #if there is some slack

  > source ../scripts/route.tcl

  > timeDesign -postRoute

  > optDesign -postRoute #if there is some slack

  > source ../scripts/save_output.tcl

  > report_area » ../reports/area.rpt

  > report_power » ../reports/power.rpt

  > reportGateCount » ../reports/gateCount.rpt

Bibliography

[1] Alexander Graham Bell. *IMPROVEMENT IN TELEGRAPHY (US174465A)*. 1876.

[2] *NASA Facts: Voyager - NASA-NF-87/10-77*. eng. Tech. rep. 1977. URL: http://hdl.handle.net/2060/19780009180.

[3] Nasim Farahini. *SiLago: Enabling System Level Automation Methodology to Design Custom High-Performance Computing Platforms*. 2016. ISBN: 9789175959009.

[4] Alberto Castella. "Design of a multi-mode interleaver for the SiLago platform". In: (2017).

[5] Rizwan Asghar. *Flexible Interleaving Subsystems for FEC in Baseband Processors*. 1312. 2010, p. 189. ISBN: 9789173933971.

[6] Liisa Lehtiranta *et al.* "The Constructive Research Approach: Problem Solving for Complex Projects". In: *Designs, Methods and Practices for Research of Project Management*. 2015, p. 520. ISBN: ISBN 978-1-4094-4880-8.

[7] C. E. Shannon. "A Mathematical Theory of Communication". In: *SIGMOBILE Mob. Comput. Commun. Rev.* 5.1 (2001), pp. 3–55. ISSN: 1559-1662. DOI: 10.1145/584091.584093. URL: http://doi.acm.org/10.1145/584091.584093.

[8] Claude Berrou. *Error-correction coding method with at least two systematic convolutional codings in parallel, corresponding iterative decoding method, decoding module and decoder*. 1991. URL: https://patents.google.com/patent/US5446747A/en.

[9] C. Berrou, A. Glavieux, and P. Thitimajshima. "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1". In: *Proceedings of ICC '93 - IEEE International Conference on Communications*. Vol. 2. IEEE, pp. 1064–1070. ISBN: 0-7803-0950-2. DOI: 10.1109/ICC.1993.397441. URL: http://ieeexplore.ieee.org/document/397441/.

[10] Fred Daneshgaran and Marina Mondin. "Design of interleavers for turbo codes: iterative interleaver growth algorithms of polynomial complexity". In: *IEEE Transactions on Information Theory* (1999). ISSN: 00189448. DOI: 10.1109/18.782105.

[11] Hamid R. Sadjadpour *et al.* "Interleaver design for turbo codes". In: *IEEE Journal on Selected Areas in Communications* (2001). ISSN: 07338716. DOI: 10.1109/49.924867.

[12] C. Fragouli and R.D. Wesel. "Semi-random interleaver design criteria". In: 2003. DOI: 10.1109/glocom.1999.831723.

[13] Oscar Y. Takeshita and Daniel J. Costello. "New deterministic interleaver designs for turbo codes". In: *IEEE Transactions on Information Theory* (2000). ISSN: 00189448. DOI: 10.1109/18.868474.

[14] Carlos J Corrada-bravo and San Juan. "Deterministic Interleavers for Turbo Codes with Random-like Performance and Simple Implementation". In: (2003), pp. 555–558.

[15] Jing Sun and O.Y. Takeshita. "Interleavers for turbo codes using permutation polynomials over integer rings". In: *IEEE Transactions on Information Theory* 51.1 (2005), pp. 101–119. ISSN: 0018-9448. DOI: 10.1109/TIT.2004.839478. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1377495%20http://ieeexplore.ieee.org/document/1377495/.

[16] 3GPP. "3GPP TS 36.212 V15.3.0 (2018-09); Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding". In: *Technical Report* (2019). URL: https://www.3gpp.org/DynaReport/36212.htm.

[17] Eirik Rosnes and Oscar Y. Takeshita. “Optimum distance quadratic permutation polynomial-based interleavers for turbo codes”. In: *IEEE International Symposium on Information Theory - Proceedings* 2 (2006), pp. 1988–1992. ISSN: 21578101. DOI: 10.1109/ISIT.2006.261897.

[18] B Moision and J Hamkins. “Coded Modulation for the Deep-Space Optical Channel : Serially Concatenated Pulse-Position Modulation”. In: *JPL Interplanetary Network Progress Report* (2005).

[19] G. R. Blakley. “A Computer Algorithm for Calculating the Product AB Modulo M”. In: *IEEE Transactions on Computers* (1983). ISSN: 00189340. DOI: 10.1109/TC.1983.1676262.

[20] B. Classon *et al.* “Contention-free interleavers”. In: *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.* (2004), pp. 52–52. DOI: 10.1109/isit.2004.1365089.

[21] Ajit Nimbalker *et al.* “Contention-free interleavers for high-throughput turbo decoding”. In: *IEEE Transactions on Communications* 56.8 (2008), pp. 1258–1267. ISSN: 00906778. DOI: 10.1109/TCOMM.2008.050502.

[22] T.K. Blankenship, B Classon, and V Desai. “High-throughput turbo decoding techniques for 4g”. In: *Proc. Int. Conf. 30 Wireless and Beyond* (2002), pp. 137–142.

[23] A. Giulietti, L. van der Perre, and M. Strum. “Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements”. In: *Electronics Letters* (2002). ISSN: 00135194. DOI: 10.1049/el:20020148.

[24] Yan-Xiu Zheng and Yu.T. Su. “A new interleaver design and its application to turbo codes”. In: 2003. DOI: 10.1109/vetecf.2002.1040453.

[25] C. Berrou *et al.* “Designing good permutations for turbo codes: towards a single model”. In: 2004. DOI: 10.1109/icc.2004.1312507.

[26] David Gnaedig *et al.* “On multiple slice turbo codes”. In: *Annales Des Télécommunications* 60.1 (2005), pp. 79–102. ISSN: 1958-9395. DOI: 10.1007/BF03219808. URL: https://doi.org/10.1007/BF03219808.

[27] Oscar Y. Takeshita. “On maximum contention-free interleavers and permutation polynomials over integer rings”. In: *IEEE Transactions on Information Theory* 52.3 (2006), pp. 1249–1253. ISSN: 00189448. DOI: 10.1109/TIT.2005.864450.

[28] Stefan Weithoffer *et al.* “25 Years of Turbo Codes: From Mb/s to beyond 100 Gb/s”. In: *10th International Symposium on Turbo Codes \& Iterative Information Processing (ISTC 2018)* (2018). URL: https://hal-imt-atlantique.archives-ouvertes.fr/hal-01869012.

[29] Stefan Weithoffer, Kira Kraft, and Norbert Wehn. “Bit-level pipelining for highly parallel turbo-code decoders: A critical assessment”. In: *2017 IEEE AFRICON: Science, Technology and Innovation for Africa, AFRICON 2017*. 2017. ISBN: 9781538627754. DOI: 10.1109/AFRCON.2017.8095467.

[30] Thomas Ilnseher *et al.* “A 2.15GBit/s turbo code decoder for LTE advanced base station applications”. In: *2012 7th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*. IEEE, 2012, pp. 21–25. ISBN: 978-1-4577-2115-1. DOI: 10.1109/ISTC.2012.6325191. URL: http://ieeexplore.ieee.org/document/6325191/.

[31] G. Wang *et al.* “Parallel Interleaver Design for a High throughput HSPA+/LTE Multi-Standard Turbo Decoder”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.5 (2014), pp. 1376–1389. ISSN: 1549-8328. DOI: 10.1109/TCSI.2014.2309810.

[32] Rahul Shrestha and Roy P. Paily. “High-throughput turbo decoder with parallel architecture for lte wireless communication standards”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* (2014). ISSN: 15498328. DOI: 10.1109/TCSI.2014.2332266.

[33] Robert G. Maunder. “A Fully-Parallel Turbo Decoding Algorithm”. In: *IEEE Transactions on Communications* 63.8 (2015), pp. 2762–2775. ISSN: 0090-6778. DOI: 10.1109/TCOMM.2015.2450208. URL: http://ieeexplore.ieee.org/document/7137638/.

[34] An Li *et al.* “VLSI Implementation of Fully Parallel LTE Turbo Decoders”. In: *IEEE Access* 4 (2016), pp. 323–346. ISSN: 21693536. DOI: 10.1109/ACCESS.2016.2515719.

[35] An Li *et al.* “1.5 Gbit/s FPGA Implementation of a Fully-Parallel Turbo Decoder Designed for Mission-Critical Machine-Type Communication Applications”. In: *IEEE Access* 4 (2016), pp. 5452–5473. ISSN: 21693536. DOI: 10.1109/ACCESS.2016.2599408.

[36] M. May, C. Neeb, and N. Wehn. “Evaluation of High Throughput Turbo-Decoder Architectures”. In: *2007 IEEE International Symposium on Circuits and Systems*. 2007, pp. 2770–2773. DOI: 10.1109/ISCAS.2007.378627.

[37] Alexander Worm, Holger Lamm, and Norbert Wehn. “Design of low-power high-speed maximum a priori decoder architectures”. In: *Proceedings -Design, Automation and Test in Europe, DATE* (2001). ISSN: 15301591. DOI: 10.1109/DATE.2001.915035.

[38] Matthias May *et al.* “A 150Mbit/s 3GPP LTE Turbo code decoder”. In: *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE, 2010, pp. 1420–1425. ISBN: 978-3-9810801-6-2. DOI: 10.1109/DATE.2010.5457035. URL: http://ieeexplore.ieee.org/document/5457035/.

[39] Stefan Weithoffer, Frederic Pohl, and Norbert Wehn. “On the applicability of trellis compression to Turbo-Code decoder hardware architectures”. In: *International Symposium on Turbo Codes and Iterative Information Processing, ISTC*. 2016. ISBN: 9781509034017. DOI: 10.1109/ISTC.2016.7593077.

[40] M. Jezequel. “Turbo4: a high bit-rate chip for turbo code encoding and decoding”. In: 2006. DOI: 10.1049/ic:19990784.

[41] Amir M. Rahmani *et al. The Dark Side of Silicon*. Ed. by Amir M. Rahmani *et al.* Cham: Springer International Publishing, 2017, pp. 1–347. ISBN: 978-3-319-31594-2. DOI: 10.1007/978-3-319-31596-6. URL: http://link.springer.com/10.1007/978-3-319-31596-6.

[42] *Digital*. URL: https://github.com/hneemann/Digital.

[43] Chixiang Ma and Ping Lin. “Efficient implementation of quadratic permutation polynomial interleaver in turbo codes”. In: *2009 International Conference on Wireless Communications and Signal Processing, WCSP 2009* (2009), pp. 1–5. DOI: 10.1109/WCSP.2009.5371680.

[44] Arash Ardakani, Mojtaba Mahdavi, and Mahdi Shabany. “An efficient VLSI architecture of QPP interleaver/deinterleaver for LTE turbo coding”. In: *Proceedings - IEEE International Symposium on Circuits and Systems* (2013), pp. 797–800. ISSN: 02714310. DOI: 10.1109/ISCAS.2013.6571967.

[45] Heiner Lasi *et al.* “Industry 4.0”. In: *Business and Information Systems Engineering* 6.4 (2014), pp. 239–242. ISSN: 00218774. DOI: 10.1007/s12599-014-0334-4. URL: https://doi.org/10.1007/s12599-014-0334-4.

Alphabetical Index

3GPP, 12

AlgoSil, 20
AM, 27
AM+, 27
ASIC, 23

Block Codes, 10

Coding Theory, 10
Compound Codes, 10
Convolutional Codes, 10

ECC, 10
EDA, 39

FEC, 10
FPMAP, 16

Gajski-Kuhn Y-chart, 24
GCD, 15

HDL, 25

Information theory, 9
Interleavers, 12

LDPC, 10
LTE, 12

NOC, 19

P-MAP, 16
PMAP, 16

QPP, 12

RSC, 10
RTL, 23

SiLago, 18

Turbo Codes, 10

UXMAP, 16

VLSI, 1

XMAP, 16

TRITA EECS-EX-2019:256