Fast Prototyping of Massive MIMO User Equipment Using PYNQ

Keming Mao
ke0457ma-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Liang Liu
Co-supervisor: Sijia Cheng

Examiner: Erik Larsson

January 8, 2024

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

When technology is being developed in wireless communication systems, it is crucial to achieve very high spectrum efficiency. The massive Multiple-Input Multiple-Output (MIMO) system is one of the techniques used to ensure ultra-high spectrum efficiency by spatial multiplexing. With multiple antennas implemented on the Base Station (BS), the information transmission quality, throughput, and capacity are considerably increased. Utilizing the same time and frequency resources, massive MIMO allows multiple User Equipment (UE) to concurrently connect to the Base Station (BS).

Implementing UE using a Register-Transfer-Level (RTL) approach may consume considerable time and effort, especially when optimizing fast-speed, low-power, and small-area designs. Thus, the High-Level-Synthesis (HLS) will be performed in this design to reduce the demanding amount of developing time. Using C++ to implement Hardware Description Language (HDL) can lead to inefficient designs due to the lack of control over low-level circuits. In this situation, several optimization directives will be employed to fulfill the specific requirements.

In this thesis, the design will be implemented in Python Productivity for ZYNQ (PYNQ) to conduct real-time verification. A loop-back verification will be performed between the UE and PYNQ RX in the baseband to confirm the design's functionality on the hardware. Extracting RTL code from Vitis HLS to PYNQ accelerates hardware implementation development due to its fast iteration characteristics.

Acknowledgement

I sincerely express my thankfulness to my supervisor Professor Liang Liu and co-supervisor, Ph.D. student Sijia Cheng for your help and patience. Finally, I want to acknowledge my parents and all my friends who gave their support and encouragement.

Keming Mao

Popular Science Summary

Due to the increasing demand for high transmission efficiency and large capacity in wireless systems, researchers made great efforts to exploit multi-path propagation. It can be dated back to the 90s when the concept of multi-channel transmission was proposed. Many engineers improved the MIMO system over the last decades, Nowadays, MIMO systems play a crucial role in achieving faster and more efficient wireless transmission. This summary will clarify the evolution of massive MIMO systems, from Single-Input Single-Output (SISO) to massive MIMO.

SISO stands for single input and output systems. These systems have unique transmission paths between the TX and RX, which can result in poor reliability and performance. However, Single-Input Multiple-Output (SIMO) systems can help improve this situation. By replacing the single transmission path with two paths, the reliability of the system is doubled compared to SISO. Multiple-Input Single-Output (MISO) also has the same ability as SIMO. The quality of transmission is a crucial aspect of modern wireless communication, the MIMO system improves not only the quality but also transmission speed and capacity to some extent. There is a clear distinction between MIMO and massive MIMO, which is the number of antennas located at the BS. Multi-User MIMO is a type of MIMO technology where the number of antennas is the same between the UE and the BS. In contrast, massive MIMO has a much larger number of antennas located at the BS compared to the UE. The implementation of massive MIMO involves using multiple antennas at both the transmitter and receiver sides. This results in a more complex process for developing RTL designs. However, using High-Level Synthesis (HLS) can greatly speed up this process.

The development of HLS can be dated back to 1968 when it was first invented to help engineers to abstract circuit design from a high level. Back then, the HLS was primarily proposed by researchers who were mostly located on campus. The application was limited to small designs. During the 90s, several Electronic Design Automation (EDA) companies including Cadence, Synopsys, and Mentor launched the HLS platform for commercial use, Unfortunately, the quality of the results was considered poor in the industrial design. Currently, universities and industries collaborate to make the HLS an important part of circuit design. Nowadays, high-level programming languages like C++, C, and System Verilog are widely used in

various fields such as machine learning, DSP design, and System on Chip (SoC) to conduct the HLS development process. The accuracy of the RTL code translation in scheduling and timing mapping has significantly improved over the last few years. Therefore, this thesis will use HLS for fast prototyping of massive MIMO UE.

Table of Contents

List of Figures

List of Tables

List of Acronyms

MIMO	Multiple-Input Multiple-Output
BS	Base Station
UE	User Equipment
RTL	Regsiter Transfer Level
HLS	High-Level Synthesis
HDL	Hardware Description Language
PYNQ	Python Productivity for ZYNQ
SISO	Single-Input Single-Output
SIMO	Single-Input Multiple-Output
MISO	Multiple-Input Single-Output
EDA	Electronic Desing Aautomation
SoC	System on Chip
5G NR	Fifth Generation New Radio
SDMA	Space Division Multiplexing Access
FDMA	Frequency Division Multiplexing Access
TDMA	Time Division Multiplexing Access
OFDM	Orthogonal Frequency Division Multiplexing
CDMA	Code Division Multiplexing Access
FFT	Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform
ISI	Inter-Symbol Interference
STBC	Space Time Block Coding
QPSK	Quadrature Phase-Shift Keying

16QAM	16 Quadrature Amplitude Modulation
64QAM	64 Quadrature Amplitude Modulation
CP	Cyclic Prefix
ICI	Inter-Carrier Interference
PL	Programming Logic
PS	Processing System
GPIO	General Purpose Input Output
MMIO	Memory Mapped IO
DMA	Direct Memory Access
II	Iteration Interval
FIFO	First-In First-Out

Chapter 1
Introduction

The introduction will consist of three parts. The motivation and challenge section will provide an overview of the massive MIMO system and challenge in this thesis, the thesis purpose will clarify the aim to implement the UE transmitter using HLS, and the outline will provide a summary of all chapters.

1.1 Motivation and Challenges

The Fourth Generation (4G) wireless systems use massive MIMO systems to increase data rates [1], signal quality, and spectrum efficiency. Furthermore, the integration of millimeter waves, also called millimeter bands, into RF systems can also improve the aforementioned parameters. Due to the increased demand for bandwidth, the 5G NR frequency range1 and range2 can both reach up to gigahertz, resulting in significantly low power received at RX as shown in (1.1). P_{rx} and P_{tx} represent power at the receiver and transmitter side respectively. G_{rx} and G_{tx} represent gain at the receiver and transmitter side. λ represents wavelength, and R represents the distance between the two sides. Implementing millimeter wave technology introduces significant challenges due to the usage of high-frequency bandwidth, which ranges from 30GHz to 300GHz.

$$P_{rx} = \frac{P_{tx}}{4\pi R^2}\frac{\lambda^2}{4\pi}G_{rx}G_{tx} \tag{1.1}$$

One way to enhance the power gain at RX is to use the massive MIMO system. By implementing multiple antennas, the Space Division Multiplexing Access (SDMA) is introduced. Compared to Time Division Multiplexing Access (TDMA) and Frequency Division Multiplexing Access (FDMA), the SDMA increases the capability of bandwidth and immunity to external interference. However, improving spectrum efficiency leads to increased computational complexity. According to Robin and Robert [2], pilot contamination, signal detection, channel estimation, precoding, hardware impairment, and user scheduling present the most significant challenges. Pilot contamination mostly happens when the pilot signals interfere with neighboring cells. This phenomenon will deteriorate the channel estimation quality. When the multiple antennas were implemented, more effort was needed

to deal with the complexity of computation when detecting signals, estimating channels, and precoding. This thesis will concentrate on enhancing the performance of hardware by implementing some configurable improvements in modulation schemes and user scheduling. Specifically, a reconfigurable modulator that supports three modulation schemes and reconfigurable uplink pilot signals to support four different UE will be developed. However, introducing reconfigurability creates challenges in both the modulator and pilot signal areas. Furthermore, fast prototyping of UE can also be challenging since the traditional approach of using RTL is time-consuming and weak in wireless signal processing.

1.2 Thesis's Purpose

With the advancement of wireless communication systems, signal processing, and operations have become more complex on hardware. The HLS has played an important role recently by virtue of its fast iteration capability. Additionally, HLS using C/C++ based development process provides an opportunity for powerful algorithm crafting. Therefore, some research has been conducted to compare the pros and cons of RTL and HLS approaches. Ghattas and Ali performed an evaluation between HLS and HDL when designing radix-2 FFT and radix-4 FFT for digital signal processing [3]. The results indicate that HLS can maintain the same speed and resources as RTL while requiring less development time. It also provides flexible directive constraints for optimization without the need for code refactoring.

When implementing a wireless system UE that focuses on signal processing on FPGA/ASIC for transmission, it is crucial to balance area and speed [4]. This thesis utilizes IFFT and FFT algorithms with specific requirements in a reconfigurable efficient UE transmitter, making HLS an ideal option. Further details on hardware architecture and optimization will be discussed later.

1.3 Outline

This section provides an overview of the motivation and challenges in this thesis, giving a preliminary explanation of why using massive MIMO in wireless systems and why using HLS to implement UE on PYNQ.

Chapter 2 will elaborate on the background information to give preliminary ideas on the key points of the thesis. To begin with design application, and the standards and protocols should be followed. Afterward, the approaches and methodology will be explained.

In Chapter 3, architecture design and details will be presented. Starting from elaborating on the UE system, the performance requirements and the block diagram of the design will be shown.

Chapter 4 focuses on design optimization, using several directives to improve design performance.

Chapter 5 presents loop-back verification on PYNQ, including the comprehensive understanding of design architecture, the discussion of the verification results, and suggestions for future work.

Chapter 2

Background Information

2.1 Background in Wireless System

2.1.1 Frequency-selective Fading

In 2G, TDMA is the primary access scheme for users, while CDMA was widely adopted as the primary approach for 3G. However, the use of these two access technologies can result in a phenomenon known as frequency-selective fading, caused by Inter-Symbol Interference (ISI), which can degrade the signal quality. Thus, a scheme with multi-carriers was introduced in 4G and 5G. As Figure 2.1 illustrated, the inter-communication between BS and users may contain multiple paths. The multi-path propagation will interfere with each other. Assuming there is one user and two paths when communicating. After FFT, The two paths with different timing can be shown below in (2.1), C denotes the attenuation factor, $R(w)$ denotes the received signal in the frequency domain, t_0 and t denote the first path delay and delay between two paths respectively.

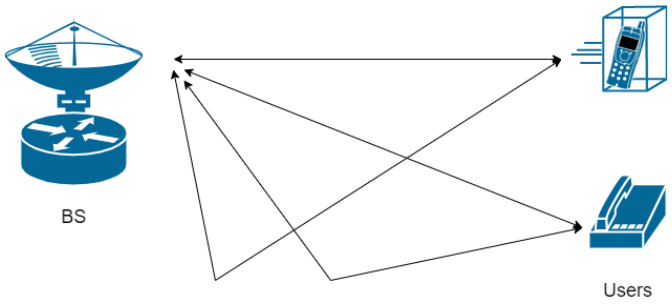


Figure 2.1: Multi-path propagation.

$$R(w) = CF(w)e^{-jwt_0}(1 + e^{-jwt}) \tag{2.1}$$

$$H(w) = \frac{R(w)}{F(w)} = Ce^{-jwt_0}(1 + e^{-jwt}) \tag{2.2}$$

Rewriting the frequency function as (2.2) will result in frequency-selective fading

concerning w.

2.1.2 OFDM

The core idea of OFDM technology is to divide a wide-frequency carrier into multiple orthogonal sub-carriers with smaller bandwidths and use these orthogonal sub-carriers to transmit and receive signals. Considering the set of functions as (2.3), each two of them's integral is 0.

$$1, sinwt, coswt, sin2wt, cos2wt, ..., sinnwt, cosnwt \qquad w \in [-\pi, \pi] \tag{2.3}$$

This set of functions is defined as orthogonal functions which can be utilized in OFDM. Assuming the bandwidth is BW, the number of subcarriers is N, so the subcarrier spacing is

$$\Delta f = \frac{BW}{N} \tag{2.4}$$

for the timing domain, the signals transmitted can be denoted as the (2.5),

$$f(t) = \sum A_i \sin(2\pi f_i t) + \sum B_i \cos(2\pi f_i t) \tag{2.5}$$

the f_i equals $f_0 + (i-1)\Delta f$, rewrite as (2.6),

$$f(t) = \sum C_i e^{j2\pi f_i t} \tag{2.6}$$

To implement an OFDM transmitter, IFFT logic is needed. The 2.6 can be interpreted as conducting IFFT on the information signal C_i of each subcarrier. The Figure 2.2 illustrates the basic building blocks of OFDM TX.

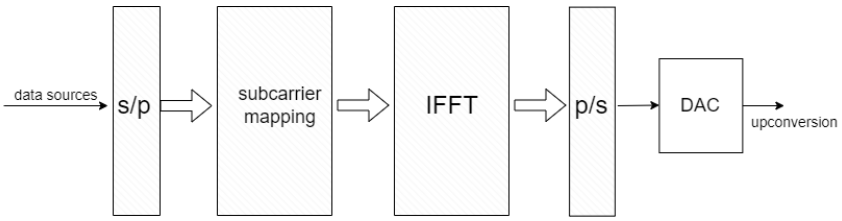


Figure 2.2: Basic structure of OFDM TX.

Based on multi-carrier capability and orthogonality, the frequency-selective fading is limited to smaller bandwidths, thus decreasing the impact of this phenomenon.

2.1.3 Massive MIMO System with OFDM

Assuming a massive MIMO system with m antennas at TX and k users at RX. The architecture as the Figure 2.3 showed below,

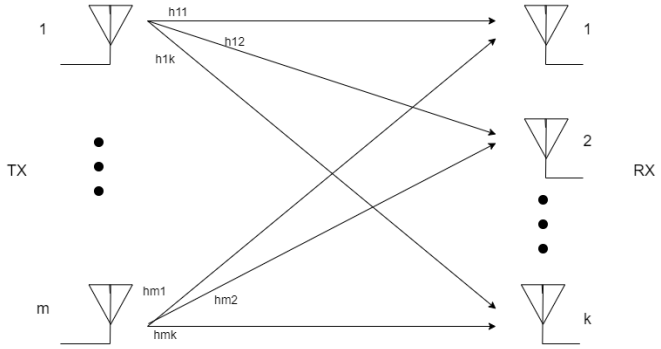


Figure 2.3: A massive MIMO system.

this system can be represented as a matrix model as the (2.7) below, $\boldsymbol{S_r}$ and $\boldsymbol{S_t}$ are received signal vector and transmitted signal vector, the sizes are $n_r \times 1$ and $n_t \times 1$ respectively. The n_r and n_t are antenna numbers on both sides. The $\boldsymbol{n_0}$ is a $n_r \times 1$ noise vector in the channel. $\mathbf{H}$ is a $n_r \times n_t$ channel matrix. As the (2.8) indicates, the h_{11} is the channel factor between the first antenna in TX and the first antenna in RX.

$$\boldsymbol{S_r} = \boldsymbol{H_c}\boldsymbol{S_t} + \boldsymbol{n_0} \tag{2.7}$$

$$\boldsymbol{H_c} = \begin{pmatrix} h_{11} & \cdots & h_{1n_t} \\ \vdots & \cdots & \vdots \\ h_{n_r1} & \cdots & h_{n_rn_t} \end{pmatrix} \tag{2.8}$$

A basic structure of massive MIMO OFDM TX is shown below, Space Time Block

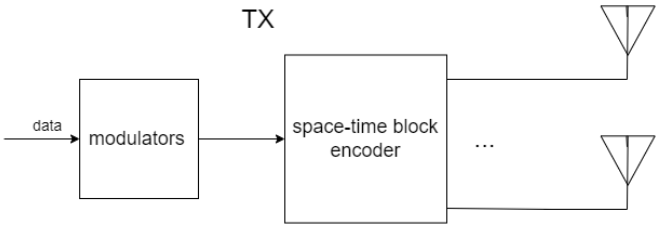


Figure 2.4: OFDM TX with STBC.

Coding (STBC) [5] is used to transmit multiple copies of a data stream in wireless communications. Generate multiple received versions of data through many antennas to improve the reliability of data transmission. There are three key concepts of massive MIMO [5],

- Spatial multiplexing
- Spatial diversity
- Beamforming

Massive MIMO systems employ spatial multiplexing to significantly boost data throughput and spectrum efficiency. The use of spatial multiplexing in massive MIMO systems enhances the system's capacity to handle more data by allowing for independent streams of information to be sent in parallel.

Wireless systems are often affected by obstacles such as buildings, air, and water, which can lead to a degradation in signal quality and affect the transmission of information. One method to overcome this issue is through the use of spatial diversity, which involves the parallel transmission of redundant streams of information. This technique can reduce the likelihood of errors occurring during the transmission process. On the receiving side, the redundant information can be processed to overcome the negative impact caused by the obstacles. As a result, spatial diversity can significantly improve the quality of the signal.

Beamforming is a technique used to improve the throughput and capacity in MIMO systems. Beamforming narrows the beam to improve signal quality, particularly in massive MIMO systems. Instead of sending signals in all directions, beamforming focuses the signal in a specific direction, as illustrated in Figure 2.5. This reduces the impact from different beams and enhances the signal strength in the interest direction.

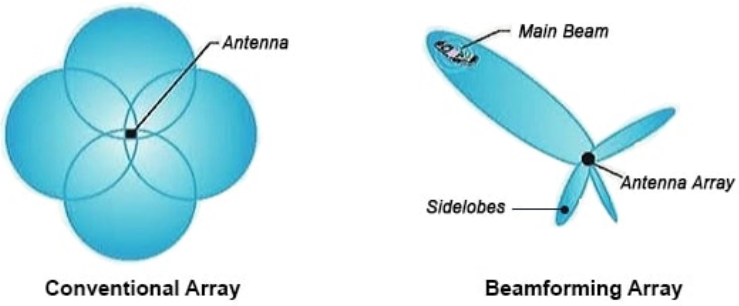


Figure 2.5: Beamforming comparison [6].

2.1.4 5G New Radio (NR)

The 5G NR standard was released at the end of 2017, it makes it compatible with 4G LTE and increases the frequency spectrum up to GHz (mmWave). Although both 4G and 5G use OFDM to exploit frequency and timing resources, 5G outperforms in terms of spectrum efficiency, speed, and low latency [7]. A short

comparison according to Misha's post [8] are shown below in Table 2.1

Technology	Data Rates	Latency	Spectrum Efficiency
5G (NR)	Peak 20 Gb/s	1ms	DL- 30bits/Hz
4G (LTE)	Peak 300 Mb/s	10-50 ms	DL- 6bits/Hz

Table 2.1: A brief comparison of 4G and 5G.

The main difference between 4G and 5G frame structures is that 5G uses a parameter called numerology to specify subcarrier spacing, while 4G uses fixed 15 kHz to configure the frame structure. As Figure 2.6 displayed, the length of the subframe is fixed, but slots vary with the numerology 2^{μ}.

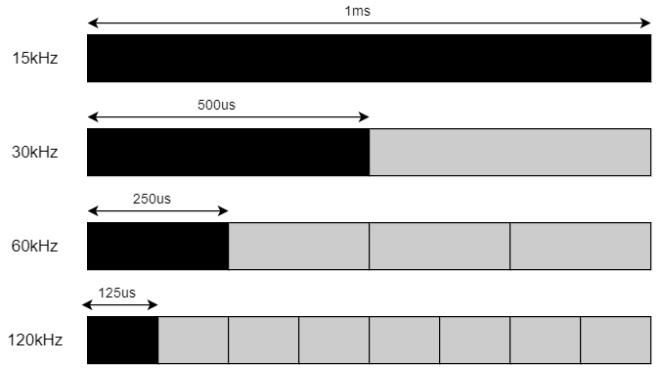


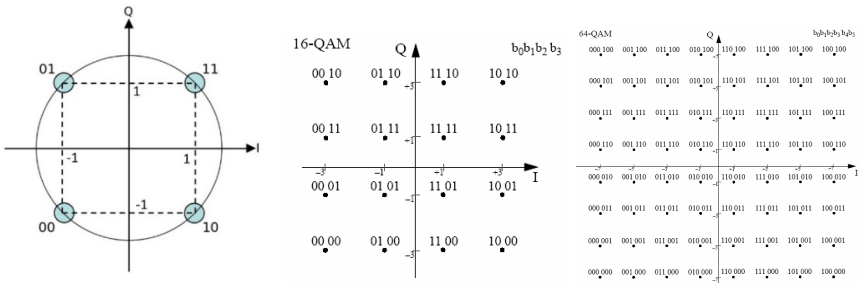
Figure 2.6: Subframe with different subcarrier spacing.

2.1.5 Modulation Schemes

This thesis includes three modulation schemes: Quadrature Phase-Shift Keying (QPSK), 16-Quadrature Amplitude Modulation (16QAM), and 64-Quadrature Amplitude Modulation (64QAM), each scheme maps the received bitstream in the frequency domain. Three schemes are illustrated below in Figure 2.7, each dot on the constellation diagram represents a unique amplitude and phase in the frequency domain.

2.2 HLS Implementation

Regarding HLS, some may assume it involves an automated process of converting high-level languages such as C++ or C into RTL code. Professor Daniel D. Gajski, however, defines the HLS as a mapping of a behavioral description of a digital



(a) QPSK [9] **(b)** 16QAM [10] **(c)** 64QAM [10]

Figure 2.7: Mdulation constellation diagrams.

system into an RTL design consisting of a data path and a control unit [11]. This definition not only requires engineers' software programming ability but also requires hardware knowledge. Extensive research has been conducted on using HLS by software engineers to achieve hardware acceleration.

2.2.1 Basic Vitis HLS Workflow

Compared to the traditional RTL approach, HLS has multiple advantages. One of them is accelerating the development process, which makes HLS a promising tool, especially in hardware algorithm crafting. Nonetheless, Juan J. Alonso researched evaluating performance loss of Stencil Computation using HLS [12], which showed weaknesses in resources and performance. Thus, it is crucial for engineers to use a better coding style and perform in the right usage. This thesis will use Vitis HLS as the development platform. The process is significantly different from RTL, Figure 2.8 simply illustrates the workflow of Vitis HLS.

Similar to RTL, the C++ code needed to be verified by functional simulation, and a C code testbench was added when simulating. The basic environment of the testbench is illustrated in Figure 2.9. The generator will operate the DUT by providing input data, in our case, MATLAB provides golden results for further verification. The monitor displays the step results when simulating, and the scoreboard performs a comparison between the DUT and reference model. Whenever there is a mismatch in the scoreboard, the simulation will fail. After simulation, users can optimize the design by adding directives, such as *pipeline, unfolding, dataflow*, etc. The output of C synthesis are reports containing timing and resource information, and RTL codes. Afterward, the co-simulation will be performed between the C testbench and RTL model, and the cycle-based waveform will be outputted for further verification.

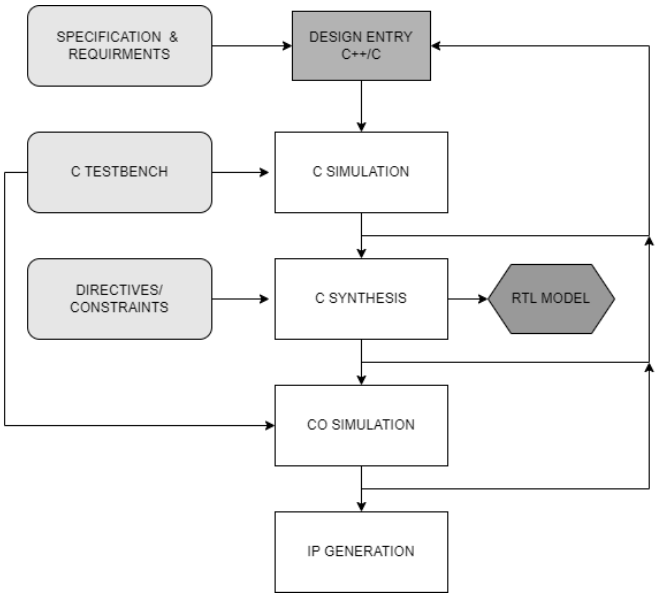


Figure 2.8: Workflow in Vitis HLS.

2.2.2 A Deep Dive In HLS Implementation

HLS allows for hardware design using high-level programming languages. However, it differs from HDL in that it is not clock-based. Hence, converting code to hardware must be efficient and accurate. This subsection will elaborate on some key procedures in HLS. Regarding to [13], some key concepts are *compilation, allocating, scheduling, and binding.*

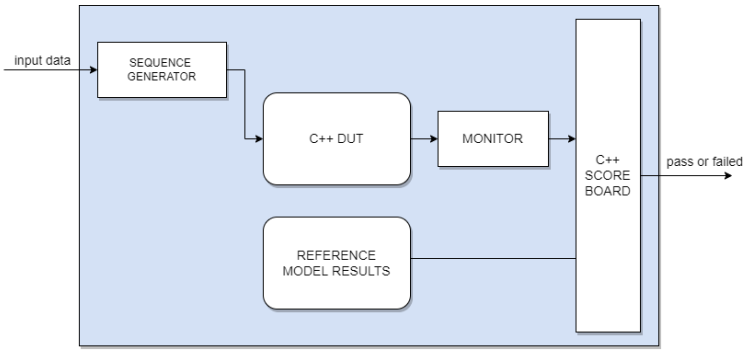


Figure 2.9: Environment of C testbench.

Compilation

In HLS, the code is compiled by analyzing the high-level language and abstracting functions as a data flow graph. This exhibits data dependencies between operations and enables further allocation and optimization.

Allocating

Allocating refers to selecting a specific type and number of hardware. This process may be similar to RTL design, when starting to develop code, it is necessary to determine the required hardware resources. The HLS tools choose the hardware by analyzing the code. For example, when performing matrix multiplication, the operation requires multiple adders and multipliers, and the results may need to be stored in memory such as registers or RAMs. The HLS hardware standard libraries may contain some information regarding the area, latency, and power for further optimization.

Scheduling

In C/C++, the code can be executed either in a serial manner or concurrently. In hardware, all the operations are executed based on the clock cycle. Scheduling refers to the process of organizing operations that take place in each clock cycle. Assuming the operation $y = a * b + c + d * e$, the scheduling can be both below in Figure 2.10, The first scheme demonstrates that operations are scheduled sequentially using one multiplier and one adder. On the other hand, the second scheme illustrates that the operations without data dependence are scheduled in parallel to achieve the smallest latency with one adder and two multipliers. The users can optimize scheduling for lower resource usage or higher throughput.

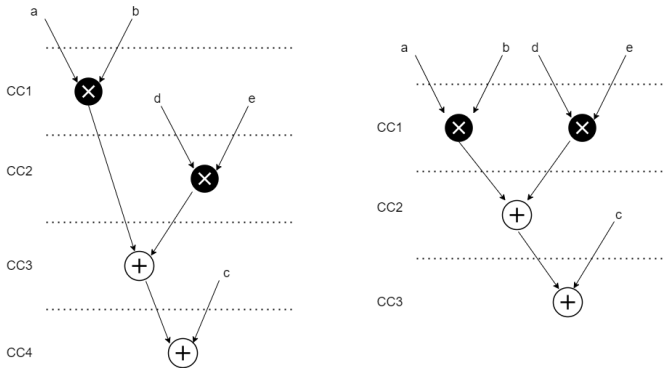


Figure 2.10: Different scheduling schemes.

Binding

Binding refers to mapping the operations into hardware, this procedure may have multiple mapping schemes as Figure 2.11 displayed, both of them are the binding results of Figure 2.10. The first binding needs two multipliers, one adder, and three registers. The second binding needs two DSP resources, one ALU, and three registers to achieve the operations. Engineers can optimize the design by inserting directives which may result in different mapping schemes.

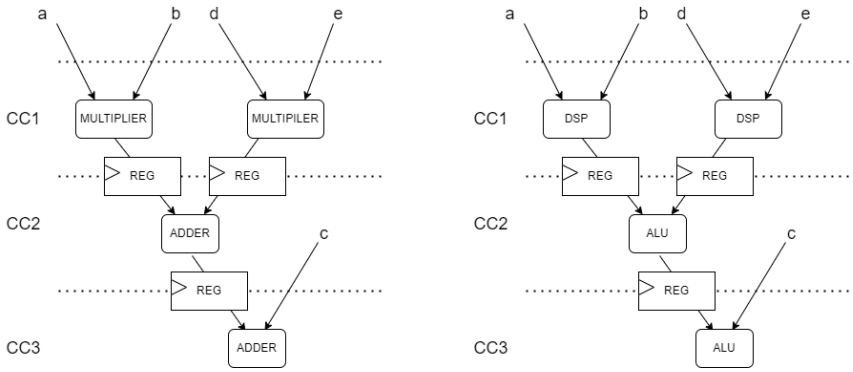


Figure 2.11: Different mapping schemes.

2.3 PYNQ Framework

The PYNQ framework includes an embedded processor that can be operated using the Python programming language. Besides the processor, PYNQ also includes an FPGA part called overlay. The processor and overlay are interconnected, enabling efficient control of the overlay through the processor. The overlays can be programmed in RTL or HLS. This feature simplifies the design process and allows for quick iteration. One of the primary advantages of PYNQ over ZYNQ is that it offers a Python-based programming framework, which facilitates fast prototyping of designs using Jupyter Notebook, a web-based programming tool. PYNQ has various features, which are listed below.

- **Python programming**: PYNQ enables users to interact with FPGA using Python easily, simplifying hardware control through a high-level language.
- **Jupyter notebook**: This tool is user-friendly, allowing for specific result processing and overlay testing.
- **ARM processor**: By incorporating an ARM processor, users can easily integrate hardware and software, while facilitating implementation of designs by software engineers.

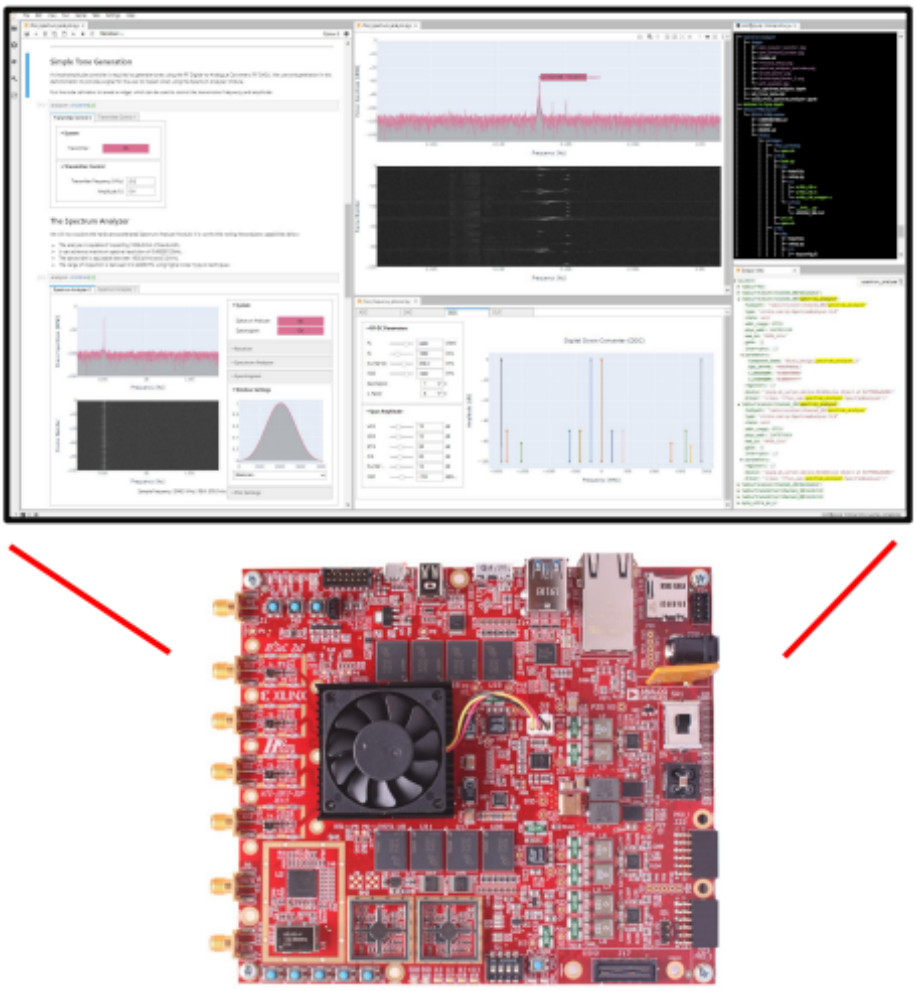


Figure 2.12: RFSoC2x2 board [14].

Chapter 3

Overall Architecture Design of the UE

3.1 An Overview of the UE

The top-level architecture is shown in Figure 3.1, and the bitstream data source is converted into parallel for further processing. The constellation mapping block can use one of three modulation schemes based on an external select signal. For reconfigurability, the uplink pilot signal is determined by an external signal and can configure 4 different UEs. By adding uplink pilot signals and performing sub-carrier mapping, the data is pre-processed in the frequency domain.

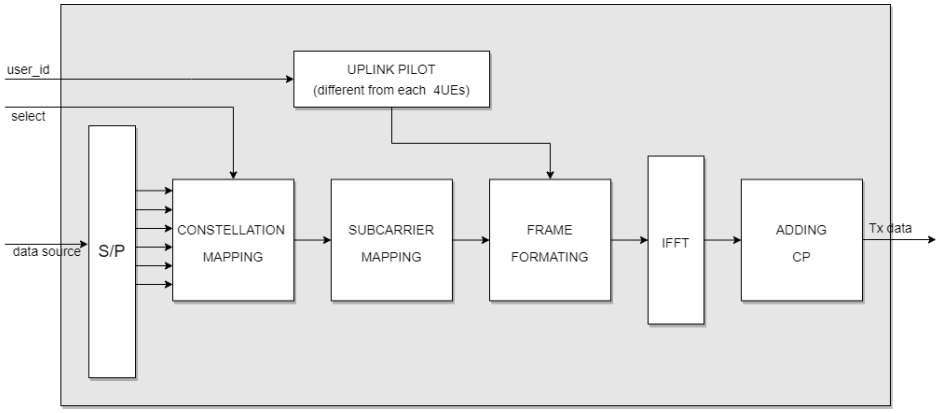


Figure 3.1: UE top-level architecture.

IFFT will convert frequency domain data into time domain. The output of this design is OFDM-framed data, carrying the source information in the time domain. The following subsections will exhibit the design in detail.

3.1.1 Reconfigurable Modulator

Different amounts of bits are required in QPSK, 16QAM, and 64QAM as Figure 2.7 illustrates, therefore, it is necessary to implement a serial-to-parallel circuit before sending the data source to the modulator. As shown in Figure 3.2, the select signal

chooses one of three modulation schemes to encode. By simply implementing shift registers, the data source is divided into 6 parallel inputs. The modulator utilizes 2 inputs in QPSK, 4 inputs in 16QAM, and 6 inputs in 64QAM.

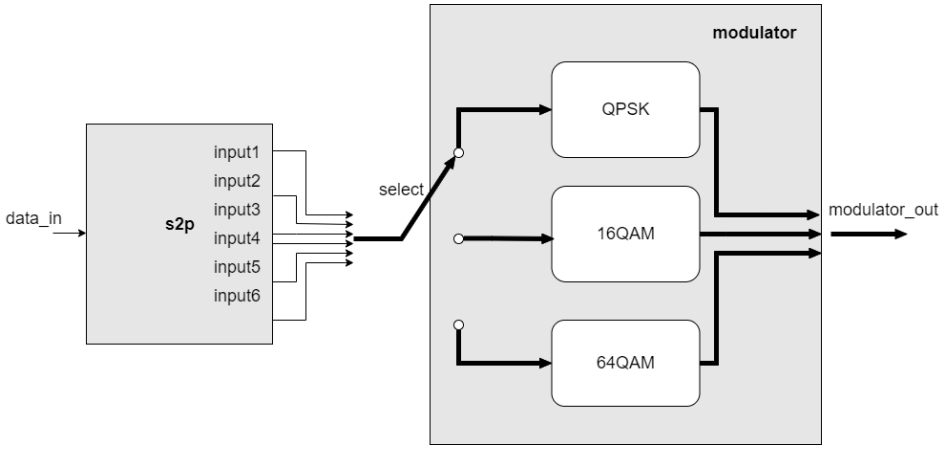


Figure 3.2: Basic structure of serial-to-parallel circuit and modulator.

3.1.2 Subcarrier Mapping

This subsection provides some knowledge on state-of-the-art subcarrier mapping technology and elaborates on the design's details. Assuming the information data is M, the IFFT point is 2^n, and $2^n - M$ zeros are needed to add in the symbol to perform IFFT. The process of subcarrier mapping involves mapping source data into a vector of size 2^n. The zeros work like the unused spectrum guard band to prevent interference. Regarding the research [15], there are three ways of subcarrier mapping, distributed, localized, and interleaved. Figure 3.3 illustrates the distributed and localized subcarrier mapping.

In this design, the localized subcarrier mapping is performed. The details are displayed below in Table 3.1.

Specification	Value
N_{symbol}^{data}	600
N_{IFFT}	1024
Mapping scheme	localized

Table 3.1: Subcarrier mapping parameters.

In this thesis, the mapping circuit maps 600 data into 1024 subcarriers, inserting

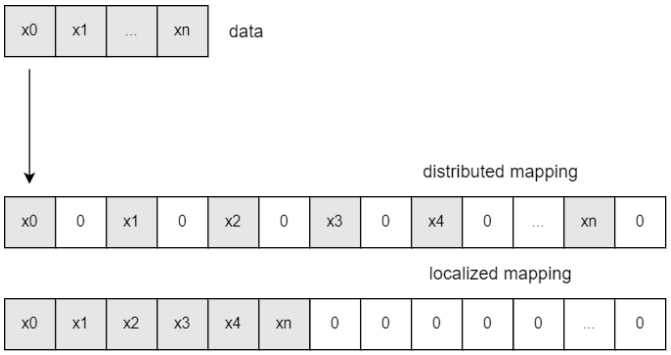


Figure 3.3: Subcarrier mapping schemes.

424 zeros. A simple data symbol is displayed below, it can be noted that the mapped data starts from no.1 subcarrier. The no.0 subcarrier is called the DC subcarrier, normally, the value should be 0. In the baseband, modulating the DC subcarrier to another frequency band causes the DC offset problem. It also can be noted that the last part of the data mapped into the 1-300 subcarriers and the first part of the data mapped into 724-1023 subcarriers. The switch is caused by the image frequency after it has been sampled.

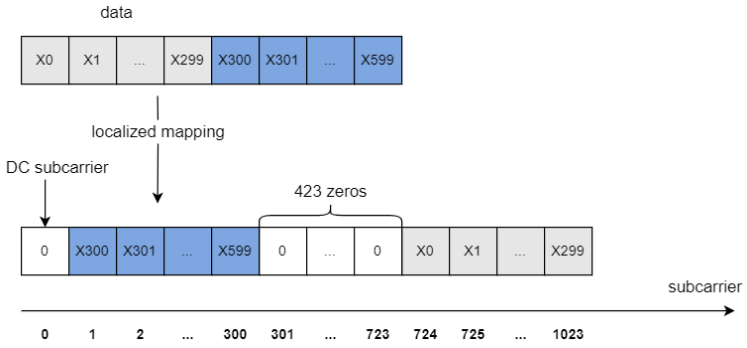


Figure 3.4: Data symbol structure.

3.1.3 Reconfigurable Uplink Pilot

In real-life scenarios, the signal received at the RX is influenced by multiple factors, and the channel may vary. To decode signals accurately, pilot signals transmitted by the TX at the beginning of each subframe are used to perform channel estimation. Assuming the channel is noiseless, rewrite (2.7) as below,

$$\hat{\boldsymbol{H}} = \frac{\boldsymbol{S_r}}{\boldsymbol{S_t}} \tag{3.1}$$

The channel estimation can be obtained with the received pilot signal and transmitted pilot signal as (3.1). $\hat{\boldsymbol{H}}$ is the estimated channel matrix. $\boldsymbol{S_r}, \boldsymbol{S_t}$ represent the received pilot signal and transmitted pilot signal respectively. Nonetheless, the division on hardware may be costly both in area and timing. So, it is crucial to use an effective method for conducting channel estimation.

In this design, an external signal can configure a different uplink pilot signal for UE. In each transmission, one of the four users is identified by the user-id signal. The pilot signal spacing is set to 4 and a ROM stores 150 pilot values which are placed in the pilot symbol. The pilot symbol is illustrated in Figure 3.5 below, the uplink pilot is only applied in the first symbol of each transmission.

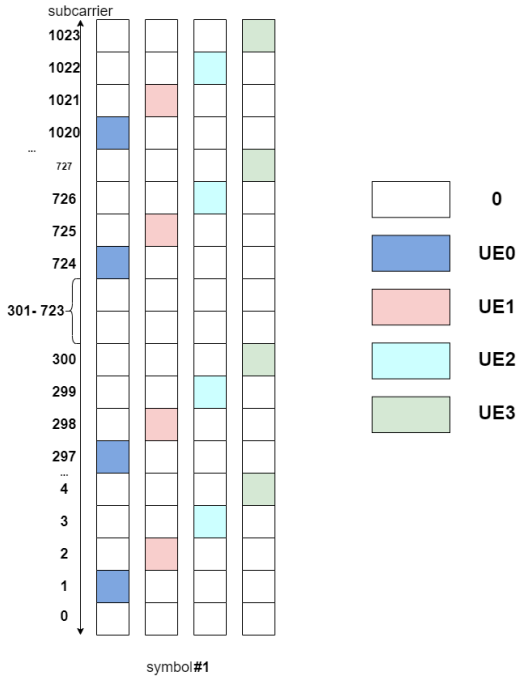


Figure 3.5: Pilot symbol structure.

3.1.4 FFT Using Radix-4 Butterfly

On hardware, implementing a DFT/IDFT algorithm is time-consuming and resource-intensive. To achieve the same functionality, engineers use radix-2/radix-4 FFT/IFFT. The algorithmic complexity is displayed below in Table 3.2, the algorithmic complexity decreased by using radix-4 also called butterfly radix-4 transformation. The two algorithms differ in the number of divided parts used in each computation. The former algorithm divides the computation into two parts, while the latter algorithm divides it into four parts. With more complexity in radix-4, however, it

requires 75 percent as many complex multipliers as radix-2. It is also efficient to utilize radix-4 FFT/IFFT with the input length of 4^n. In this design, the IFFT length is 1024, thus, the radix-4 IFFT is performed.

Algorithm	Complexity
DFT/IDFT	$\mathcal{O}(n^2)$
radix-2	$\mathcal{O}(n \log n)$
radix-4	$\mathcal{O}(n \log n)$

Table 3.2: Algorithmic complexity comparison.

Assuming the DFT function is (3.2), N represents the length of input, $k = 0, 1, ..., N-1$.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{N}} \tag{3.2}$$

Using radix-4 algorithm to divide the (3.2) into 4 parts, W_N^{nk} represents the rotation factor $e^{-j\frac{2\pi kn}{N}}$,

$$\begin{aligned} X(k) &= \sum_{n=0}^{N/4-1} [x_1(n)W_N^{nk} + x_2(n)W_N^{(n+N/4)k} + x_3(n)W_N^{(n+N/2)k} + x_4(n)W_N^{(n+3N/4)k}] \\ &= \sum_{n=0}^{N/4-1} [x_1(n) + x_2(n)W_N^{(N/4)k} + x_3(n)W_N^{(N/2)k} + x_4(n)W_N^{(3N/4)k}]W_N^{nk} \end{aligned} \tag{3.3}$$

as the equation above, the DFT has been divided into 4 parts with a cycle of $\frac{4}{N}$, rewrite yields,

$$\begin{aligned} X(4k) &= \sum_{n=0}^{N/4-1} [x_1(n) + x_2(n) + x_3(n) + x_4(n)]W_{N/4}^{nk} \\ X(4k+1) &= \sum_{n=0}^{N/4-1} [x_1(n) - jx_2(n) - x_3(n) + jx_4(n)]W_N^n W_{N/4}^{nk} \\ X(4k+2) &= \sum_{n=0}^{N/4-1} [x_1(n) - x_2(n) + x_3(n) - x_4(n)]W_N^{2n} W_{N/4}^{nk} \\ X(4k+3) &= \sum_{n=0}^{N/4-1} [x_1(n) + jx_2(n) - x_3(n) - jx_4(n)]W_N^{3n} W_{N/4}^{nk} \end{aligned} \tag{3.4}$$

in this case $k = 0, 1, ..., \frac{N}{4} - 1$, the N points FFT is transformed in $\frac{N}{4}$ points FFT, extract the basic computation as the (3.5) is shown below,

$$\begin{bmatrix} X(4k) \\ X(4k+1) \\ X(4k+2) \\ X(4k+3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x_1(n) \\ x_2(n) \\ x_3(n) \\ x_3(n) \end{bmatrix} \tag{3.5}$$

this computation can be represented in butterfly FFT, which is illustrated in Figure 3.6,

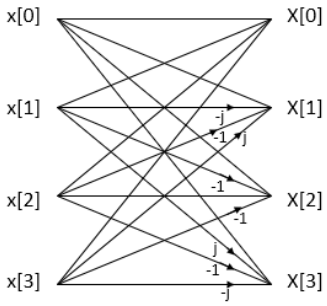


Figure 3.6: 4-point butterfly FFT [16].

ROMs are required to store the rotation factor $e^{-j\frac{2\pi kn}{N}}$ with different phases, the radix-4 butterfly is completed by the FFT core with multiple stages, the 1024-point vector is first divided into 256 4-point FFT, then divided into 64 4-point FFT, and so on.

3.1.5 OFDM Symbol Frame

This thesis uses the 5G NR protocol to communicate, the transmission in 5G is based on the OFDM symbol. The requirements and specifications are listed below in Table 3.3,

Parameters	Value
Bandwidth	50 MHz
$f_{sampling}$	61.44 MHz
Subcarrier spacing	60 kHz
OFDM symbol	16.7μs

Table 3.3: System parameters.

The OFDM symbol frame is displayed below in Figure 3.7, the subframe duration is fixed in 1ms, with 60KHz subcarrier spacing, the $N_{slots}^{subframe} = 4$. Each slot contains 14 symbols. The first symbol corresponds to uplink pilots while the next

six correspond to data signals. The remaining symbols are set to zero to accommodate downlink signals. After performing IFFT on the frequency domain data, the information bits are converted into the time domain.

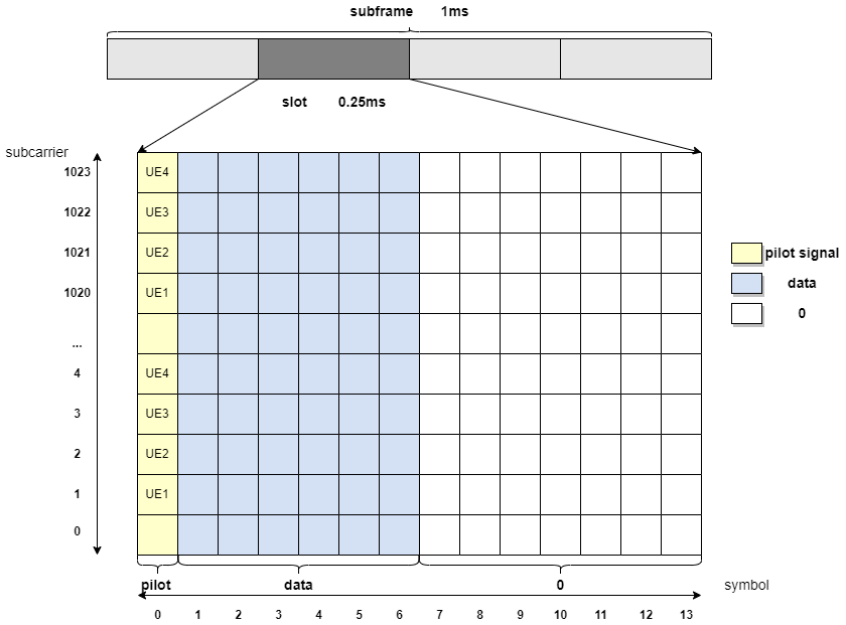


Figure 3.7: OFDM symbol frame.

Before sending the data, a necessary guard band called Cyclic Prefix (CP) is added to the symbol. Assuming that each symbol's end has μ data added to its start, the $x_1(n)$ is listed below,

$$x(N-\mu), ..., x(N-1), x(0), x(1), x(2), ..., x(N-1) \tag{3.6}$$

the length of μ data is called CP, the number of valid signals is N, let

$$h(x) = [h(0), h(1), ..., h(\mu)] \tag{3.7}$$

with (3.6) and (3.7),

$$y(n) = h(n) * x(n) = \sum_{k=0}^{\mu} h(k)x(n-k) \tag{3.8}$$

from (3.6) have

$$x_1(-\mu) = x_1(N-\mu) \tag{3.9}$$

combine (3.8) and (3.9), The linear convolution can be converted into circular convolution, which can be represented by the symbol $\otimes$.

$$y(n) = \sum_{k=0}^{\mu} h(k)x((n-k))_N R_N(n) = h(n) \otimes x_1(n) \tag{3.10}$$

Therefore, by converting to circular convolution, the IFFT/FFT can be performed using the radix-4 butterfly algorithm. The CP is an approach that not only improves the computation efficiency but also reduces the effects of Inter-Carrier Interference (ICI) and ISI. In 5G, with subcarrier spacing 60KHz, the number of CP is 144, and the time domain data structure is displayed below.

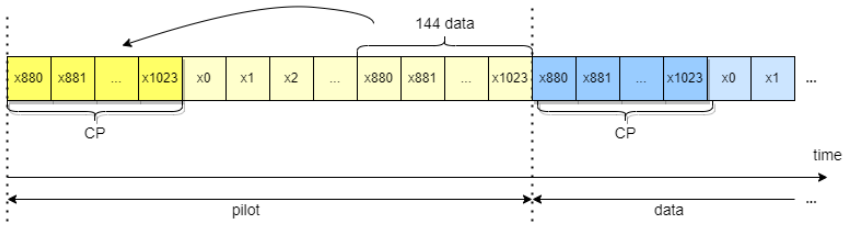


Figure 3.8: Time domain after IFFT.

Chapter 4

Design Optimization and Implementation Results

4.1 Optimization Directives

In Vitis HLS, there are multiple directives available for optimization, which can affect both speed and resource utilization. This section will explain the essential directives that are often used. In the next section, the comparison between the optimized and un-optimized designs will be performed.

4.1.1 Loop Unrolling

By replicating the circuits to several parts, the loop unrolling directive enables iterations to be executed in parallel, decreasing latency but increasing area and resource utilization. loop with a loop bound of 6, by performing the full unrolling on this loop, there will be 6 replication circuits executed in parallel. For large loops, the unrolling can be implemented in partial with a factor, as the Figure 4.1 displayed,

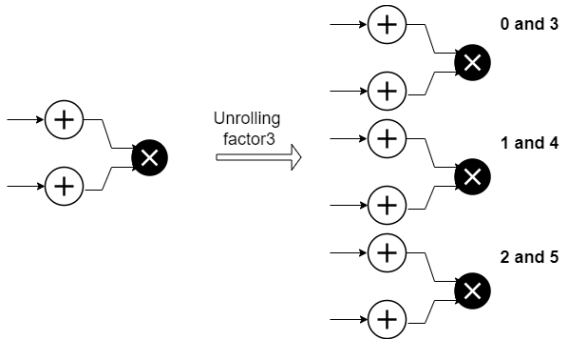


Figure 4.1: Partial loop unrolling.

the loop is unrolled into 3 circuits, each of which takes charge of two iterations, the first and the third iterations are performed on the first circuit, and so on. Partial unrolling provides an approach to balance between area and performance.

4.1.2 Loop Merge

The loop merge directive merges two loops, which reduces the total number of execution cycles. When two loops execute in serial, the total cycles considering the loop entry and exit are $2 \times (8 + 1) = 18$. However, by merging two loops, the execution cycles are reduced to 9 cycles.

```
for (int i =0; i<8; i++){
    c[i]=a[i]+b[i];}

for (int i =0; i<8; i++){
    d[i]=a[i]-b[i];}
```

```
for (int i =0; i<8; i++){
    c[i]=a[i]+b[i];
    d[i]=a[i]-b[i];}
```

Figure 4.2: Loop merge optimization.

Using loop merge can greatly enhance the performance of Vitis HLS compiled code. This is because the compilation order is sequential, later loops must wait for earlier loops to finish. resulting in poor performance. By merging two loops, the control logic decreases, reducing the utilization of resources and improving latency.

4.1.3 Pipeline

The most commonly used approach to improve the speed in RTL is pipeline. Similarly, the directive pipeline in HLS also leads to a faster design. As the figure illustrated below, the three operations executed in serial which results in a latency of 9 clock cycles. The pipeline allows the Adder operation to be executed immediately, without waiting for all operations to complete.

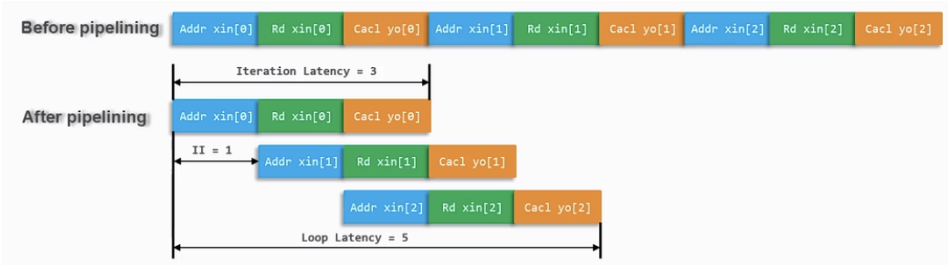


Figure 4.3: Pipeline optimization [17].

The design reaches a Iteration Interval (II) of 1 after the pipeline. which means the next new iteration will be executed in every clock cycle. The latency of the entire loop decreased from 9 to 5, which resulted in a faster design. However, achieving fast speed requires adding more circuits because pipelining the design necessitates more control logic. Additionally, utilizing a pipeline directive can also enhance the throughput. Thus, the pipeline directive can be a trade-off between performance and area.

4.1.4 Dataflow

The dataflow directive can improve the latency of several functions with data dependency. Assuming three functions executed in serial, the total latency can be calculated by adding the latency of all three functions together.

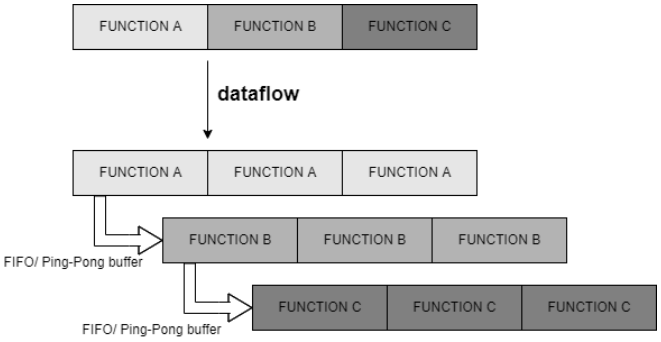


Figure 4.4: Dataflow optimization.

After dataflow, function B can execute as long as it has the required data from function A, without waiting for function A to complete. The overall latency was reduced through parallel computation. Figure 4.4 demonstrates the implementation of a channel between two functions, using a FIFO, registers, or Ping-Pong buffer, to enable the overlap of two functions. The dataflow directive can minimize latency and increase throughput, providing an alternative method for better throughput.

4.1.5 Array Partition

Arrays are often used to store data in high-level languages. In HLS, arrays are synthesized to ROM, FIFO, RAM, etc. Optimizing arrays is crucial to avoid performance bottlenecks and reduce area usage. It is possible to specify arrays to hardware like SPRAM and DPRAM with resource directives. The most commonly used method to optimize an array is partitioning. By specifying the partition factor, the array is divided into several parts, which are then mapped to actual memory. This improves throughput and avoids bottlenecks. There are three ways to partition an array, block, cyclic, and complete.
As Figure 4.5 displayed, the Block partition divides the original array into smaller arrays in a continuous manner, while the cyclic partition divides the array into smaller ones in a round-robin manner. The complete option can significantly increase resource usage and decrease performance when transferring arrays into registers.

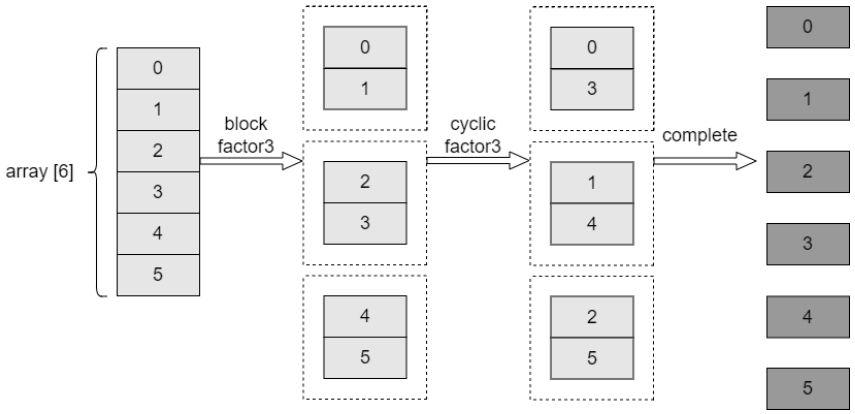


Figure 4.5: Different array partition schemes.

4.2 Design Optimization

4.2.1 Unoptimized Solution

The software used to optimize is Vitis HLS 2023.1. The first solution on HLS optimization using the directives displayed below in Table 4.1 to ensure the success of synthesis. The AXI-stream interface is used to implement the inputs and outputs for communication with DMA. The internal stream is used between the modulator and the subcarrier mapping module, the stream will be synthesized in a First-In First-Out (FIFO) with a depth of 3600 to avoid deadlock. Memory optimization is applied on the output to partition the array with a block factor 14, it will ensure the memory does not exceed the capacity.

Directives	Value
Clock_period	$10ns$
Stream_depth_mod_out	3600
Array_partition_block	14
Interface	AXI-S

Table 4.1: Unoptimized solution directives.

By performing all the necessary directives, the utilization report is shown below in Table 4.2. The equation below calculates the required throughput. n represents the data processed, while T_k represents the total time used to process n data. 1024 data are waiting to be processed for one symbol, and the required time for one symbol is mentioned in Table 3.3.

$$throughput_0 = \frac{n}{T_k} = \frac{1024Samples}{16.7\mu s} = 61.32MSamples/s \tag{4.1}$$

The unoptimized solution requires fewer resources, but it results in poor perfor-

Resources	Total	Utilization%
BRAM	103	2%
FF	12226	1%
DSP	0	0
LUT	19117	1%

Table 4.2: Unoptimized solution resources results.

mance. The timing information results are listed below, noticing the throughput of the unoptimized design is $18.3MSamples/s < throughput_0$. Thus, further optimization should be applied to achieve a faster design.

Modules	Latency(cycles)	II(cycles)
Input loop	72008	72008
Modulator	72008	72008
Pilots	13589	13589
IFFT	3195	3195
CP	2347	2347
Loop output	16354	16354

Table 4.3: Unoptimized solution timing results.

$$throughput = \frac{n}{T_k} = \frac{32704Samples}{1787.1\mu s} = 18.3MSamples/s \tag{4.2}$$

4.2.2 Solution with Loop Merge

As Table 4.3 displayed, the modulator and loop input has the most latency, thus, to improve performance and decrease hardware costs, loop merging is utilized during synthesis, which combines multiple loops into fewer loops to improve latency. The resources cost is shown below,

Resources	Total	Utilization%
BRAM	97	2%
FF	12302	1%
DSP	0	0%
LUT	18520	1%

Table 4.4: Loop merge solution resources results.

The timing results are shown below. Noticeably, the latency and initiation interval of both the loop input and modulator have significantly decreased due to

the directive loop merge. As a result, the throughput has been boosted to 29.6 $MSamples/s$. Unfortunately, it still falls short of the requirement.

Modules	Latency(cycles)	II(cycles)
Input loop	7170	7170
Modulator	3617	3617
Pilots	13589	13589
IFFT	3195	3195
CP	2347	2347
Loop output	16354	16354

Table 4.5: Loop merge solution timing results.

$$throughput = \frac{n}{T_k} = \frac{32704Samples}{1104.88\mu s} = 29.6MSamples/s \tag{4.3}$$

4.2.3 Solution with Array Partition

To get better performance, the directive array partition is applied to synthesis. In this solution, the block partition is applied to synthesis. The memory optimization will significantly increase the resources with a better performance. The resource report is shown below,

Resources	Total	Utilization%
BRAM	93	2%
FF	17428	1%
DSP	0	0%
LUT	90924	7%

Table 4.6: Partiton solution resources results.

The timing result is displayed below, which indicates that the array partition reduces the latency of the cp and loop input. However, the loop output and pilots still contribute the most to latency. This is due to the version of Vitis HLS which is not efficient in partitioning loops with more than 3000. Additionally, the submodules do not work in an overlapping manner, so further optimization is required to decrease II.

$$throughput = \frac{n}{T_k} = \frac{32704Samples}{981.46\mu s} = 33.32MSamples/s \tag{4.4}$$

Modules	Latency(cycles)	II(cycles)
Input loop	4098	4098
Modulator	3617	3617
Pilots	13589	13589
IFFT	3195	3195
CP	584	584
Loop output	16372	16372

Table 4.7: Partition solution timing results.

4.2.4 Solution with Dataflow & Pipeline

To get better performance, the directive dataflow and pipeline are applied to synthesis, the task-level optimization will significantly improve the latency and throughput, the resources report is shown below,

Resources	Total	Utilization%
BRAM	227	5%
FF	118960	4%
DSP	14	1%
LUT	335421	25%

Table 4.8: Dataflow & pipeline solution resources results.

Although parallelism can lead to higher hardware costs, it results in better performance as shown by the timing results. Compared to the previous design, the II has decreased significantly, resulting in a much-improved throughput of 98.98 $MSamples/s$, which meets the requirement of 61 $MSamples/s$. However, further optimization could be applied to improve the II and latency, but this would significantly increase the synthesis time due to resource limitations.

Modules	Latency(cycles)	II(cycles)
Input loop	4096	4
Modulator	3617	1
Pilots	13589	1
IFFT	5254	5254
CP	584	1
Loop output	16372	14

Table 4.9: Pipeline&dataflow solution timing results.

$$throughput = \frac{n}{T_k} = \frac{32704Samples}{330.4\mu s} = 98.98MSamples/s \tag{4.5}$$

4.2.5 Analysis and Summary

This chapter discusses how to achieve desired performance using different techniques and comparing used software versions. The version of the tool used for synthesis is crucial as it can impact the efficiency of the design. In this study, synthesis was initially performed in VIVADO HLS 2017.4, which resulted in an inefficient design. However, when the same design was synthesized in Vitis HLS 2023.1, it performed better than in VIVADO HLS 2017.4. A comparison of the two releases is presented in Table 4.10, which highlights two designs with the same directives but different versions. The latest version resulted in a speed improvement of about 1.7 times faster. As explained in Chapter 2, the process of converting a high-level language to HDL involves several steps such as compilation, allocation, scheduling, and binding. The most recent version of the software may have improved binding and scheduling techniques, which leads to faster synthesis results compared to older versions.

Version	Latency(cycles)	II(cycles)
VIVADO HLS 2017.4	269868	269868
Vitis HLS 2023.1	162359	162359

Table 4.10: Comparison between two software versions.

To improve loop efficiency and reduce resource usage, one can use the loop merge directive. By merging identical loop bounds loops, the total execution time can be reduced, improving throughput by approximately 1.6 times faster than an unoptimized loop. Array partition is applied to enhance memory and loop throughput, resulting in better performance but at the cost of increased hardware. The pipeline and dataflow top-level directives have been employed to enhance the design speed and improve the throughput performance, which is now three times better than the previous solution. The use of dataflow and pipeline has led to the insertion of more registers, which has improved the work frequency and enabled the overlapping of different functions. This means that the following function starts to work as soon as there is needed data from the previous function, without waiting for the previous one to finish. However, the final optimization has 25% more resource costs than the previous solution. The comparison of resources is shown below.

Solutions	BRAM	FF	DSP	LUT
Unoptimized	2%	1%	0	1%
Loop merge	2%	1%	0	1%
Partition	2%	1%	0	7%
Pipeline&Dataflow	5%	4%	1%	25%

Table 4.11: Comparison of resources utilization

It's no surprise that the LUT incurs the highest cost. When having multiple pipeline stages, the control logic and data dependency may require additional

hardware to manage it effectively. Additionally, applying pipeline directives will unroll all sub-loops or sub-functions, resulting in increased hardware utilization.

Solutions	Throughput
Requirement	$61.32 MSamples/s$
Unoptimized	$18.3 MSamples/s$
Loop merge	$29.6 MSamples/s$
Partition	$33.32 MSamples/s$
Pipeline&Dataflow	$98.98 MSamples/s$

Table 4.12: Comparison of throughput

In hardware design, the amount of work completed in a given amount of time, known as throughput, can be affected by two factors: the design's latency and the initiation interval (II). However, latency can be reduced by implementing loop merge, while pipeline and dataflow can decrease the II. Based on the information presented in Table 4.11 and Table 4.12, it can be inferred that by increasing resource utilization, the throughput can be improved up to 5.4 times compared to the unoptimized solution. This highlights the need to balance throughput and utilization in hardware design.

Chapter 5

Design Test on PYNQ and Future Work

5.1 Loop-back Verification on PYNQ

5.1.1 PYNQ Frame Structure

This design includes a Programmable Logic (PL) UE and a Processing System (PS) RX for loop-back verification. The PS is a processing system based on dual ARM Cortex A9 cores, which integrates internal memory and external memory interfaces, and a large number of peripherals, GPIO, UART, SD/SDIO, IIC, SPI, Ethernet, CAN, etc. The PL part is a programmable logic unit based on the Xilinx 7 series architecture. Through the PL part, many peripheral overlays can be customized for PS [18]. The PYNQ PS/PL interface has several kinds of ports, there are 2 Master GP ports, 2 Slave GP ports, 4 Slave HP ports, 1 Slave ACP port, and GPIO for simple control signals and interrupts on the PS side. Furthermore, the PS/PL interface utilizes four ways of communication,

- General Purpose Input Output (GPIO)
- Memory Mapped IO (MMIO)
- Direct Memory Access (DMA)
- Memory Allocation

GPIO can only support a small amount of bits or bytes of data. DMA is commonly used in larger chunks of transfer. Virtual memory in Linux-based PS is used to implement buffers for data transmission to and from DMA overlays. MMIO can read and write the memory-mapped location especially suited to the scenario for a small amount of data. In this thesis, the control signal is transmitted using GPIO, and the UE inputs and outputs are transmitted via DMA.

The system architecture is shown above in Figure 5.1. Python implementation of RX on PS has the same architecture as UE but in reverse order. In the PL, the generated HLS IP is implemented with DMA. The input and results are transmitted between PS and PL with the AXI4 protocol. PS configures the UE through AXI GPIO IP. The user-id and mod-select signals control uplink pilots and modulation schemes respectively. After receiving the results from PL, the loop-back

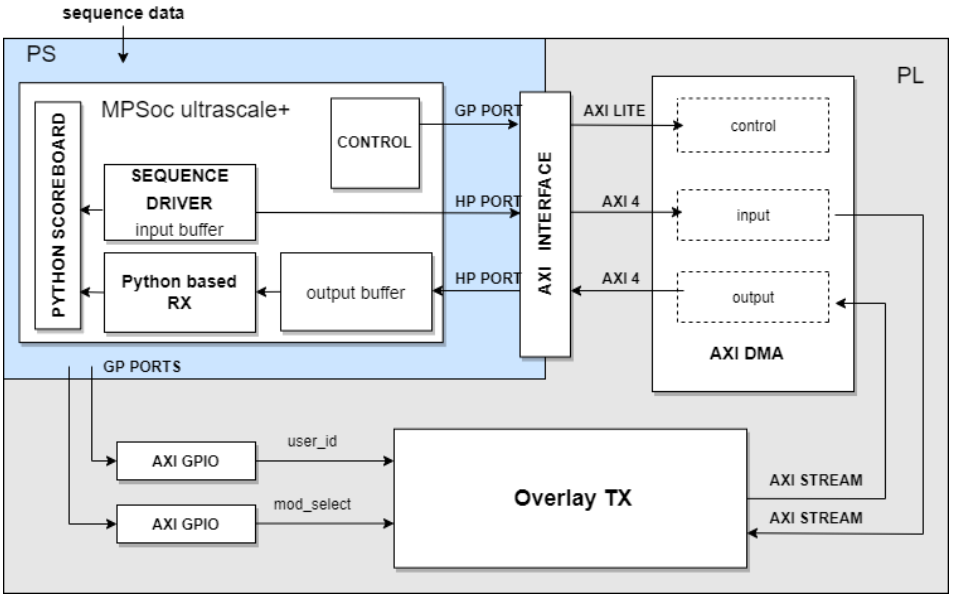
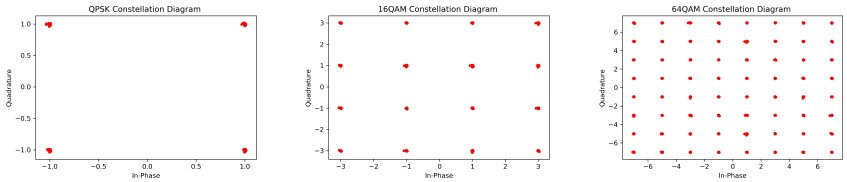


Figure 5.1: PYNQ frame structure.

verification should be applied to check the functionality. On the PS side, a scoreboard is used to compare results from RX and raw data. If there is a mismatch, the verification fails.

5.1.2 Reconfigurable Modulator Test

As shown in Figure 5.2, the modulated information is received signal on the PYNQ PS side. An external signal controls the reconfigurable modulator to perform three modulation schemes. Compared to Figure 2.7, QPSK has 4 dots while 16QAM and 64QAM have 16 and 64 dots respectively on the diagram.

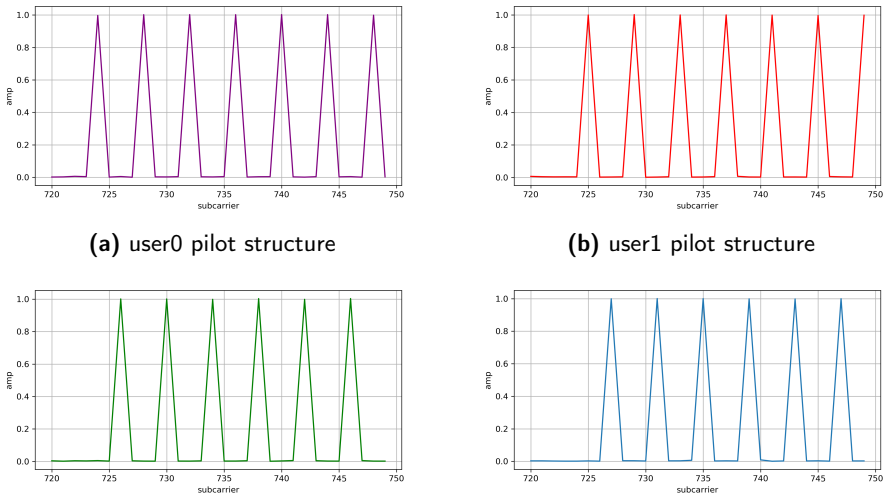


(a) QPSK result **(b)** 16QAM result **(c)** 64QAM result

Figure 5.2: Reconfigurable modulator test results.

5.1.3 Reconfigurable Pilots Test

The image in Figure 5.3 illustrates 150 reconfigurable pilot signals allocated to each of the 4 users. There are 600 pilots altogether, stored in a ROM. The partial subcarrier from 720 to 750 is shown in Figure 5.3, with user0's pilots starting from 724 and the remaining users following in sequence.



(a) user0 pilot structure

(b) user1 pilot structure

(c) user2 pilot structure

(d) user3 pilot structure

Figure 5.3: Reconfigurable pilot test results.

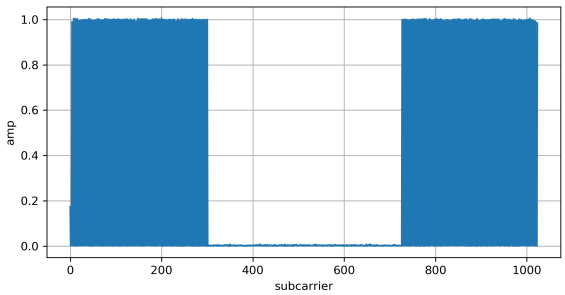
5.1.4 Pilot and Data Symbols

As shown in Figure 5.4(a), there are a total of 1024 subcarriers with 424 zeros inserted. The pilot signals are modulated with QPSK, resulting in an amplitude of approximately 1.0. Figure 5.4(b) to Figure 5.4(d) showed three types of modulated symbols. QPSK has a 1.0 amplitude, while 16QAM and 64QAM have different amplitudes due to modulation schemes.

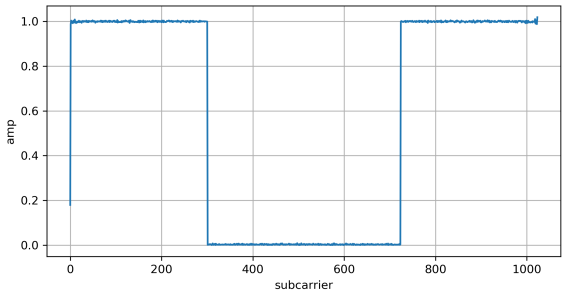
5.2 Conclusion&Future Work

Conclusion

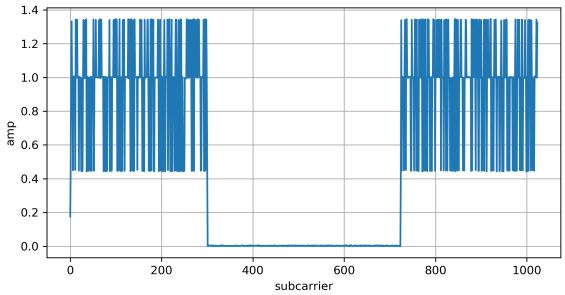
This thesis concludes by presenting the conclusion and discussing possibilities for future work. Specifically, the thesis focuses on the implementation of a reconfigurable UE on the PYNQ platform. The reconfigurable modulator offers three modulation schemes, namely: QPSK, QAM14, and QAM64, which can be switched using an external signal for reconfiguration. Additionally, the users' pilot signal



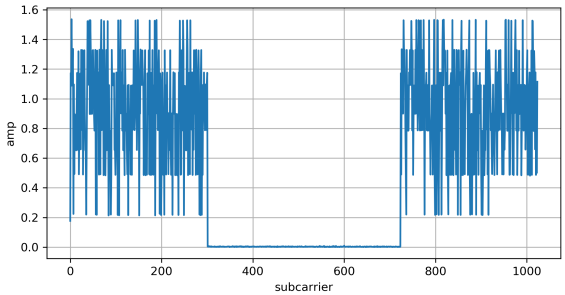
(a) pilot signal symbol



(b) QPSK data symbol



(c) 16QAM data symbol



(d) 64QAM data symbol

Figure 5.4: Pilots and data symbols.

that supports 4 UES separately is also controlled by an external signal. A Python-based RX and overlay TX are used to conduct loop-back verification on PYNQ, using DMA to transfer data between PS and PL. After multiple rounds of optimization, the design throughput can achieve up to 98.98 $MSamples/s$.

The high-level synthesis approach used in this design is a promising tool for future hardware design, with advantages over traditional RTL design in the following ways.

- High-level abstraction for design
- Faster prototyping
- Optimizing and parallelizing can be done easily
- Enhancing the efficiency of the development process.

Nonetheless, HLS needs relatively high demands on tools, and using it may require a certain level of expertise and experience. For instance, under the same directives, the latency of synthesized design decreased by 40% in Vitis HLS 2023.1 compared to that in VIVADO HLS 2017.4. While using HLS for circuit synthesis, it may be necessary to include specific directives to ensure proper control of the synthesis tool. Moreover, HLS may incur additional hardware costs when compared to RTL design. In such cases, the user may need to adopt unconventional solutions to address these issues effectively. Overall, HLS is a powerful approach to designing hardware but may require extra effort for users and higher compatibility with tools.

Future Work

Several aspects can be improved in the future.

- Theoretically, the Iteration Interval (II) can be reduced to 1, which could significantly improve throughput. However, due to limitations in the current version of Vitis HLS, pipelining the entire design is impractical.
- Better architecture for FFT and CP may be required to reduce latency and II due to data dependencies. This design can process large amounts of data simultaneously, but reducing latency can further improve performance and give a faster response.
- This design is based on the baseband, however, it is capable of implementation for up-conversion in the channel for further use.

References

[1] T. Hanninen, M. S. Saud, H. Y. Amin and M. Juntti, MIMO detector implementations using high-level synthesis tools from different generations, 2017 51st Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 2017, pp. 489-493, doi: 10.1109/ACSSC.2017.8335387.

[2] Chataut, R.; Akl, R. Massive MIMO Systems for 5G and beyond Networks—Overview, Recent Trends, Challenges, and Future Research Direction. Sensors 2020, 20, 2753. https://doi.org/10.3390/s20102753

[3] G. Akkad, A. Mansour, B. ElHassan, F. L. Roy and M. Najem, FFT Radix-2 and Radix-4 FPGA Acceleration Techniques Using HLS and HDL for Digital Communication Systems, 2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET), Beirut, Lebanon, 2018, pp. 1-5, doi: 10.1109/IMCET.2018.8603064.

[4] Shuangnan Liu, Francis CM Lau, and Benjamin Carrion Schafer. 2019. Accelerating FPGA Prototyping through Predictive Model-Based HLS Design Space Exploration. In Proceedings of the 56th Annual Design Automation Conference 2019 (DAC '19). Association for Computing Machinery, New York, NY, USA, Article 97, 1–6. https://doi.org/10.1145/3316781.3317754

[5] P. Patil, M. R. Patil, S. Itraj and U. L. Bomble, A Review on MIMO OFDM Technology Basics and More, 2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC), Mysore, India, 2017, pp. 119-124, doi: 10.1109/CTCEEC.2017.8455114.

[6] What is Beamforming? everything RF. (n.d.). https://www.everythingrf.com/community/whatisbeamforming.

[7] Trick C. 4G vs LTE vs 5G: What's the Difference? Trenton Systems. March 2023. https://www.trentonsystems.com/blog/4g-vs-lte-vs-5g.

[8] Misha. What is the Difference Between 5G NR and 4G LTE? https://blog.router-switch.com/2020/06/what-is-the-difference-between-5g-nr-and-4g-lte/. Published October 16, 2023.

[9] Solved Given the QPSK constellation diagram shown below, | Chegg.com. Available at: https://www.chegg.com/homework-help/questions-and-answers/given-qpsk-constellation-diagram-shown-using-matlab-plot-qpsk-signal-given-bit-rate-10mbps-q69179863.

[10] Hen-Geul Yeh. Figure 5. 64-QAM Signal Constellation with Gray Coding. Figure 4. 16-QAM Signal Constellation with Gray Coding. ResearchGate. https://www.researchgate.net/figure/64-QAM-Signal-Constellation-with-Gray-Coding_fig5_4260156.

[11] D. D. Gajski and L. Ramachandran, Introduction to high-level synthesis, in IEEE Design & Test of Computers, vol. 11, no. 4, pp. 44-54, Winter 1994, doi: 10.1109/54.329454.

[12] W. Altoyan and J. J. Alonso, Investigating Performance Losses in High-Level Synthesis for Stencil Computations, 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Fayetteville, AR, USA, 2020, pp. 195-203, doi: 10.1109/FCCM48280.2020.00034.

[13] P. Coussy, D. D. Gajski, M. Meredith and A. Takach, An Introduction to High-Level Synthesis, in IEEE Design & Test of Computers, vol. 26, no. 4, pp. 8-17, July-Aug. 2009, doi: 10.1109/MDT.2009.69.

[14] Patrick Lysaght. (n.d.). Xilinx research labs FPGA21 RFSoC PYNQ Tutorial. In https://www.rfsoc-pynq.io/pdf/isfpga_Rfsoc_2x2_overview.pdf.

[15] Pervej MF, Sarkar MZI, Roy TK, Hasan MdM, Rahman MdM, Bain SK. Analysis of PAPR reduction of DFT-SCFDMA system using different sub-carrier mapping schemes. International Conference on Computer and Information Technology (ICCIT). December 2014. doi:10.1109/iccitechn.2014.7073067

[16] Dan Boschen. Radix-2 vs Radix-4 FFT. Signal Processing Stack Exchange. https://dsp.stackexchange.com/questions/68375/radix-2-vs-radix-4-fft.

[17] Yue zhen. Vivado HLS (High-level Synthesis) note5, for loop optimization_vivado simulation for the loop-CSDN blog. https://blog.csdn.net/h_ang/article/details/90116641.w

[18] PS/PL Interfaces — Python productivity for Zynq (Pynq). https://pynq.readthedocs.io/en/v3.0.0/overlay_design_methodology/pspl_interface.html.