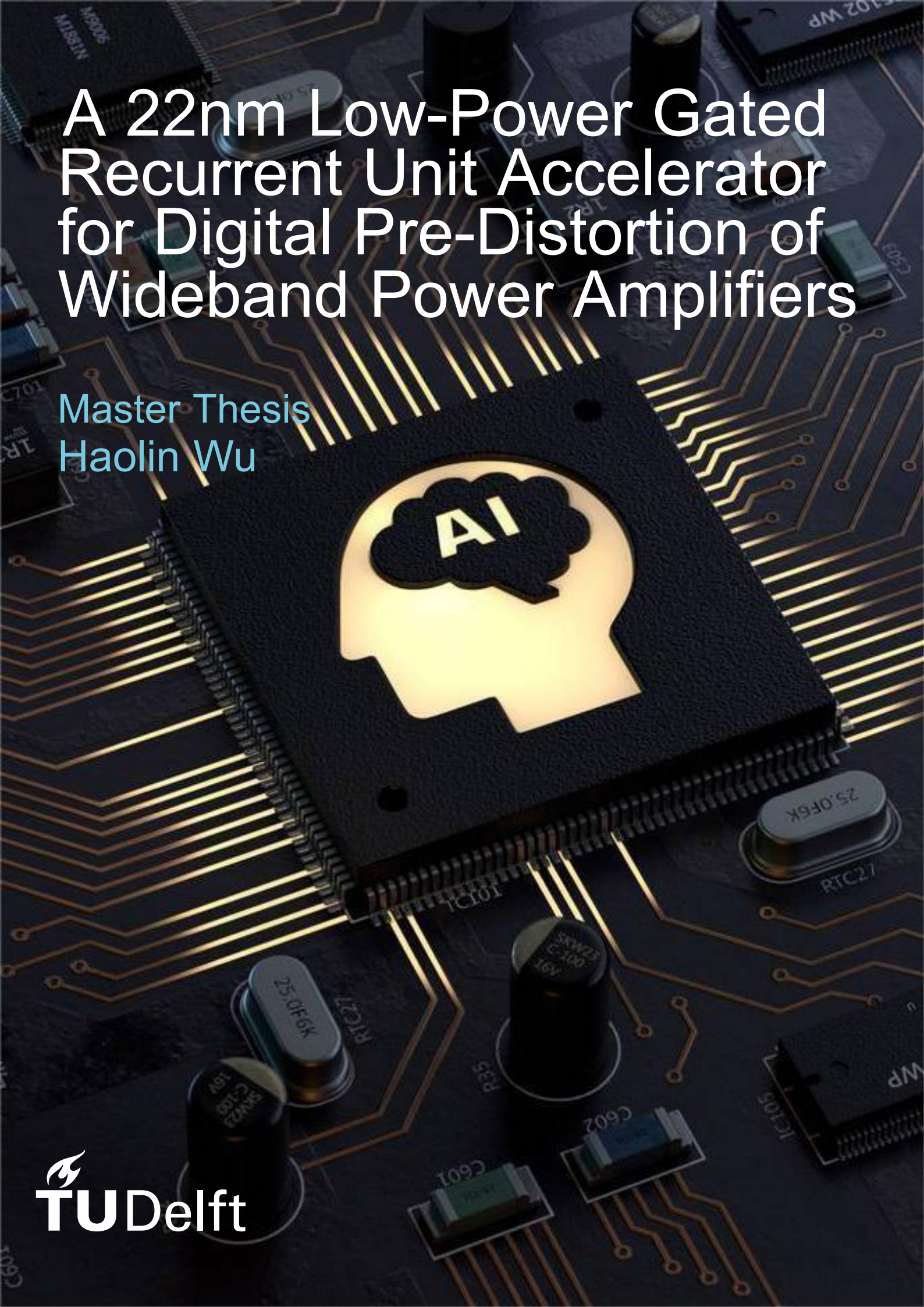
A 22nm Low-Power Gated Recurrent Unit Accelerator for Digital Pre-Distortion of Wideband Power Amplifiers

Master Thesis
Haolin Wu



TU Delft

A 22nm Low-Power Gated Recurrent Unit Accelerator for Digital Pre-Distortion of Wideband Power Amplifiers

Master Thesis

by

Haolin Wu

to obtain the degree of Master of Science
at the Delft University of Technology

Student number: 5706505
Project duration: November, 2023 – June, 2024
Thesis committee:

Prof. Dr. Ir. Leo de Vreede,	TU Delft, chairman
Dr. Ir. Chang Gao,	TU Delft, supervisor
Dr. Ir. Rajendra Bishnoi,	TU Delft

This thesis is confidential and cannot be made public until December 31, 2025.

TU Delft

Acknowledgments

First of all, I am very glad to study at TU Delft. Over the past two years, I have learned a lot in digital integrated circuit design. I believe that the knowledge and experience gained during these two years will significantly contribute to my future success.

Then, I would like to thank my supervisor, Dr. Chang Gao, for his guidance and support throughout this work. Dr. Gao often provides me with insightful suggestions, guiding me step by step to find solutions to problems. In addition, Dr. Gao encourages and comforts me when I am feeling down or suffer from setbacks. I am deeply thankful for the extensive knowledge and skills I have acquired through his mentorship.

Next, I would like to thank my two other master committee members, the chair, Prof. Leo de Vreede and Dr. Rajendra Bishnoi. I sincerely appreciate the professors for taking the time out of their busy schedules to present as members of the committee for my defense. I learned a lot on the course, hardware architecture for artificial intelligence, taught by Dr. Rajendra Bishnoi. The project, BNN accelerator has laid the foundation for my thesis.

Finally, I would like to thank my parents as well as my friends. Your support and companionship have made my life and study colorful and meaningful. And I would like to thank my senior, Ang Li and Yizhuo Wu, for the guidance on my thesis.

I hope I will become an excellent engineer and have a brilliant career.

Haolin Wu
Delft, June 2024

Abstract

As communication capacity continues to expand, the application of deep neural networks (DNNs) for digital pre-distortion (DPD) has become increasingly prominent in addressing non-linearity issues in wideband power amplifiers (PAs). The advent of the fifth-generation (5G) era imposes higher requirements on DPD regarding frequency and latency. The integration of multiple-input multiple-output (MIMO) technology and micro base stations has driven the trend towards low-power, small-area DPD chips. This paper presents a high-performance, Gated Recurrent Unit (GRU)-based hardware architecture, characterized by high parallelism, and low resource consumption, enabling real-time signal processing by DPD. A novel method is proposed, employing quantization-aware training (QAT) with Hardsigmoid and Hardtanh functions to quantize the floating-point model in software. The optimized algorithm is implemented on hardware with inter-layer pipelining and retiming to optimize timing and increase clock frequency. Additionally, hardware-efficient linear functions, Hardsigmoid and Hardtanh, are utilized for activation functions to minimize hardware overhead. Experimental results demonstrate that hardware implementation achieves an Adjacent Channel Power Ratio (ACPR) of **49.48 dBc** and an Error Vector Magnitude (EVM) of **46.05 dB**, showin minimal degradation compared to the floating-point model (**49.58 dBc**/ **46.70 dB**). Simulated under 22nm CMOS technology, the DPD chip, operating at **2 GHz**, occupied an area of **0.047 mm^2** and is capable of handling signals with bandwidth up to **70 MHz**. The highest throughput reaches **256.5 GOp/s** while the power efficiency reaches **1.3154 TOp/s/W**.

Contents

Nomenclature

CDMA Code Division Multiple Access

WCDMA Wide-band Code Division Multiple Access

OFDM Orthogonal Frequency Division Multiplexing

VR Virtual Reality

ADC Analog-to-Digital Converter

AM Amplitude Modulation

FM Frequency Modulation

QAM Quadrature Amplitude Modulation

RF Radio Frequency

SoC System on Chip

MIMO Multiple-input Multiple-output

NMSE Normalized Mean Squared Error

ACPR Adjacent Channel Power Ratio

EVM Error Vector Magnitude

MP Memory Polynomial

GMP Generalized Memory Polynomial

TDNN Time Delay Neural Network

RVTDNN Real-Valued Time-Delay Neural Network

VDLSTM Vector Decomposed Long Short-term Memory

SVDLSTM Simplified Vector Decomposed Long Short-term Memory

DVR Decomposed Vector Rotation

IoT Internet of Things

FPGA Field Programmable Gate Array

ASIC Application Specific Integrated Circuit

PWL Piece-wise Linear Approximation

QAT Quantization-aware Training

DNN Deep Neural Network

AI Artificial Intelligence

CPU Central Processing Unit

DSP Digital Signal Processing

LSTM Long Short-term Memory

PSK Phase Shift Keying

PA Power Amplifier

DPD Didital Pre-Disotrtion

GRU Gated Recurrent Unit

LUT Lookup Table

RNN Recurrent Neural Network

FDMA Frequency Division Multiplexing Access

TDMA Time Division Multiple Access

1 Introduction

1.1. Motivation

With the advancement of technology, wireless communication has surpassed the limitations of voice calls. Innovations and applications in millimeter waves, micro base stations, and MIMO technologies have expanded the bandwidth of the 5G wireless communication technology, increasing peak data transmission rates over 10 Gbps or higher [2] [33]. The integration of these advancements into daily life has facilitated the proliferation of the Internet of Things (IoT), mobile payments, smart wearables, and live streaming. Meanwhile, these emerging industries demand extremely low latency from wireless communication systems. To achieve high coverage and high-speed communication, 5G base stations are becoming increasingly miniaturized and energy-efficient [35]. In these base stations, wireless mobile communication relies on PAs to amplify small signals for transmission via antennas. PAs consume the majority of the energy in this process [20], making their efficiency and power consumption critical concerns for 5G communication systems.

However, the linearity and efficiency of PAs are often in conflict. When a PA operates in its linear region, there is a constant gain between its input and output. As the signal power increases, the non-linearity of the PA causes the signal gain to decrease, significantly deviating from the ideal output. This introduction of non-linearity leads to spectral regrowth, which can interfere with adjacent channel communications and cause in-band distortion, increasing the bit error rate and degrading communication quality. Sacrificing efficiency to achieve linearity by backing off the PA to its linear operating region is an impractical solution for real-world communication systems. To address this issue, DPD technology is employed. DPD processes the digital signal before it enters the PA, using a model that inversely mirrors the PA's non-linear characteristics to compensate for the non-linearity. This method has found widespread application in wireless communication systems due to its effectiveness in improving PA performance without sacrificing efficiency.

Designing a high-performance DPD hardware circuit for communication systems characterized by low latency and low power consumption is of significant importance. However, traditional PA models, such as the Volterra series model and the Generalized Memory Polynomial (GMP) model, face significant challenges as we move into the 6G era. With the widening frequency bands, mathematical modeling of wideband PAs requires an increasing number of parameters, struggling to meet real-time requirements. Additionally, the hardware overhead and power consumption associated with these models have become substantial.

The rapid development of deep learning, which excels in addressing non-linearity, offers a novel approach to solving DPD issues. Given the GRU model's ability to handle sequential data efficiently and its relatively low parameter count, this thesis proposes a low-power digital hardware accelerator of a GRU Recurrent Neural Network (RNN)-based artificial intelligence (AI)-DPD. This approach aims to leverage the strengths of GRU in managing temporal dependencies while minimizing hardware complexity and power consumption, providing an innovative solution for modern wireless communication systems.

1.2. Problem Description

Currently, research and implementations are predominantly focused on the software level, and applying GRU neural networks to DPD is a novel direction. In hardware circuit design, floating-point calculations are complex operations. As a result, parameters are often quantized to reduce the computational overhead and simplify the critical path when designing GRU hardware accelerators. In DPD, the digital signals to be processed are normalized IQ-modulated signals. Directly applying the quantized trained model and input data during inference can significantly impact the accuracy of the computations. Ensuring the linearization performance of the GRU model after quantization is a key challenge in developing a high-performance DPD hardware circuit.

GRU neural networks exhibit high computational complexity. Serial computation, as performed on Central Processing Units (CPUs), cannot meet the system's data processing frequency requirements. An efficient and well-designed system architecture, including memory units, computational units, and control logic, is crucial for optimizing the critical path, increasing the system clock frequency, and reducing data processing interval. Such an architecture ensures that the GRU-based DPD hardware can effectively handle the high demands of real-time communication systems.

A high-performance GRU hardware architecture entails highly parallel computations. Consequently, the bit-width of intermediate results significantly impacts the area and power consumption. Designing an efficient data flow is one of the primary goals of this design. Moreover, lookup tables (LUTs) for activation functions with large bit-widths exacerbate the area overhead, particularly in such highly parallel designs. Therefore, it is crucial to find an efficient hardware-friendly quantization method for activation functions in this design. This ensures that the implementation remains feasible in terms of area and power while maintaining the performance of the GRU-based DPD system.

1.3. Objective

The main objectives of this work are as follows:

- In software design, use a QAT quantization strategy to quantize the float-point model with mixed precision. Ensure that the quantized model has negligible linearization performance loss compared to the original model.
- In software design, compare the accuracy of the model using LUTs to quantize the activation function and that of the model using the Hardsigmoid/Hardtanh function to quantize the activation function.
- In hardware design, design a suitable hardware microarchitecture and data flow of GRU computation to increase throughput and reduce data processing cycles.
- In hardware design, compare the resource utilization and power consumption of the LUTs method and that of the Hardsigmoid/Hardtanh function method.
- In hardware design, optimize the critical path to increase the frequency.

1.4. Contributions

The main contributions of this work are as follows:

- Using QAT quantization strategy to quantize the floating-point model: The ACPR of the quantized model with a format of **Q2.10** on hardware can reach **-49.48 dBc**, which has slight loss compared with the ACPR of the floating-point model, **-49.58 dBc**.
- Proposing a method to quantize the activation function in AI-DPD: The method of using the Hardsigmoid/Hardtanh function to replace the LUTs for quantizing the activation function significantly reduces the area. In this work, the resource utilization reaches **18.85x** reduction for the sigmoid function and **35.25x** reduction for the tanh function compared to implementation with LUTs method.
- Proposing a highly parallel structure utilizing retiming and inter-layer pipeline to implement the GRU neural network: The chip can operate at **2GHz** with **7.5ns** latency simulated under 22nm

CMOS technology. The area of the chip is **0.047 mm^2** and the power consumption reaches **194.726mW**. What is more, the throughput and the power efficiency of the chip are **256.5 GOp/s** and **1.3154 TOp/s/W**, respectively.

1.5. Outline

The structure of this thesis is shown as follows:

- Chapter 1 introduces the motivation of this work, the objectives to be achieved, and the contribution made.
- Chapter 2 provides the background of DPD. Firstly, the wireless communication and transmission chain are introduced briefly. Then, the chapter discusses the non-linearity of PA in detail, including two different types of distortion and evaluation criteria. Meanwhile, some traditional and state-of-the-art methods for DPD are elaborated. Finally, the chapter analyzes the method of acceleration of the RNN algorithm on hardware in terms of frequency and power consumption.
- Chapter 3 outlines the implementation of the GRU model in both software and hardware design. This includes the quantization and proposed Hardsigmoid/Hardtanh function to quantize activation functions on software as well as the design of the structure, data stream, and individual submodules on hardware.
- Chapter 4 presents the experimental setup of software and hardware and analyzes the experimental results, and provides corresponding evaluations.
- Chapter 5 concludes the thesis and lists possible future work.

2 Background

2.1. Brief History of Wireless Communication

Over the past few decades, wireless communication technology has rapidly advanced, becoming an indispensable component of modern technology and daily life. The first generation of wireless communication technology (1G) emerged in the 1980s [39], employing Frequency Division Multiplexing Access (FDMA) and a cellular mobile network architecture. Due to technological limitations, signals were transmitted through analog modulation, and the primary service offered was voice communication. However, this generation suffered from low spectral efficiency and high susceptibility to interference, resulting in limited capacity and unstable call quality. The second generation of wireless communication technology (2G) marked the transition from analog to digital modulation [39]. Core technologies such as Time Division Multiple Access (TDMA) and Code Division Multiple Access (CDMA) enhanced the stability and reliability of communication systems [11]. Building on the advancements of 2G, the third generation (3G) introduced Wideband Code Division Multiple Access (WCDMA), significantly improving data transmission rates and enabling the transmission of multimedia data (images, audio, video) [5], thereby enriching users' lives and entertainment experiences. The advent of the fourth generation (4G) brought about further enhancements with Orthogonal Frequency Division Multiplexing (OFDM) technology [37], which increased the ideal peak data transmission to several hundred Mbps. This paved the way for the proliferation of applications such as live streaming and mobile payments.

In recent years, the rise of the IoT, autonomous driving, and virtual reality (VR) has driven the development of the 5G of wireless communication technology. To meet the growing demand for faster data transmission rates and larger system capacity, 5G wireless communication technology extends carrier frequencies to the Sub-6GHz band (450MHz-6000MHz) and the mmWave band (24GHz-100GHz) [33]. With the integration of MIMO [45] and Beamforming technology, data transmission speed can reach 10Gbps and beyond [2] [33]. Consequently, the higher data transmission rates and ultra-low latency provided by 5G facilitate applications such as edge computing, and remote medical care. However, these advancements also pose significant challenges for processing ultra-wideband signals, particularly concerning the real-time performance and power consumption of chips.

2.2. Signal Transmission Chain

The propagation of information in wireless communication system is a highly complex process, shown in Figure 2.1, which mainly comprises the following processes:

- Signal Generation: As the initial stage of information transmission, original signals can be categorized into two types: analog signals and digital signals. Digital signals primarily originate from devices such as Digital Signal Processor (DSP) and computers, while analog signals are mainly generated by sensors, microphones, and similar devices.
- Sampling, Quantization, and Encoding: In modern communication systems, to ensure robustness against interference and operational flexibility, information is transmitted as digital signals. Analog signals, such as those used in telephone communications, must be sampled, quantized, and encoded via an Analog-to-Digital Converter (ADC) in the baseband.

- Modulation: Electromagnetic waves with lower frequencies are easily absorbed by the Earth's surface and cannot effectively propagate. Additionally, the low-frequency wireless spectrum is limited, potentially causing signal congestion. Considering antenna design, where the length is proportional to the wavelength of radio signals (which is inversely proportional to frequency), low-frequency signal transmission complicates the manufacture of base stations and antennas [49]. Hence, high-frequency modulation becomes essential. Common modulation techniques include Amplitude Modulation (AM), Frequency Modulation (FM), Phase Shift Keying (PSK), and Quadrature Amplitude Modulation (QAM). QAM decomposes the signal into two orthogonal components, sine (I) and cosine (Q), and modulates them separately, enhancing spectrum utilization and adjacent channel interference immunity [46]. QAM is extensively employed in 5G mobile communication systems.
- Radio Frequency (RF) Transmission: After modulation, the signal's power remains relatively low and is inadequate for transmission via the antenna. Therefore, the signal requires amplification by PAs to achieve sufficient RF power to drive the antenna. RF transmission represents a significant portion of the total power consumption in the signal propagation process [20].
- Propagation and Reception: The RF signal, once radiated through the antenna, propagates through the air to reach the receiving end. Upon arrival, the signal undergoes processing steps including filtering, amplification, and demodulation, ultimately converting the RF signal back into its original form.

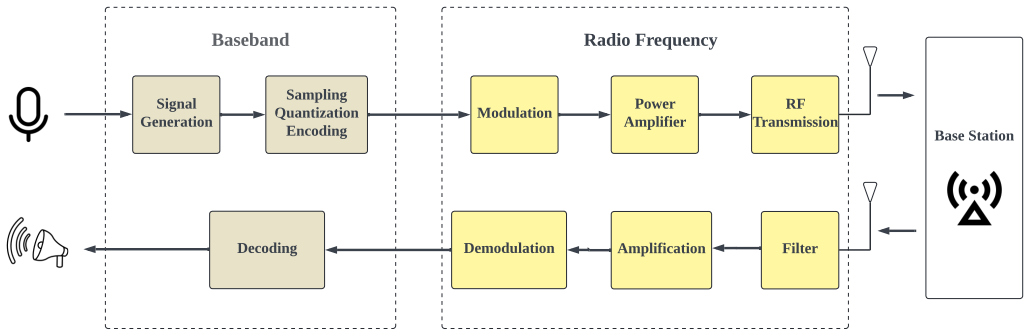


Figure 2.1: Propagation of information in wireless communication system

The advent of 5G communication has introduced advanced technologies such as millimeter-wave, beamforming, and MIMO. These innovations have led to a denser deployment of base stations and network nodes, transitioning from a central base station-centric model to one that is more customer and device-centric [36] [35]. Consequently, smaller base stations have assumed more critical roles in the network infrastructure. This shift necessitates a significant reduction in power consumption and cost for these small base stations, making the design of high-performance System on Chips (SoCs) with low power and low latency a top priority.

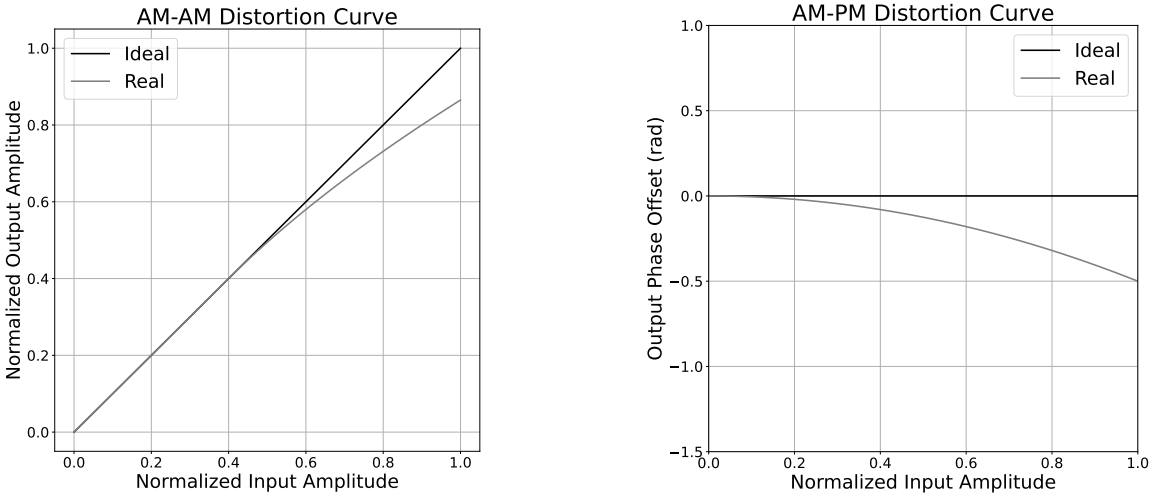
2.3. Non-linearity of PA

In wireless communication systems, PAs are essential components responsible for amplifying small signal power to levels suitable for antenna transmission, thus ensuring effective signal propagation. Ideally, PAs should provide constant gain, achieving linear amplification of the input signal. However, due to the inherent non-linear characteristics of PAs, non-linear distortion arises when operating in the saturation region. As the input signal power increases, this non-linearity causes the actual output to deviate from the ideal gain curve, resulting in distortions. Specifically, this manifests as amplitude-amplitude (AM-AM) and amplitude-phase (AM-PM) distortions in the time domain [32]. In the frequency

domain, on one hand, the spectrum of output is broadened. The non-linearity of PA induces high-order intermodulation of the transmitted signal during amplification, causing spectral components to spill into adjacent frequency bands, adversely affecting the communication quality of adjacent channels. On the other hand, the nonlinearity induces in-band signal distortion [17], which results in significant interference within the communication signal, thereby increasing the in-band bit error rate.

2.3.1. Memoryless Distortion

Memoryless distortion in PAs refers to the distortion of the output signal being solely dependent on the current input signal. When the input signal amplitude is low, the PA operates in the linear region, and its output gain can be approximately considered constant. However, as the input signal amplitude increases, in the AM-AM characteristic curve, shown in Figure 2.2a, the gain of the PA is compressed with increasing input power, deviating from the ideal linear curve and bending downwards. In the AM-PM characteristic curve, shown in Figure 2.2b, this manifests as output signal phase shift compressing with increasing input power.



(a) AM-AM (b) AM-PM

Figure 2.2: Conceptual AM-AM and AM-PM characteristics of PAs with memoryless distortion

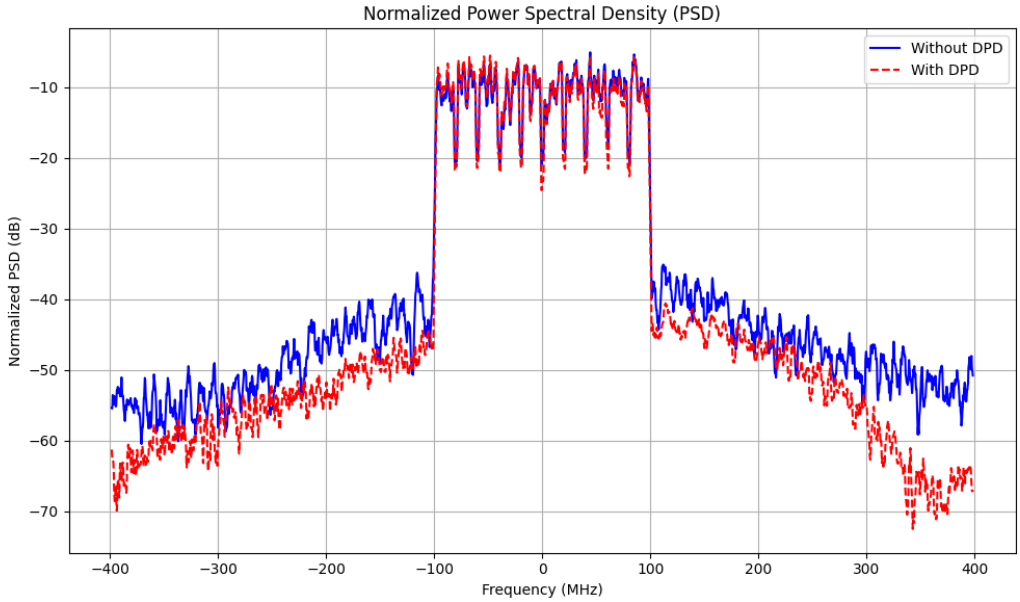
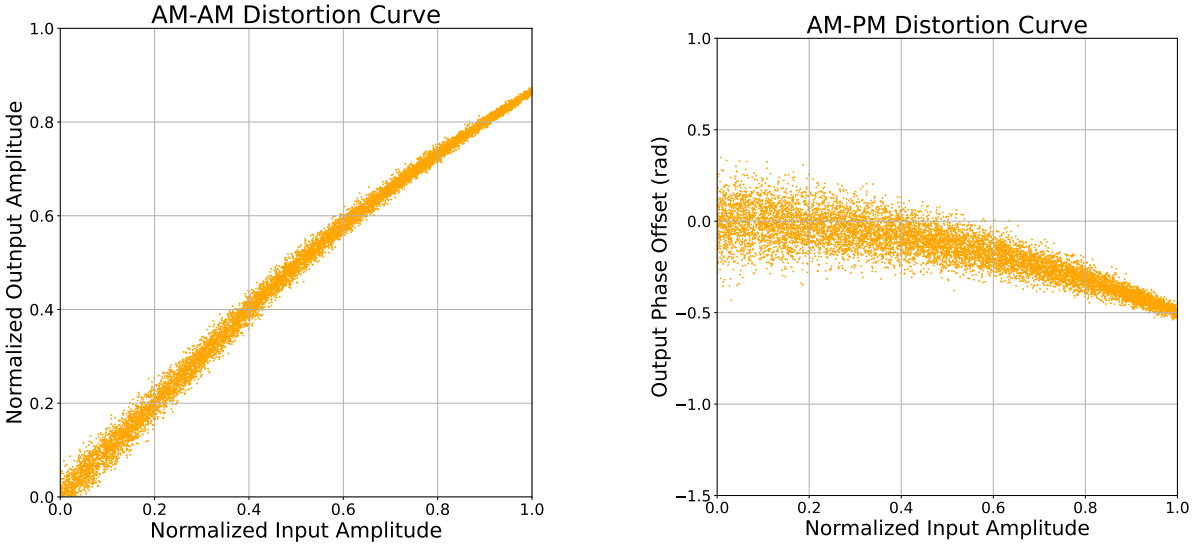


Figure 2.3: A power spectral density plot example showing the spurious emission of PA output without DPD vs. with DPD.

In the frequency domain, shown in Figure 2.3, the non-linearity of the PA introduces direct current components, second harmonic, and third harmonic into the output signal [42] [17], leading to intermodulation distortion between signals of different frequencies in the system. This distortion is evidenced in the spectrum graph as a degradation of flatness within the signal band and significant spectral expansion outside the band.

2.3.2. Memory Distortion

Memory distortion in PAs refers to the phenomenon where the output signal is influenced not only by the current input signal but also by previous input signals, indicating the presence of a time correlation between the amplifier's output and input. With the increase in bandwidth and circuit complexity, thermal effects generated by electronic components such as capacitors, inductors, and transistors have become the primary sources of memory distortion in PA [17]. In the time domain, the memory effect is manifested as variations in output gain and phase for input signals of the same amplitude.



(a) AM-AM

(b) AM-PM

Figure 2.4: Conceptual AM-AM and AM-PM characteristics of PAs with memory distortion

AM-AM distortion curve in Figure 2.4a and AM-PM Distortion Curve in Figure 2.4b exhibit dispersion characteristics, where the degree of divergence in the data points reflects the strength of the PA's memory effect. A more pronounced divergence in the characteristic curve indicates a stronger memory effect, whereas a lesser divergence suggests a weaker memory effect. Similarly, in the frequency domain, memory distortion further exacerbates signal spectrum expansion.

2.3.3. Evaluation Criteria

The evaluation metrics for PAs non-linearity primarily include Normalized Mean Squared Error (NMSE), Adjacent Channel Power Ratio (ACPR), and Error Vector Magnitude (EVM).

Normalized Mean Squared Error

The NMSE reflects the magnitude of the discrepancy between the actual output of the PA and the ideal output. A smaller value indicates a closer proximity of the actual output to the desired value. The mathematical expression is shown as follows:

$$NMSE = 10\lg \frac{\sum_{n=0}^{N-1} \left|y(n) - y'(n)\right|^2}{\sum_{n=0}^{N-1} |y(n)|^2} \tag{2.1}$$

The equation defines $y(n)$ as the actual output signal of the PA, $y'(n)$ as the ideal output signal of the PA and N as the number of samples taken.

Adjacent Channel Power Ratio

The definition of ACPR is the ratio of adjacent channel power to the main channel power, mainly reflecting the impact of amplifier non-linear distortion on adjacent channels. A smaller value indicates lower non-linear distortion and less interference between adjacent channels. The mathematical expression is shown as follows:

$$ACPR = 10\lg\left(\frac{\int_{adj} PSD(w)dw}{\int_{main} PSD(w)dw}\right) \tag{2.2}$$

In the equation, $PSD(w)$ represents the power spectral density (PSD) function, 'adj' denotes the adjacent frequency range, and 'main' indicates the main channel frequency range.

Error Vector Magnitude

The EVM quantifies the average power of the difference between the actual output and the ideal output signals, relative to the average power of the ideal output signal. It serves as a metric for assessing the quality of modulation signals. A smaller value indicates lower distortion and higher signal quality. The mathematical expression is shown as follows:

$$EVM = \sqrt{\frac{\sum_{n=1}^{N}\left((I'n - In)^2 + (Q'n - Qn)^2\right)}{\sum_{n=1}^{N}\left(In^2 + Qn^2\right)}} \tag{2.3}$$

In the equation,In and Qn are the I/Q components of the ideal output signal. While $I'n$ and $Q'n$ are the I/Q components of the actual output signal. N is the length of the input vector signal.

2.4. Digital Pre-Distortion

In wireless communication systems, the RF front end consumes the majority of the power [20]. To improve efficiency, it is desirable for PAs to operate near the saturation region. Therefore, addressing the non-linearity that arise near saturation becomes crucial. This challenge has led to the development and implementation of DPD technology.

2.4.1. Principle

The principle of DPD involves passing the input signal through a mathematical model that inversely mirrors the non-linear characteristics of the PA before it reaches the PA. The output from this mathematical model is then used as the new input to the PA. By cascading these two stages, the non-linear distortion caused by PA is counteracted, resulting in linear amplification of the output signal relative to the input signal. This mathematical model is referred to as the digital pre-distorter. Figure 2.5 illustrates the process of DPD. The graph demonstrates that the output characteristic curve of the digital pre-distorter is inverse to that of PA, ensuring that the product of their gains remains constant [32].

In the graph, V_{in}, V_D and V_{out} represent the input signal, the output signal after digital pre-distorter and the output signal amplified by PA. $F(x)$ denotes the function of digital pre-distorter, while $G(x)$ represents the function of the PA. The overall system function is expressed as follows:

$$V_{out} = G(F(V_{in})) \tag{2.4}$$

2.4.2. Conventional Methods

The conventional DPD model needs to model the non-linearity of the PA at first, which involves fitting the input-output relationship of the PA. Based on the results of this modeling, non-linear compensation is subsequently applied to the PA to mitigate the distortion effects.

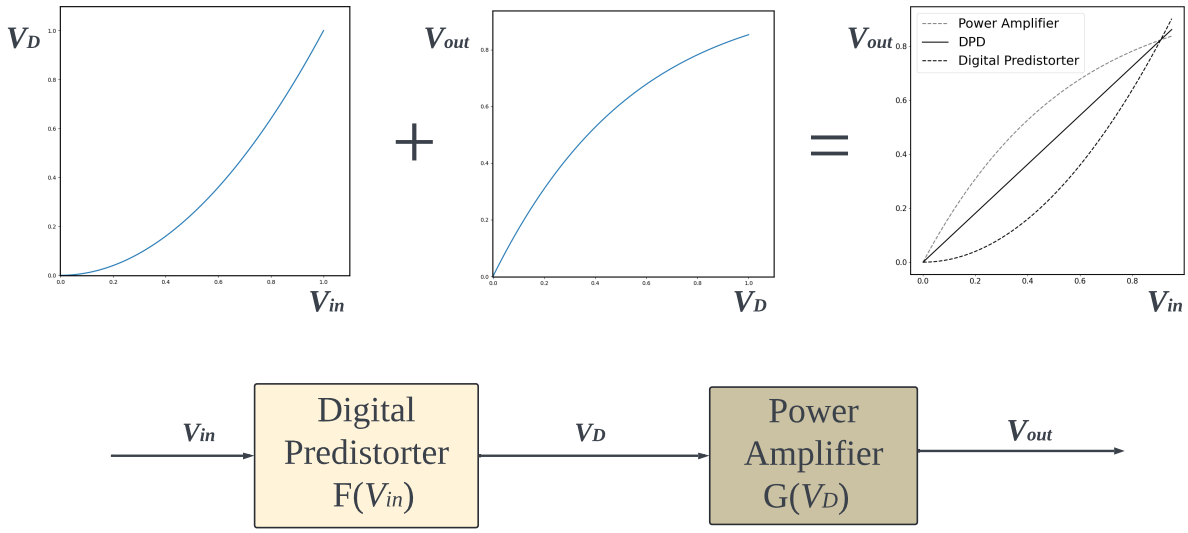


Figure 2.5: Process of DPD

Volterra Series Model

The Volterra series [14] model is a widely used mathematical model for describing the non-linearity of PAs. This model employs polynomial expansions to characterize the system's non-linear behavior. Unlike the Taylor series, the Volterra series incorporates higher-order terms and delay memory terms, effectively capturing the memory relationship between the current output of the system and past inputs. In discrete non-linear time-invariant systems, the Volterra series can be expressed as follows:

$$\begin{aligned} y_{vol}(n) = & \sum_{q_1=0}^{Q} h_1(q_1)x(n-q_1) \\ & + \sum_{q_1=0}^{Q}\sum_{q_2=0}^{Q} h_2(q_1,q_2)x(n-q_1)x(n-q_2) \\ & + \cdots \\ & + \sum_{q_1=0}^{Q}\sum_{q_2=0}^{Q}\cdots\sum_{q_k=0}^{Q} h_k(q_1,q_2,\cdots q_k)x(n-q_1)x(n-q_2)\cdots x(n-q_k) \end{aligned} \tag{2.5}$$

In the equation, $x(n)$ and $y_{vol}(n)$ represent the input and output functions of the system, respectively. Q denotes the memory depth, k is the non-linearity order, h_k represents the k-th order non-linear transfer function, and it can be fitted using the least squares method.

However, as the memory depth and non-linearity order increase, the parameters in the Volterra series exhibit exponential growth, leading to a substantial rise in computational complexity and resource requirements. This escalation presents significant challenges for hardware design in subsequent DPD implementations, rendering the Volterra series model impractical for modern wireless communication systems characterized by wide bandwidth and strong non-linearity.

Generalized Memory Polynomial Model

Some simplified models have been proposed to mitigate the challenges of excessive parameters and computational complexity in the Volterra model. The even-order non-linear components of the PA are far from the center frequency and can be filtered using bandpass filters, while the odd-order non-linear components primarily influence PA non-linearity. Based on this observation, a simplified model, known

as the Memory Polynomial (MP) model, is proposed by Bell Telephone Laboratories [24], which retains only the diagonal terms of the Volterra series. It can be expressed as:

$$y_{MP}(n) = \sum_{k=1}^{K} \sum_{q=0}^{Q} a_{k,q} x(n-q) \left| x(n-q) \right|^{k-1} \tag{2.6}$$

Among them, K is the non-linear order, Q is the memory depth, and a is the coefficient of the polynomial.

While the simplification of the Volterra series model reduces the parameter count and simplifies the structure, it also results in a loss of precision due to the omission of cross-terms. Consequently, the MP model is inadequate for strongly non-linear systems with wide bandwidth. To address this limitation, the GMP model has been proposed [30]. The GMP model retains both the diagonal and cross-terms of the MP model, thereby enhancing model accuracy. Its expression is as follows:

$$\begin{aligned} y_{GPM}(n) = & \sum_{m=0}^{M_a} \sum_{k=1}^{K_a} a_{mk} x(n-m) \left| x(n-m) \right|^{k-1} \\ & + \sum_{m=0}^{M_b} \sum_{k=2}^{K_b} \sum_{p=1}^{P} b_{mkp} x(n-m) \left| x(n-m-p) \right|^{k-1} \\ & + \sum_{m=0}^{M_c} \sum_{k=2}^{K_c} \sum_{q=1}^{Q} c_{mkp} x(n-m) \left| x(n-m+q) \right|^{k-1} \end{aligned} \tag{2.7}$$

Compared to the MP model, although the GMP model exhibits increased computational complexity, its significant improvement in accuracy has led to widespread application in hardware and engineering.

2.4.3. Deep Learning Methods

Although traditional DPD mathematical models excel in mathematically and physically analyzing systems and can achieve optimal performance through manual parameter tuning, the increasing complexity of PA non-linearity with wideband signals in modern wireless communication systems presents significant challenges. The complexity in modeling and the subsequent rise in computational requirements render traditional methods less practical. In recent years, neural networks have gained prominence due to their robust fitting capabilities, making them highly applicable in modeling non-linear systems. This advancement offers a new direction for DPD research.

RVTDNN Model

The Time Delay Neural Network (TDNN) is the pioneer neural network model applying to PA [34]. A neural network is composed of neurons, each consisting of several inputs and a single output. The Real-Valued Time-Delay Neural Network (RVTDNN) is a multi-input multi-output neural network structure employed in DPD [18]. The network comprises input, hidden, and output layers. It takes the real (I) and imaginary (Q) components of signals at different time instances as inputs to the neural network model. Through processes such as weighted summation, bias addition, and activation in multiple hidden layers, the values of the signals I and Q at the current time instance are outputd. The network structure is illustrated in Figure 2.6 and Figure 2.7.

The accuracy of modeling PA non-linearity is closely linked to both the number of neurons in the hidden layer and the depth of the hidden layers. Generally, increasing the number of neurons and the depth of the hidden layers enhances accuracy. However, this also results in higher hardware resource costs and longer network training times.

RNN Model

RNN is a type of neural network characterized by feedback connections, capable of handling sequential data while maintaining internal states during processing [29] [9] [23]. This property confers inherent advantages to RNN models in modeling PA non-linearity. Two notable models, the vector decomposed long short-term memory (VDLSTM) model and the simplified vector decomposed long short-term memory (SVDLSTM) model, have been applied to DPD [27]. Another research has highlighted the efficacy

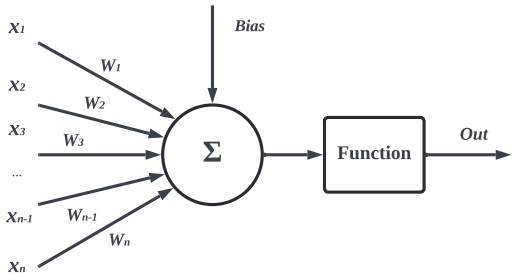


Figure 2.6: Schematic diagram of a neuron

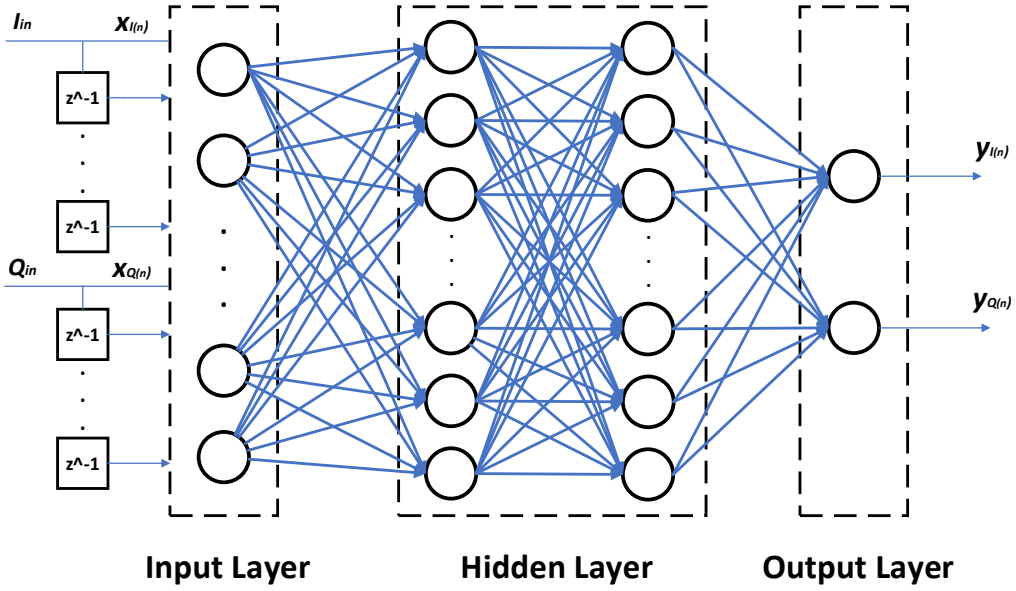


Figure 2.7: Basic structure of TDNN

of RNN models over traditional GMP models in addressing PA non-linearity. [26]. Additionally, the decomposed vector rotation (DVR) method has been proposed to enhance linearization performance [25].

An RNN neural network consists of the input layer, hidden layer, and output layer, as depicted in Figure 2.8. In the structure, U, W and V represent the weight matrices, X_t is the input at the current time step, S_{t-1} is the hidden state from the previous time step, and O_t is the output at the current time step. The update of the hidden state S_t depends on both the current input X_t and the previous hidden state S_{t-1}. This relationship can be described by the following equation:

$$O_t = g(V * S_t) \tag{2.8}$$

$$S_t = f(U * X_t + W * S_{t-1}) \tag{2.9}$$

As mentioned in previous chapters, RNN neural network models have a natural advantage in handling time-series signals. However, during the process of training, issues such as exploding gradients and vanishing gradients may arise [38]. To address the challenges of long-term dependencies, variants of RNN, such as the Long Short-term Memory (LSTM) and GRU structures, have been developed [38].

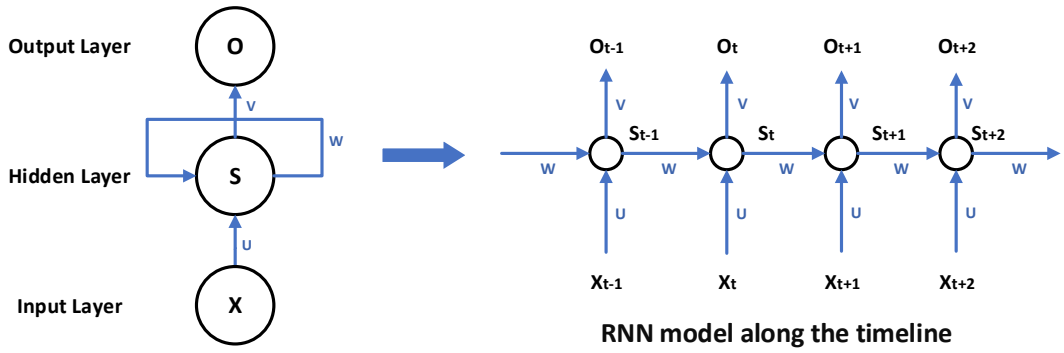


Figure 2.8: The standard RNN and unfolded RNN.

The GRU structure, an optimized version of the LSTM structure, features fewer parameters and a simpler architecture while maintaining comparable accuracy [10].

2.4.4. OpenDPD

Y. Wu et al. [48] propose an open-source end-to-end deep learning model and DPD framework to address PA non-linearity. The study also introduces a dataset similar to MNIST, which serves as a benchmark for evaluating DPD models. This approach primarily comprises three steps:

- Data Extraction: The IQ-modulated signals at different frequencies are passed through the input and output of the PA and the digital transmitter. These signals are then extracted to form the training datasets.
- Train PA Model: Given the varying non-linear characteristics of PAs in different experimental environments, multiple models are applied to train the PA. The results are compared to establish a foundation for subsequent DPD training.
- Train DPD Model: The previously trained PA model remains unchanged. The DPD model is then trained by cascading it with the PA model, and the training results are evaluated based on specific criteria (NMSE, ACPR, EVM).

Experimental results indicate that after 100 epochs of training, the Dense Gated Recurrent Unit (DGRU) model, comprising a single GRU neural network layer and two fully connected layers, achieves an ACPR of $-44.69/-44.47dBc$ and an EVM of $-35.22dB$, making it the most outstanding performer among various neural network models.

The quantized model of the DGRU also demonstrates excellent performance. Y. Wu et al. [47] propose a low-complexity mixed-precision quantization model to reduce the computational complexity and energy consumption of floating-point models while maintaining acceptable precision and accuracy. Experimental results show that quantizing floating-point numbers to integer types has a minimal impact on output results and significantly reduces power consumption. When the weights and inputs of the GRU model are quantized to 16-bit integers, the ACPR reaches $-43.75/-45.27dBc$, which is nearly comparable to the FP32 results of $-43.36/-45.30dBc$. Furthermore, the energy consumption and power usage of the quantized model are $1.13nJ$ and $0.72W$, respectively, compared to $2.94nJ$ and $1.88W$ for the float model, representing a 60% reduction. This study provides theoretical support and a basis for hardware implementation of SoC.

2.5. DPD Digital Circuit Implementation

When designing DPD systems, it is imperative to develop an appropriate hardware architecture capable of processing signals efficiently and accurately in real-time while also minimizing computational complexity and power consumption. Inspired by the experimental results of OpenDPD, the GRU neural network demonstrates significant advantages in terms of precision, parameter quantity, and complexity. Consequently, the objective of this paper is to design a GRU-based DPD digital circuit. Before

embarking on the design of a hardware system, it is essential to comprehend factors that influence power consumption and frequency, as well as structures of accelerators and quantization strategies from previous studies. Additionally, it is necessary to know about the implementation platform of the hardware system.

2.5.1. Frequency and Power Efficiency

Clock frequency is an important metric for measuring hardware performance, as it determines the maximum operating frequency of the chip. If excessive computations are performed within a single clock cycle, long combination logic chains can lead to timing violations in the early register-transfer level stage [13], thereby reducing the clock frequency. Given a clock frequency f_{clk} and a processing interval of n cycles, the data processing frequency can be calculated as:

$$f_{data} = \frac{f_{clk}}{n} \tag{2.10}$$

There exists a constraint relationship between f_{clk} and n. Generally, an increase in the processing interval may positively impact the clock frequency. However, f_{data} may not increase proportionally. Additionally, an increase in clock frequency will also lead to higher power consumption. Therefore, in the design of DPD hardware systems, balancing the relationship between clock frequency, processing interval, and data processing frequency is crucial to ensure power efficiency.

Power consumption is another essential metric when designing hardware circuits. The power consumption of digital circuits can be divided into two parts: static power and dynamic power. Static power refers to the power consumed when there are no activities in the circuit [15]. This power consumption is typically due to leakage current in transistors and is related to the production and the number and types of components [15]. Dynamic power, on the other hand, refers to the power consumed by the transition activity of signals inside the circuit. Its consumption is influenced by factors such as frequency, power supply voltage, and load capacitance and can be expressed as follows:

$$P_{dynamic} = \alpha C v^2 f \tag{2.11}$$

C stands for capacitance, V is the supply voltage, f is the clock frequency, and α is the dynamic power factor related to process and design parameters. Therefore, it is crucial to optimize circuit structure to ensure that a high-performance hardware system operates at a very high frequency.

In addition, simplifying workload and reducing complexity can fundamentally reduce power consumption and simplify circuit structure. The relationship between dynamic power consumption and computation is extensively analyzed in previous research [47], as expressed in the formula:

$$E = E_{MUL} + E_{ADD} + E_{MEM} \tag{2.12}$$

Therefore, reducing multiply-accumulate operations and memory accesses can effectively reduce power consumption.

Furthermore, in deep learning model frameworks, most models use floating-point precision to ensure accuracy. However, floating-point multiply-accumulate computations are less hardware-friendly compared to fixed-point operations. Within an acceptable range of precision loss, deep learning models using fixed-point arithmetic demonstrate excellent performance on hardware.

F. Fang et al. [12] compare the area and power of hardware implementations of inverse discrete cosine transform (IDCT) with floating-point precision, lightweight floating-point precision, and fixed-point precision. The results illustrate that fixed-point implementation occupies $36905um^2$, which is almost half of the lightweight floating-point implementation and one-fifth of the floating-point implementation.The power consumption is $12.6mw$, which is approximately 70% of that of lightweight floating-point implementation and 23% of that of floating-point implementation.

Y. F. Tong et al. [41] find that the accuracy remains nearly the same level even if the bit-width decreases when the initial number is sufficiently large. However, when the bit-width is small, the reduction will lead to a significant drop in accuracy. It is also noted that energy consumption has an approximate linear relationship with bit-width. Other studies have found that the data width in hardware significantly impacts power consumption and frequency. M. van Baalen et al. [3] observe that reducing weight bit-width has a slight influence on the ResNet18 model's accuracy, dropping only from 70.34 to 69.73, while power consumption decreases dramatically by 17.5%. He also finds that with the same power

consumption, mixed precision obtains 0.5% more accuracy than fixed bit-width, while under the same accuracy, mixed precision consumes 4% less power than fixed bit-width.

Therefore, balancing accuracy and bit width is an important part of designing a deep-learning hardware system.

2.5.2. Method of Acceleration on Hardware

Multi-processing unit and Calculate in Parallel

In CPUs, the scarcity of computing resources often necessitates sequential execution of most programs, significantly affecting processing efficiency. In contrast, Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA) boast abundant logical and computing resources, enabling parallel computation as their primary feature. By leveraging parallel computation, tasks that require multiple cycles on a CPU can be completed in fewer cycles on an FPGA or ASIC, thereby enhancing performance.

For instance, summing four numbers on a CPU involves sequential calculations, requiring three cycles to complete. On an FPGA or ASIC, the process is optimized by dividing the four numbers into two groups, calculating each group in parallel, and then summing the intermediate results. This approach reduces the total number of cycles to two. As the quantity of numbers increases, the reduction in cycles becomes even more pronounced. The processes on a CPU and FPGA/ASIC are shown in Figure 2.9.

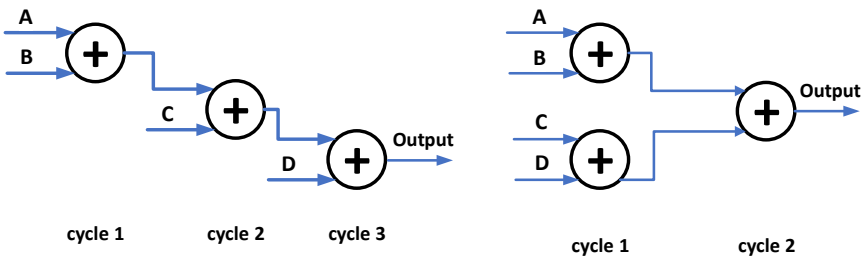


Figure 2.9: Process of calculation on CPU and FPGA/ASIC

Thus, the key to designing a GRU-based DPD hardware accelerator lies in enhancing computational parallelism, which reduces data processing cycles and increases data processing frequency.

Time Multiplexing

In FPGA/ASIC design, it is crucial to improve the utilization of limited resources. Additionally, simpler circuits with lower resource overhead can significantly reduce chip power consumption. Time multiplexing is a technique that allows multiple functions to share the same hardware resource at different time intervals, thereby optimizing resource utilization.

For the GRU model, the computations for the reset gate, update gate, candidate memory cell, and hidden state involve numerous matrix multiplications and additions with similar computational structures. Moreover, the calculations for these gates cannot be fully parallelized. Given these characteristics, time multiplexing can be employed to optimize circuit structure and reduce hardware resource usage. By scheduling the computations of these gates at different time intervals, the same hardware resources can be reused, leading to more efficient use of FPGA/ASIC resources.

Pipeline

Pipeline design involves dividing a large-scale, multi-layer combination logic into several stages and inserting register sets at each stage to store intermediate data temporarily. With inputs fed continuously from the start of the pipeline, each stage performs its operations in an overlapping manner. A K-stage pipeline contains exactly K register sets from the input to the output of the combination logic (divided into K stages, each with one register set). The output of the previous stage serves as the input for the next stage, without feedback loops.

The application of pipelines can enhance the system's parallelism and throughput, reduce the worst negative slack (WNS), and increase the clock frequency. However, the performance improvement from pipeline design comes at the cost of increased register resource usage.

The pipeline design method is illustrated in the Figure 2.10. In this example, a task consists of three stages A, B, and C, each requiring one cycle to execute. When executed serially, two instances of this task require 6 cycles to complete. While using a pipeline, the same two tasks can be accomplished in only 4 cycles.

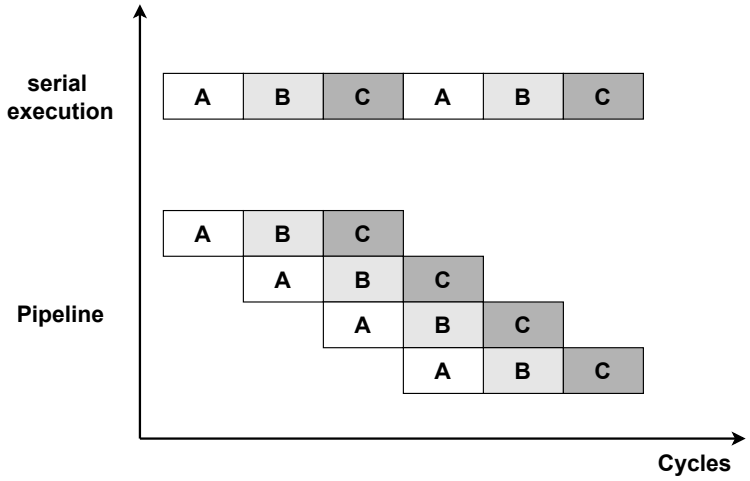


Figure 2.10: Pipeline

Memory Hierarchy

Efficient storage design can also improve hardware processing performance. In digital circuits, data transfer is a time-consuming and resource-intensive activity. As a result, reducing data transfer and the number of computing unit accesses can significantly lower the overall power consumption of the chip. Hardware storage can be categorized into three major types: cache, SRAM, and DRAM. These categories differ in terms of access speeds and storage capacities, with cache having the fastest access speed but the smallest capacity, followed by SRAM, and then DRAM with the largest capacity but the slowest access speed. Thus, the strategy of storage utilization is a critical aspect of hardware design.

2.5.3. RNN Hardware Accelerator

Model Compression and Quantization

Neural network models are usually large and parameter-heavy to achieve optimal performance, which presents challenges for hardware implementation in terms of resource utilization, power consumption, and timing. Model compression can reduce the number of parameters in a neural network model while maintaining its original performance, thereby alleviating the storage burden on the model's hardware accelerator. Model compression techniques include data quantization and data compression.

Data quantization refers to converting floating-point numbers to fixed-point numbers. This method saves storage space and simplifies the hardware multiplication structure. This paper utilizes this approach, and details will be elaborated on in the next chapter.

Data Compression, particularly through network pruning, is another common technique. Neural network parameters often include redundant or relatively small values. Ignoring the connections corresponding to these neurons has a minimal impact on the final result, as illustrated in Figure 2.11. Pruned parameters can be stored as sparse matrices in hardware. Utilizing sparse matrix multiplication instead of dense matrix multiplication can enhance performance.

S. Han et al. [19] propose a load-balanced-aware pruning method that compresses the LSTM model size by 20× while maintaining prediction accuracy. Experimental results show that the design implemented on FPGA exhibits energy efficiency 40 times greater than that of a CPU and 11.5 times greater than that of a GPU.

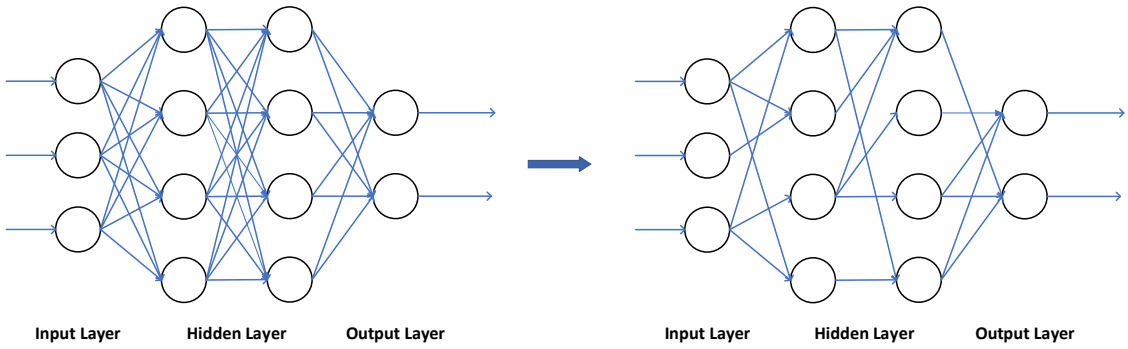


Figure 2.11: Pruning of the neural network

Activation Function Implementation

The computation of the activation function presents a significant challenge in the hardware implementation of neural networks. The activation function is a non-linear transcendental function that cannot be expressed using a finite number of algebraic operations (addition, subtraction, multiplication, division). Therefore, its hardware implementation is highly complex and resource-intensive. Two methods are proposed to address this issue: LUTs and PWL.

The LUTs method involves defining an appropriate input range for the activation function and selecting several discrete points within this range. It is important to note that the input range and the number of points depend on the precision of the model; a larger range and more points result in higher precision but also greater resource consumption. These inputs and their corresponding outputs are stored in the LUT, which functions similarly to read-only memory. Each input acts as an address, and the output corresponds to the content stored at that address. This method trades off space for time.

The PWL method approximates the non-linear activation function using several segmented linear functions. The number and the determination of these linear functions mainly depend on the model's precision requirements. The advantage of the PWL method is that each linear function can be implemented using shifts and additions on hardware. However, if the number of linear functions is too large, it may lead to significant resource consumption. An example of a PWL approximation of the sigmoid function is as follows:

$$f(x) = \begin{cases} 1, & x \geqslant 5 \\ \frac{x}{32} + \frac{27}{32}, & \frac{19}{8} \leqslant x < 5 \\ \frac{x}{8} + \frac{5}{8}, & 1 \leqslant x < \frac{19}{8} \\ \frac{x}{4} + \frac{1}{2}, & -1 \leqslant x < 1 \\ \frac{x}{8} + \frac{3}{8}, & -\frac{19}{8} \leqslant x < -1 \\ \frac{x}{32} + \frac{5}{32}, - & 5 \leqslant x < -\frac{19}{8} \\ 0, & x < -5 \end{cases} \tag{2.13}$$

M. T. Ali et al. [1] compare the implementation of the tanh(x) function on FPGA using the LUTs, PWL, and polynomial methods. The experimental results show that the LUT method is the most efficient, with resource consumption approximately one-third that of the PWL method and one-eighth that of the polynomial method. Additionally, E. Guillena et al. [18] employ PWL to approximate the sigmoid function on FPGA-based LSTM accelerator, achieving a calculation speed increase of 20.18 times compared with CPU processing.

Structure
In deep learning models, matrix multiplication is often large in scale, resulting in significant time and hardware resource consumption for memory access and arithmetic operations. The Delta RNN Algorithm presents a novel structure to address this issue by reducing the computational load. The Delta RNN Algorithm uses the change Δx between the current input x_t and the previous input x_{t-1} as the new input for the GRU model. A threshold is set, where Δx values smaller than the threshold are treated as zero, while larger values are retained. The Delta RNN Algorithm implies that when the input change is small, its impact on the output can be ignored.

Additionally, LSTM networks exhibit a highly skewed distribution of computational complexity among the primitive operators [43]. Therefore, it is crucial to design a practicable structure that balances multiplication, addition, and other operations.

D. Neil et al. [31] apply the Delta RNN Algorithm to the TIDIGITS audio digit recognition benchmark. Experimental results shows that the Delta RNN Algorithm reduces computational energy consumption by a figure of 9 without any loss in model accuracy. C. Chang et al. [16] accelerate GRU networks on an FPGA using the Delta RNN Algorithm, achieving an effective throughput of 1.2 TOp/s and a power efficiency of 164 GOp/s/W, and a 5.7x speedup compared to a conventional RNN. S. Wang et al. [43] propose the coarse-grained pipelines to split the long combination logic into small segments, thereby accelerating the LSTM model on hardware.

2.5.4. Related Work

Currently, the hardware implementation of DPD predominantly relies on conventional models, with very few studies exploring the implementation of DPD using deep learning models. And the majority of DPD hardware designs are implemented on FPGA platforms. H. Lin et al. [28] propose a low-cost hardware implementation scheme for DPD based LUT method. H. Chen et al. [7] propose a low-complexity joint-polynomial-and-LUT pre-distortion PA linearizer, maintaining a low computational complexity and an excellent ACPR. H. Hunag et al. [21] introduce a parallel processing DPD architecture, which overcomes the bandwidth constraints imposed by the maximum clock frequency of the FPGA. The results illustrate that the hardware implementation can achieve a total linearization bandwidth of 1.25 GHz. T. Cappelo et al. [6] design a scalable DPD on FPGA. The clock frequency and bit can be scaled up to 250 MHz and 16 bits. The research explores the trade-offs between the power consumption and the linearization performance in DPD system.

3

Proposed Methodology

This chapter will illustrate the software-hardware co-design methodology of this work from the software implementation to hardware implementation.

3.1. Software Implementation

The basic framework of this work consists of a GRU neural network and a fully connected neural network. Four hardware-friendly features are extracted as inputs to the neural network. To simplify the computation and storage on hardware, quantization of weight and bias is introduced. Additionally, to meet low power consumption requirements and reduce resource usage on hardware, a quantization method for the activation function is proposed.

The overall structure of DPD is shown in Figure 3.1, with our work focusing on the pre-distorter component. The framework of this work consists of three layers: the input-processing layer, the GRU layer, and the FC layer, as depicted in Figure 3.2. In this work, four features are extracted, namely I_x, Q_x, $I_x^2 + Q_x^2$ and $(I_x^2 + Q_x^2)^2$. The GRU neural network is configured with an input feature size of 4, a hidden size of 10, and a single hidden layer. The 10 outputs from the GRU layer are then mapped to 2 outputs through a fully connected layer. These two output are the results of the DPD exploiting the GRU neural network.

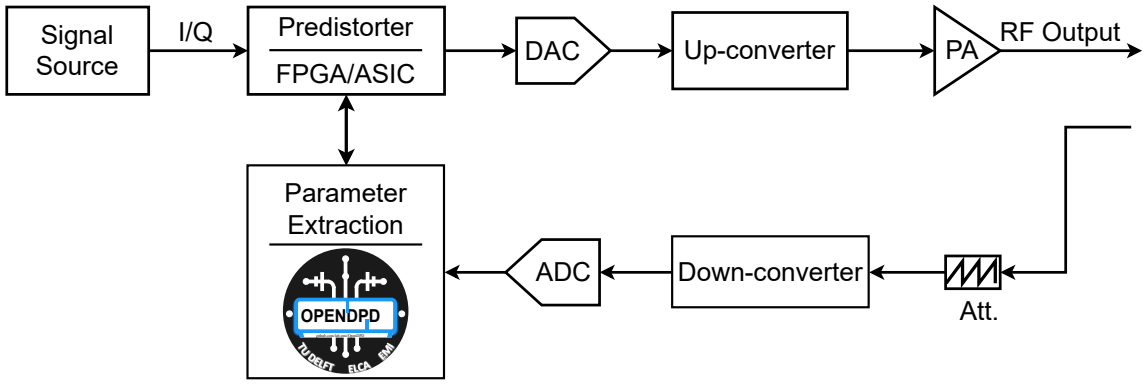


Figure 3.1: Simplified block diagram of DPD structure

3.1.1. GRU Architecture

This work prioritizes minimizing the number of model parameters and reducing complexity to design a high-performance digital circuit. Due to its efficient structure, the GRU model offers distinct advantages over other neural network types. The core idea of the GRU model is to introduce non-linear gating units that regulate the connections between the current and previous outputs, thereby mitigating issues of vanishing and exploding gradients.

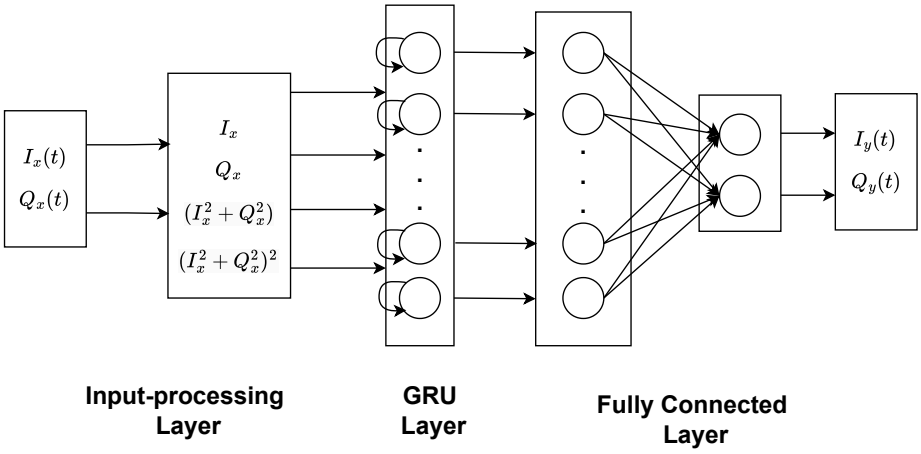


Figure 3.2: The GRU-based DPD architecture

Compared to traditional RNN models, GRU adds two gating structures and a candidate memory unit, namely reset gate (rt), update gate (zt), and candidate memory unit (nt). The structure of the GRU model is illustrated in Figure 3.3.

- Update Gate (zt): The update gate influences the hidden state, determining the extent to which the previous time step's hidden state affects the current time step. Its value ranges from 0 to 1, with values closer to 0 indicating a greater influence from the previous time step and less from the current time step, and vice versa.
- Reset Gate (rt): The reset gate updates the candidate memory unit of the current neuron, indicating the influence of the previous time step's hidden state on the current candidate cell state. Its value also ranges from 0 to 1, with values closer to 0 indicating less influence from the previous time step.
- Candidate memory unit (nt): This unit stores memory information, updated based on the reset gate's value.

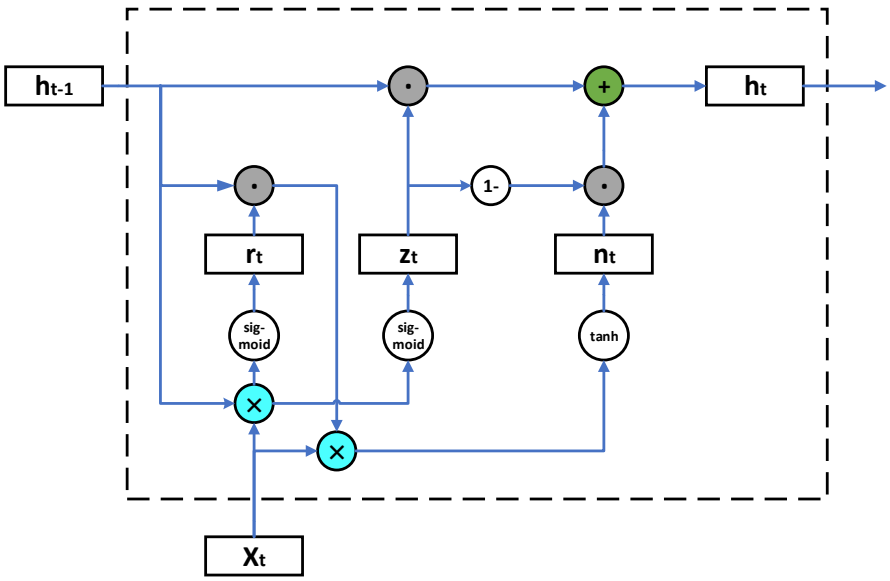


Figure 3.3: GRU neural network structure

The GRU model can be represented by the following mathematical formulas [8]:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \tag{3.1}$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \tag{3.2}$$

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{t-1} + b_{hn})) \tag{3.3}$$

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{t-1} \tag{3.4}$$

In the formulas, W_{ir}, W_{iz}, and W_{in} are input-hidden weights, while W_{hr}, W_{hz}, and W_{hr} are hidden-hidden weights. Similarly, b_{ir}, b_{iz}, and b_{in} are input-hidden biases, while b_{hr}, b_{hz}, and b_{hn} are hidden-hidden biases. The symbol $\odot$ denotes the Hadamard product, σ represents the sigmoid activation function.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.5}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.6}$$

3.1.2. Feature Extraction

Feature extraction from initial data is essential for deep-learning neural networks. While a greater number of input features often leads to higher accuracy, they also increase model complexity and result in redundant parameters. This complexity poses significant challenges for hardware design. For DPD, efficiently extracting relevant information from the original signal is a key challenge. This work employs IQ modulation for the signal, where the I and Q components inherently contain a significant amount of information within the frequency band. Additionally, most of the waveform's characteristics at the current time can be derived from these two components, such as sine signals, cosine signals, and amplitude. This can be expressed as follows :

$$|x| = \sqrt{I_x^2 + Q_x^2} \tag{3.7}$$

$$sin\theta_x = \frac{I_x}{\sqrt{I_x^2 + Q_x^2}} \tag{3.8}$$

$$cos\theta_x = \frac{Q_x}{\sqrt{I_x^2 + Q_x^2}} \tag{3.9}$$

In OpenDPD, the selected features are I_x, Q_x, $|x|$, $|x|^3$, $sin\theta_x$, $cos\theta_x$ [48], while in MP-DPD, the selected features are I_x, Q_x, $|x|$, $|x|^3$ [47]. The experimental results indicate that reducing the number of features with minimal impact on the outcome has little effect on accuracy. In this work, feature extraction not only considers software accuracy but also accounts for the cost and overhead of hardware computation. Therefore, sine and cosine features are discarded due to the cumbersome nature of division calculations on hardware. Direct division consumes large resources and reduces the clock frequency. Alternatively, using a shift-based Cordic algorithm for calculation would require many computation cycles, leading to a decrease in data processing frequency.

Moreover, selecting only I_x and Q_x results in a dramatic decrease in accuracy, even if the hidden sizes is increased. Chapter result will show the results in detail. Therefore, in addition to the normalized I and Q signals, this work also selects the square and fourth power of amplitude, which are computationally simpler in hardware operations.

3.1.3. Data Quantization

As mentioned in the Chapter 2, quantization plays a crucial role in simplifying parameter storage and computation. In this work, weights, biases as well as intermediate variables are quantized. This section will elaborate on the conversion between floating-point and fixed-point numbers, and propose a method for quantizing floating-point numbers to fixed-point numbers.

According to the IEEE-754 standard [22], the format of a 32-bit single-precision floating-point number consists of three parts, shown in Figure 3.4. The first part, the sign bit, is a single-bit number,

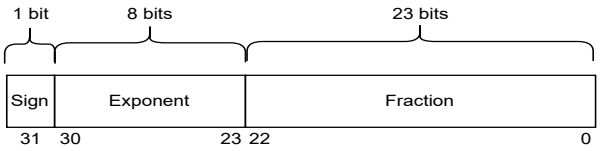


Figure 3.4: Format of 32-bit single-precision floating-point number

either 0 or 1, indicating a positive or negative sign of the number. The second part, the exponent, is an 8-bit binary number representing the magnitude of the number. The exponent uses the 'offset-binary' representation, requiring subtraction of the bias during index calculation. The third part, the fraction, represents the mantissa. The conversion formula of a floating-point number from binary to decimal for a floating-point number is as follows:

$$X_{decimal} = (-1)^{S} \times 2^{E-127} \times (1 + F) \tag{3.10}$$

Due to the complexity of floating-point numbers, their computation in hardware involves decoding, arithmetic operations, alignment, and rounding, which are cumbersome and cannot meet the requirements of this design. Therefore, it is necessary to quantify the weights and biases of the neural network in the software algorithm. Considering that the parameters of the floating-point model are signed numbers with a symmetric distribution of positive and negative values, this paper chooses symmetric quantization. Figure 3.5 shows the format of fixed-point number.

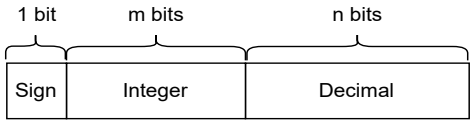


Figure 3.5: Format of fixed-point number

The format of fixed-point numbers also consists of three parts: the sign bit, the integer part, and the fractional part. In the figure, m is determined by the range while n depends on the precision of the number. Unlike floating-point numbers, the point is fixed in fixed-point numbers excluding an exponent and mantissa. Therefore, in hardware circuits, addition or subtraction can be performed directly once aligning the points. Compared to floating-point operations, many steps and circuit components are omitted. The quantization formula for converting floating-point numbers to fixed-point numbers can be represented as follows:

$$scale = 2^{-n} \tag{3.11}$$

$$\begin{aligned} X_{min} &= -2^{m+n} \\ X_{max} &= 2^{m+n} - 1 \end{aligned} \tag{3.12}$$

$$x_{floor} = \left\lfloor \frac{x}{scale} \right\rfloor \tag{3.13}$$

$$Q(x, scale) = min(max(x_{floor}, X_{min}), X_{max}) * scale \tag{3.14}$$

The key to the process is to right-shift the floating-point number to amplify it first, then round it, and finally left-shift to scale it down. The size of the scale depends on the number of valid bits after points, and the maximum and minimum values depend on the overall bit width. The floor function always quantizes the original value towards the smaller direction.

For example, to convert the floating-point number 1.53546 to a fixed-point number with a precision of 8 bits after the point and with an overall bit width of 9, the process is as follows:

- right shift: $1.53546 >> 8 => 1.53546/2^{-8} = 393.0776$

- floor: $393.0776 => 393$
- clamp:$min(max(393, -2^9), 2^9 - 1)$=>393
- left shift: $393 << 8 => 393 \times 2^{-8} = 1.53515625 = 10'b01.1000\_1001$

Moreover, every intermediate variable in the hardware structure will be subjected to quantization and rounding operations to save computational and storage units. The floor function is chosen to simplify the rounding logic. This aspect will be analyzed in detail in the hardware implementation section.

3.1.4. Activation Function Optimization

As discussed in the previous chapter, implementing activation functions through LUTs on hardware is a widely used method. The LUT method can be considered as a means of quantizing the output values of the activation function.

For example, assume the precision for both input and output is 8 bits after the point, with an overall bit width of 10. Given the input of $sigmoid(x)$ function is a fixed-point number $x = 00.0110\_0110$, the corresponding output as a floating-point number is 0.5983121949. And the binary representation after quantization is 00.1001_1001.

In the example, only one input value and its corresponding output are listed. For the entire $sigmoid(x)$ function, the size of the storage is significantly large, as depicted in Figure 3.6. The input serves as the address, ranging from 00_0000_0000 to 11_1111_1111, while the output values are 10-bit data. The storage size can be calculated with the following formula:

$$Storage\_Size = 2^{10} \times 10\ bit \tag{3.15}$$

Although the storage size seems acceptable, this design features a highly parallel hardware structure with very few model parameters. Using the LUTs method to quantize the activation function would consume significant resources and power on it rather than other logic. LUTs method are implemented in this work to serve as the baseline, and the results are presented in the next chapter. The paper proposes a method to quantize the activation function using Hardsigmoid/Hardtanh functions.

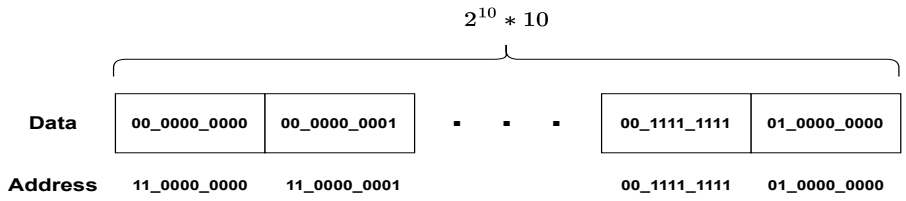


Figure 3.6: Storage of LUT method for sigmoid

In this work, we use the QAT quantization strategy for quantization, which primarily involves three steps:

- Step one: Train a floating-point neural network to obtain a baseline model.
- Step two: Insert fake quantization nodes on the baseline model and retrain the model. To be specific, this step involves quantization of weight, bias, and intermediate variables and retraining the model based on the previously saved parameters.
- Step three: Quantize the new parameters saved in the second step. These parameters are used for inference and need to be passed to the hardware.

In the second step, fake quantization nodes are inserted to retrain the model and fine-tune the parameters. During training, the model experiences a loss of accuracy while maintaining the same framework, and the parameters self-correct to approach or recover previous levels. Taking the simplest network as an example, the formulas of forward propagation and backpropagation are as follows.

$$\begin{aligned} \alpha &= W * x + b \\ y &= sigmoid(\alpha) \\ \delta &= z - y \\ W' &= W - \eta * \frac{d\delta}{dy}\frac{dy}{d\alpha}\frac{d\alpha}{dw} = W + \eta * \frac{dy}{d\alpha}x \\ b' &= b - \eta * \frac{d\delta}{dy}\frac{dy}{d\alpha}\frac{d\alpha}{db} = b + \eta * \frac{dy}{d\alpha} \end{aligned} \tag{3.16}$$

In forward propagation with quantized nodes, the quantized values (fixed-point) are used to calculate the values of α, y, and loss function δ. However, when performing backpropagation, the weights and biases are updated with the use of the original values (floating-point). As a result, when $\frac{dy}{d\alpha}$ is very small, or when x is very small, the updated weights and biases after one quantization may be the same as the value of the previous quantization. This will cause the loss function to be almost unchanged. In this case, the weight and bias of the calibration process may be blocked within a small range, making it difficult to fine-tune towards the target value.

For the $sigmoid(x)$ and $tanh(x)$ functions, the derivative is maximum near $x = 0$, and decreases away from the coordinate origin. In the process of QAT, since the baseline has converged and the LUT method constrains the original function within a small range, the precision loss after quantization causes only slight shifts near the original point. Therefore, parameters are less sensitive to changes during retraining. From a mathematical perspective, the quantization treats the activation function as a black box during retraining, requiring a mapping that is sensitive to accuracy loss but involves minimal hardware overhead and simple logic. Consequently, this paper uses linear functions, Hardsigmoid/ Hardtanh for internal mapping.

Given the similar characteristics of the derivatives of these two activation functions, this paper derives the Hardsigmoid and Hardtanh functions based on the Taylor expansion of the activation function at x=0. These functions must ensure the same output range as the original activation functions. The output of $sigmoid(x)$ is mapped to (0,1), while the output of $tanh(x)$ is mapped to (-1,1). The paper's Hardsigmoid and Hardtanh functions are defined as follows:

$$f'_{sigmoid}(x) = \frac{e^{-x}}{(1+e^{-x})^2} \tag{3.17}$$

$$f'_{tanh}(x) = \frac{4}{(e^{x}+e^{-x})^2} \tag{3.18}$$

$$Hardsigmoid(x) = \begin{cases} 1, & x > 2 \\ \frac{x}{4} + \frac{1}{2}, & -2 \leqslant x \leqslant 2 \\ 0, & x < -2 \end{cases} \tag{3.19}$$

$$Hardtanh(x) = \begin{cases} 1, & x > 1 \\ x, & -1 \leqslant x \leqslant 1 \\ 0, & x < -1 \end{cases} \tag{3.20}$$

Consider the example of $sigmoid(x)$ and its corresponding Hardsigmoid function, as shown in Figure 3.7. Assume that the quantization precision is 8 bits after the point. Then the loss of precision causes the input to vary in the range of 2^{-8}, Δx in the figure. The change of the output value using the LUT method is $\Delta y2$, while the change of the output value using the Hardsigmoid function is $\Delta y1$. The gap between the two changes is significant and tends to decrease away from the origin.

This method increases the accuracy of the model compared to the LUTs method while simplifying the hardware implementation of the activation function. The results will be discussed in the next chapter.

3.2. Hardware Implementation

This section provides a comprehensive overview of the hardware architecture, detailing the data flow and the design of sub-modules. Through this detailed exploration, this work aims to present a robust framework for hardware implementation that balances speed, resource utilization, and power consumption.

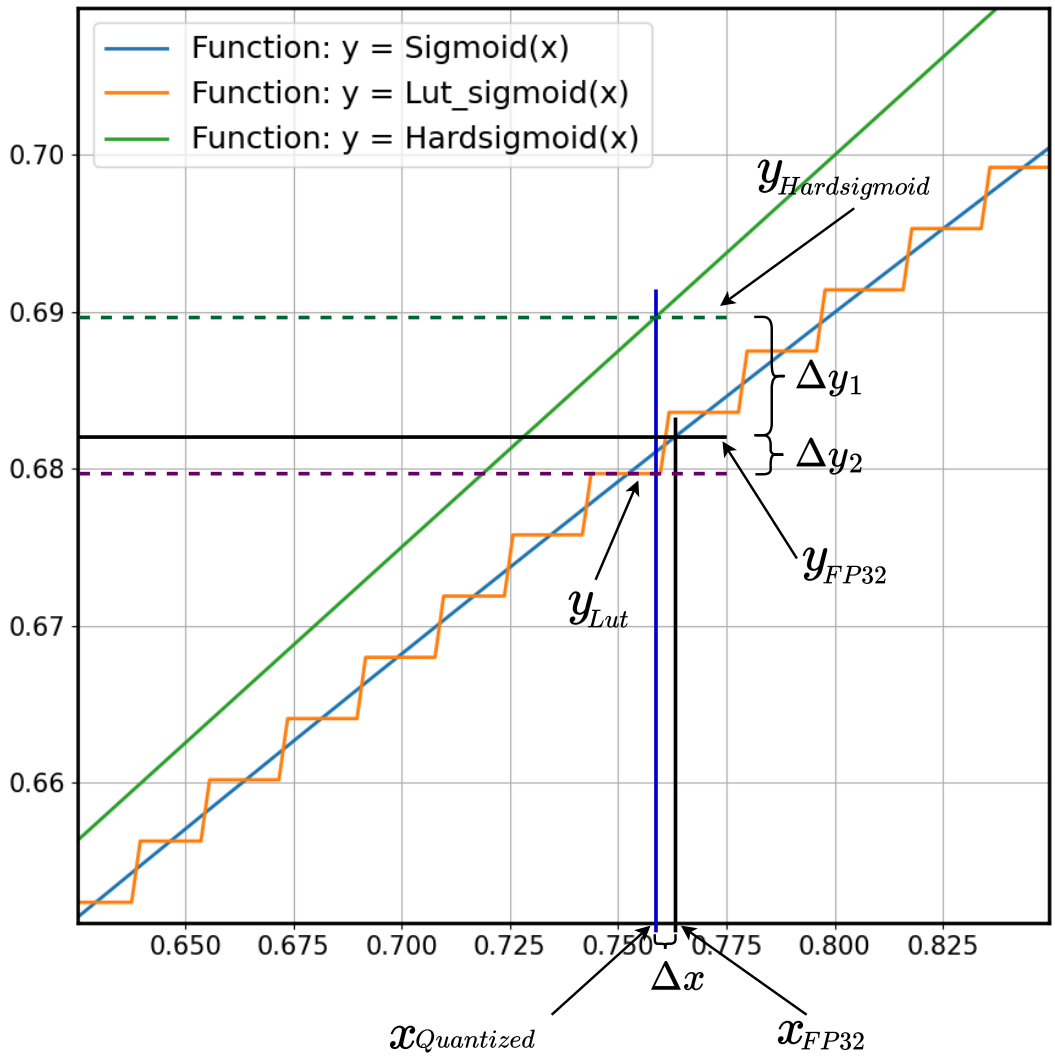


Figure 3.7: Comparison of sigmoid function approximations and quantization effects

3.2.1. Overview of Microarchitecture

The hardware design for the GRU-based DPD system consists of six main components: Feature Extractor, GRU Unit, FC Unit, X_{Weight} Buffer, H_{Weight} Buffer, and FC_{Weight} Buffer, as depicted in Figure 3.8. Thin arrows in the diagram represent single signal transmission, while thick arrows in the diagram indicate multiple signal transmission. X_{Weight} Buffer, H_{Weight} Buffer, and $Linear_{Weight}$ Buffer are used to store the parameters of the model. Given the small number of parameters in the GRU neural network, these parameters are stored using sliced registers.

The Feature Extractor module acts as the entry point of the hardware design, processing the input I_{IN} and Q_{IN} to extract and compute relevant features. These features are then sequentially outputted to the GRU Unit. The core computational modules in this design are the GRU Unit and the FC Unit.

The GRU Unit comprises six primary components. The input MAC and Ht MAC array handle matrix multiplication of x_t and h_t respectively. These two modules operate in parallel and share address lines. The Hidden State Buffer stores h_t values from the previous time step. The Sigmoid and Tanh modules serve as activation functions, while the Update Ht module is responsible for updating h_t at the current

time step.

When a feature is passed into the GRU Unit, the input MAC and Ht MAC are activated. Once the matrix multiplication is completed, the Sigmoid module is enabled. The output from the Sigmoid module is split into two parts: one part flows to the Update Ht module for updating the new h_t, and the other part flows to the Tanh module for updating n_t. After n_t is computed, it is passed into the Update Ht module to update the new h_t. Once the current time step's h_t update is finalized, the value is transmitted to the FC Unit for computing I_{OUT} and Q_{OUT}. Concurrently, the GRU Unit initiates a new computation cycle.

This design efficiently integrates parallel processing and sequential updating, optimizing both resource usage and computational speed.

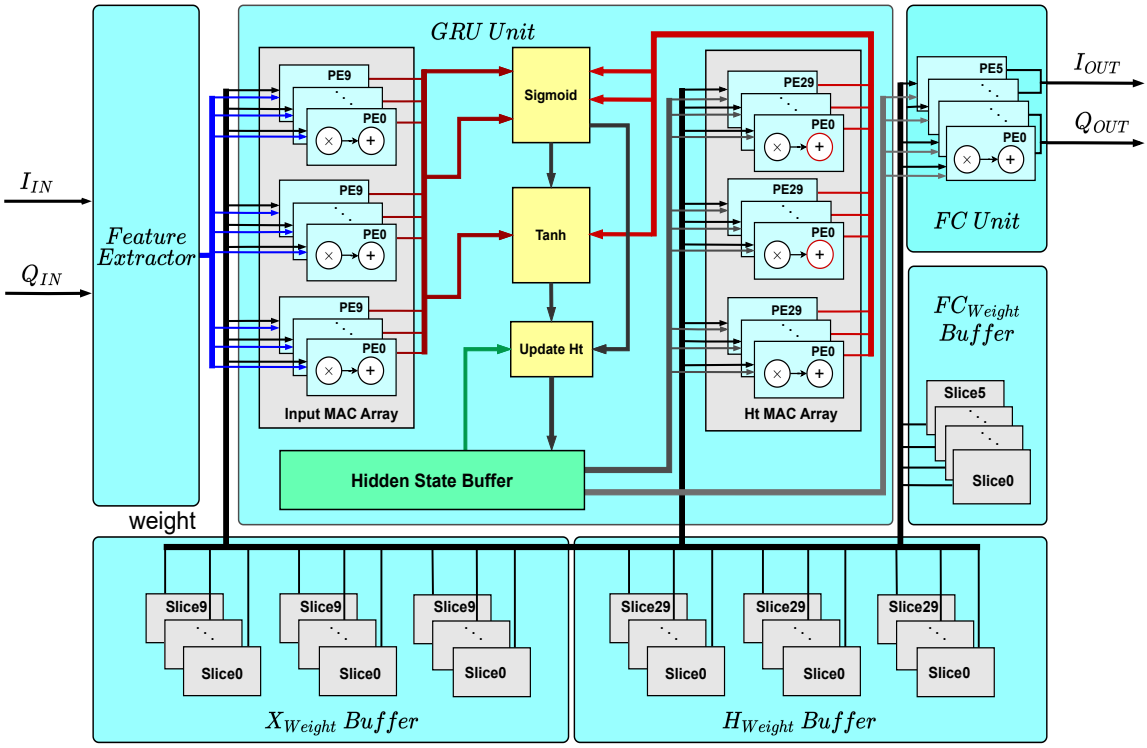


Figure 3.8: Microarchitecture of DPD hardware design

3.2.2. Structure

Limited by the CPU architecture, in software, the GRU model calculations follow the sequence rt, zt, nt, and finally ht. The first three involve matrix multiplications, and the subsequent ht update combines multiplication and addition operations. Consequently, the latter calculations must wait for the completion of the preceding modules, and its overall running time is the sum of these four parts, as shown in Figure 3.9. Unlike software, hardware circuits are characterized by high parallelism, making it unreasonable to deploy the hardware in the same sequential order.

For the same matrix, the calculation process involves traversing the elements of each row and performing multiplication and addition operations with the corresponding elements of each column of another matrix, repeating until the calculation of the last row is completed. The calculation of each row in this process is independent. For different matrices of the same size, their calculations do not interfere with each other, and when they start calculations synchronously, the addresses of the elements they access each time are the same.

Based on this, the design operates matrix multiplications in parallel with shared address lines, as shown in Figure 3.10. The number of cycles for matrix multiplication in this design depends on $max(feature\_size, hidden\_size)$. This design stores the weight matrix in a row-wise manner, and in each cycle, the elements are accessed and flow into the MAC module via shared address lines. The lines in the figure stand for address lines. In the first cycle, the multipliers calculate from $w_{00} * x_a$, $w_{10} * x_a$ to $w_{90} * x_a$, then next cycle repeats the process in column 2 until the row is traversed.

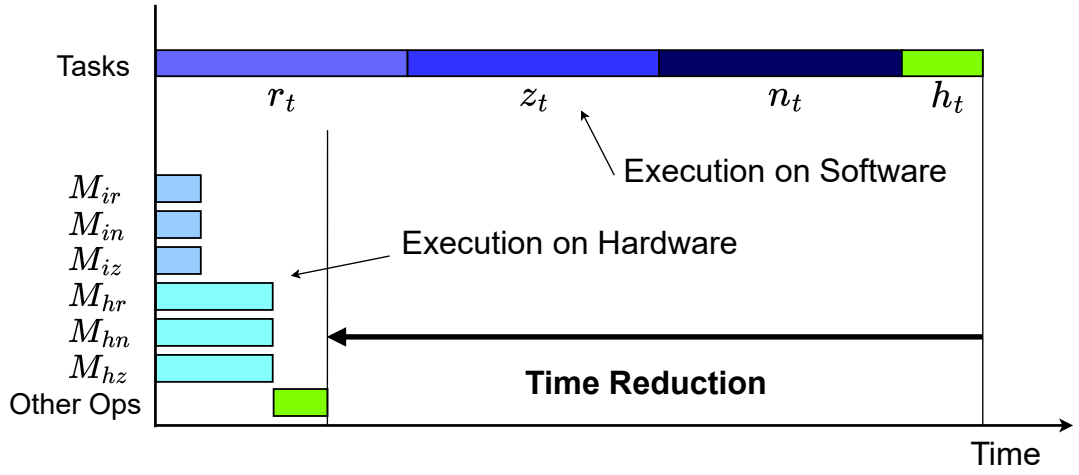


Figure 3.9: Comparison of timeline executed on software and hardware

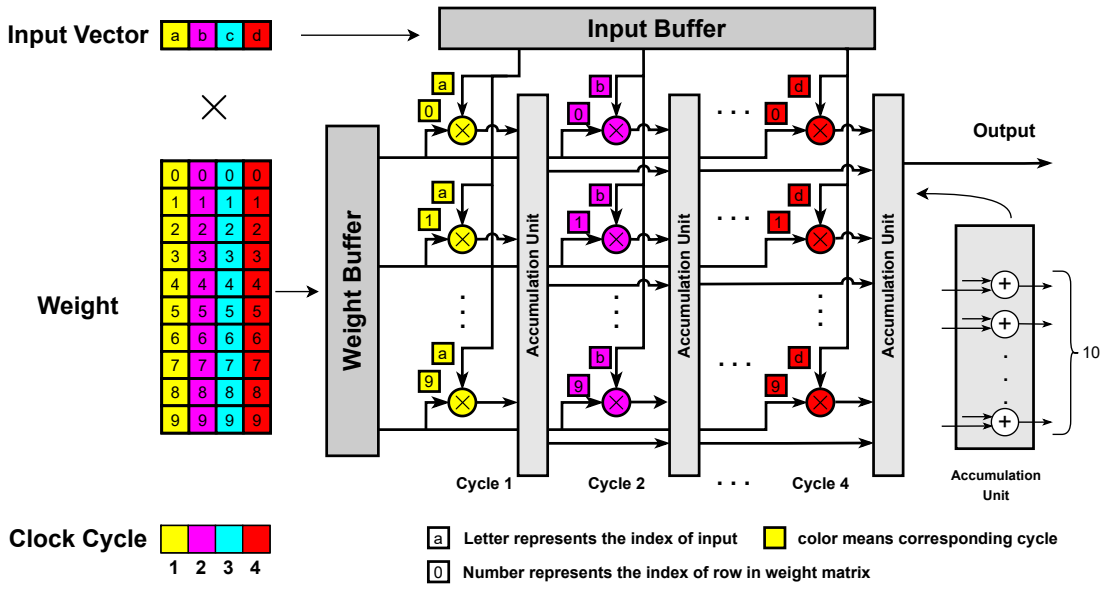


Figure 3.10: Structure of matrix multiplication on hardware

There are 6 matrix multiplications in the GRU algorithm in total, namely M_{ir}, M_{iz}, M_{in} and M_{hr}, M_{hz}, M_{hn} separately. The first three are of the same size and share one address line, while others are of the same size and share another address line. Therefore, without considering timing optimization, it takes four cycles to complete the calculation of the first three, while the next three require 10 periods with the configuration that the input size is 4 and the hidden state size is 10. Consequently, the bottleneck of the system is the number of computation cycles, as illustrated in Figure 3.9.

To enhance performance, this design focuses on further enhancing the parallelism of the computations for the final three matrices. The 10 elements in a row are divided into three groups, with quantities of 4, 4, and 2, respectively. Then the same address line is shared to calculate the six matrix multiplications. For the last three matrices, three numbers are fetched from the register in the first two cycles, completing the computation of slices with a capacity of 2 in the initial two cycles. In the subsequent two cycles, two numbers are fetched from the register to complete the remaining slices computation, as shown in Figure 3.11 and Figure 3.12. This optimization simplifies the control logic while introduces an additional set of addends during accumulation. The trade-off of sacrificing a small amount of circuit overhead to save two cycles is highly worthwhile. Ultimately, the number of computation cycles is

reduced from $max(feature\_size, hidden\_size)$ to $min(feature\_size, hidden\_size)$.

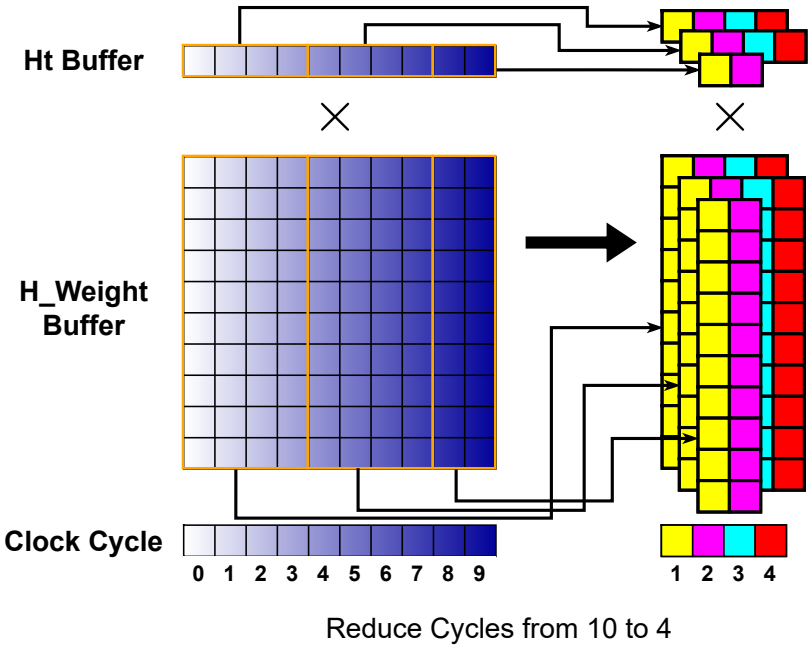


Figure 3.11: Storage of matrix on hardware after optimization

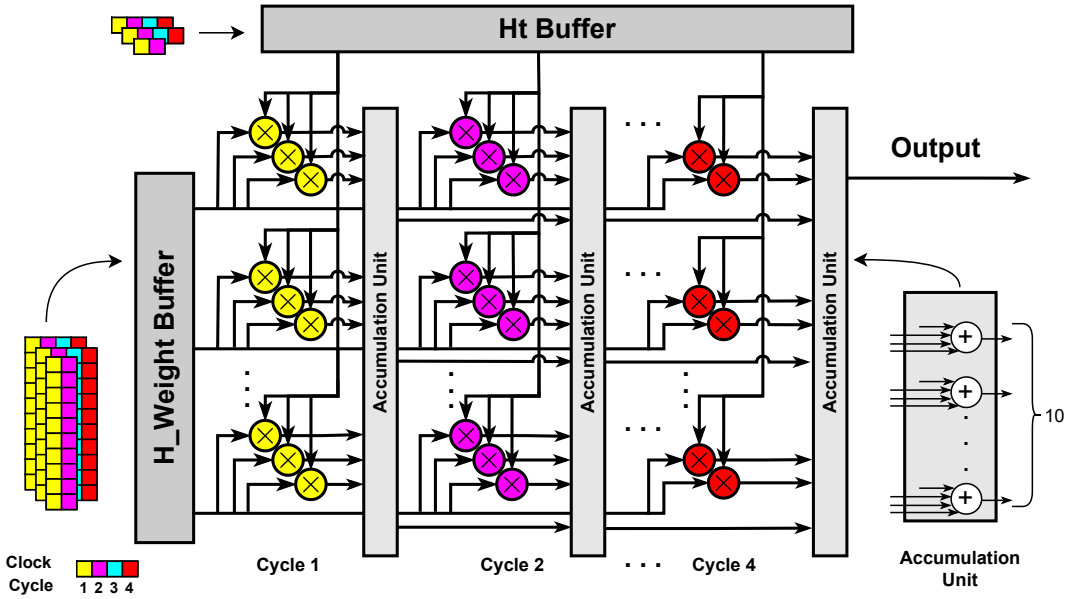


Figure 3.12: Structure of matrix multiplication on hardware after optimization

By improving the parallelism of matrix multiplication, the algorithm's runtime cycles are significantly reduced. Subsequently, the remaining computations of the GRU algorithm need to be efficiently allocated. From the PyTorch GRU formula, it is evident that the update of nt depends on the update of rt, and the update of ht depends on the update of nt. Furthermore, the matrix multiplication at the next time step depends on the current ht. Thus, a full pipelined architecture is not feasible. Our design

uses finite state machines to control the operation and switching of each module, utilizing an inter-layer pipeline and retiming method for timing operation. The states are as follows:

- IDLE: The idle state indicates that no activity in the system. When there are no I and Q inputs entering the system, the entire system remains in this state.
- MAC: The MAC state indicates that the system is performing matrix multiplication calculations.
- SIGMOID: The sigmoid state indicates that the system is computing rt and zt.
- TANH: The tanh state indicates that the system is computing nt.
- UPDATE_Ht: The update_ht state indicates that updating the ht of current time step.

The finite state machines provide a coarse partitioning of the algorithm and hardware structure. However, due to the numerous operations such as truncation, multiplication, and addition, the distribution of these operations in the state machine is crucial for timing optimization. This design introduces an inter-layer pipeline to address local timing issues within regions. Unlike full pipeline, inter-layer pipeline increases the number of data processing cycles for the entire system. The MAC module consists of a 12-bit multiplier, a 28-bit adder, and truncation, forming a long carry chain. So a register is inserted between the multiplier and adder, as illustrated in Figure 3.13.

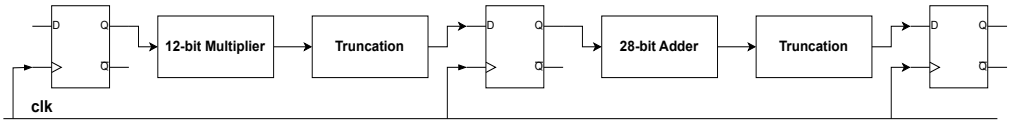


Figure 3.13: Structure of inter-layer pipeline

In a digital circuit system, the clock period is determined by the critical path, which is the longest combination logic circuit. Therefore, to increase the clock frequency, it is necessary to evenly distribute the circuit structure between registers while maintaining the algorithm structure. This design applies the retiming method. Truncation operations occur almost everywhere after each computation, and the placement of truncation operations in different states significant impact timing.

To illustrate, consider a 20-bit multiplication followed by a 20-bit addition. After multiplication, the 40-bit result is truncated to 20 bits, and after addition, the 21-bit result is truncated to 16 bits, completed in two cycles. In one design, both the multiplication and its corresponding truncation are computed in one cycle, with addition and its truncation in the second cycle. Due to the higher complexity of multiplication, the critical path is determined by multiplication and truncation. However, if the truncation after multiplication is placed in the second cycle, the critical path only depends on the multiplier. The retiming result of this design is illustrated in the Figure 3.14.

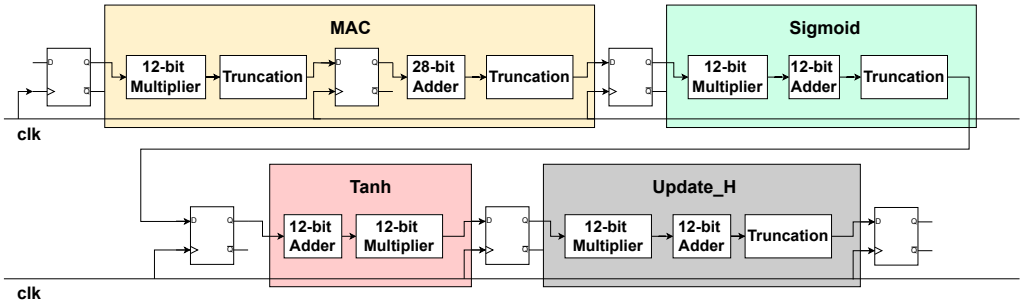


Figure 3.14: Structure of hardware design with inter-layer pipeline and retiming

For the fully connected matrix multiplication, its computational logic is identical to that of the GRU matrix multiplication. Therefore, it shares address lines and control logic with the GRU matrix multiplication and begins computation after the first ht update.

In this system, a multiply-accumulate (MAC) unit is defined as a processing element (PE). The hardware system needs 158 PEs in total. The number of PES in each module is shown in the Table 3.1.

Module	Feature Extractor	MAC_GRU	nt	ht	MAC_FC
Number	2	120	10	20	6

Table 3.1: Numbers of PEs in each module

3.2.3. Data Stream

In contrast to software calculations, implementing binary calculations on hardware requires consideration of bit width and overflow judgment in addition to designing the calculation logic.

In hardware circuits, signed numbers are stored using 2's complement, which allows binary subtraction operations to be simplified into addition operations, streamlining the hardware design. Therefore, in this design, the input signals I and Q are fed into the system in 2's complement form, and weight and bias are also stored in the register in this form.

For an n-bit signed binary number, the range is (-2^{n-1}, $2^{n-1}-1$). When adding two signed binary numbers, the result's bit width is one bit wider than the addends. For the addition of m n-bit signed binary numbers, the result's bit width can be calculated by the following formula, ensuring no overflow:

$$Bitwidth = \lceil \log_2 m(2^n - 1) \rceil \tag{3.21}$$

Similar to binary addition, binary multiplication expands the bit width of the result. In the GRU algorithm, multiple multipliers are cascaded. Without truncation, the bit width can expand excessively, wasting hardware resources. For example, when two binary numbers with an effective precision of 8 bits after the point are multiplied, the effective precision of the product result is 16 bits after the decimal point. The value after the effective digit has a minimal impact on the accuracy of the quantization model.

When optimizing adders and multipliers, the primary considerations are resource usage and critical paths. Optimization can be approached in two ways: structural optimization (e.g., Carry-lookahead adders, Bypass adders, Wallace tree multipliers) and logic chain optimization. The former involves selecting efficient structures, often handled by synthesis tools. The latter involves splitting the logic chain and increasing clock frequency at the cost of longer processing cycles. For example, a 12-bit addition might operate at 500 MHz in a single cycle, but using a split method, the same operation could achieve 700 MHz over two cycles.

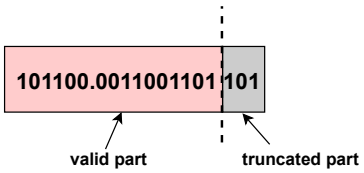


Figure 3.15: Clamp format

This design uses the $floor$ approximation method to clamp the intermediate results. Compared to the $round$ method, the $floor$ method is hardware-friendly. The binary number needs to be divided into two parts based on the precision, as shown in Figure 3.15. For the $round$ method, the process consists of two steps:

- Step 1: Determine whether to carry based on the highest bit of the truncated part
- Step 2: Processing the valid part: If a carry is needed, add 1 to the valid part and check for overflow. Otherwise, use the valid part's value as the output.

In contrast, the $floor$ method retains the valid part without other operations (comparison, addition, and overflow judgment). This can save resources and optimize critical path in the hardware circuit.

According to the software results, this design set the bit width of i and q signals, as well as weights and biases, to 12 bits in the format $Q2.10$, with a hidden_size of 10. In the MAC module, the bit width is set to 16 bits to store the matrix multiplication result in the format $Q6.10$. The $Q6.10$ format represents decimal numbers in the range of $(0, 63)$. Obviously, there will be no overflow over this range in the normalized DPD system.

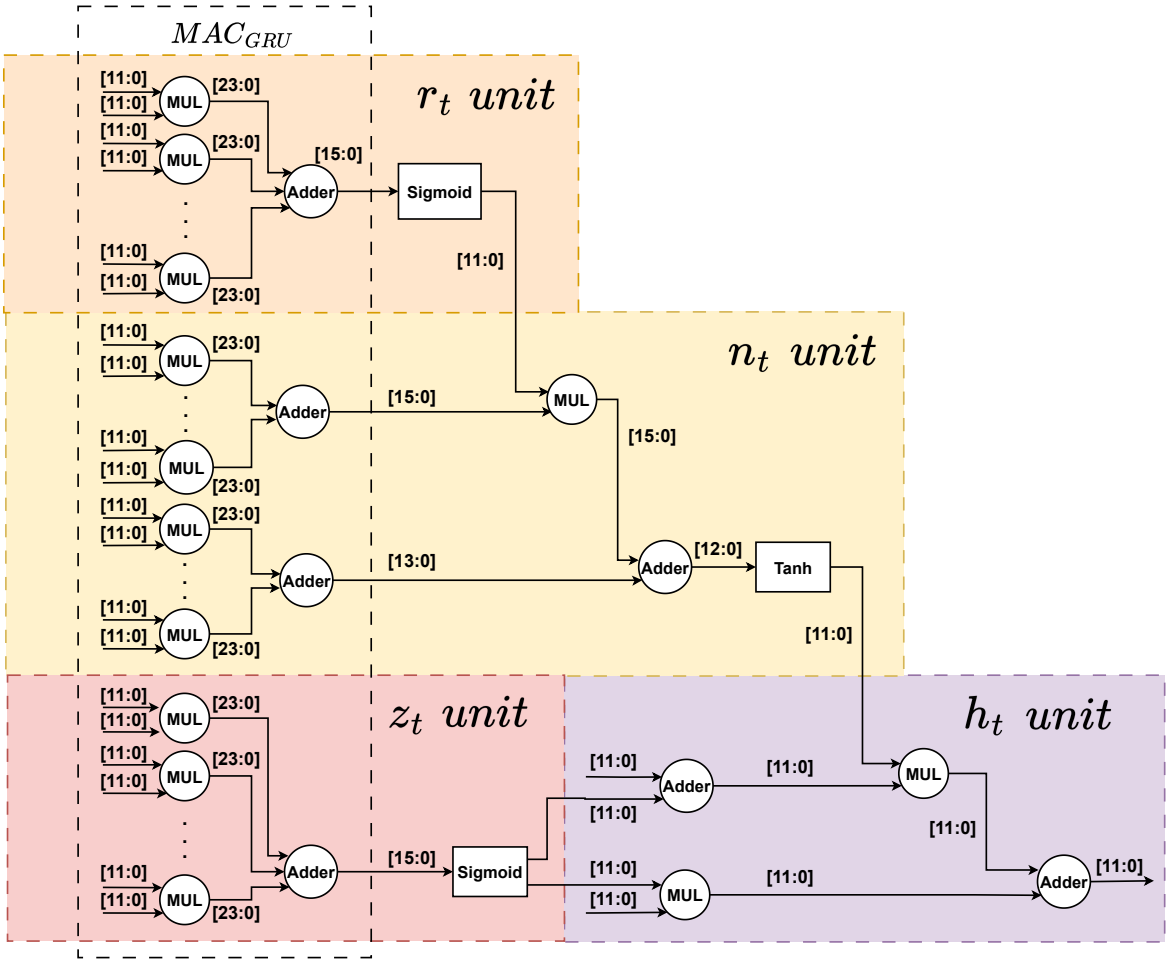


Figure 3.16: Overall dataflow of hardware design

To design a high-performance DPD hardware implementation capable of processing high-frequency signals, addition and multiplication in this design are completed in one cycle respectively. The entire data flow of the hardware system is shown in the Figure 3.16.

3.2.4. Feature Extractor

The feature extractor module, illustrated in Figure 3.17, is designed to convert the synchronized I and Q signals input into four feature signals: the original I and Q, as well as the computed $I^2 + Q^2$ and $(I^2 + Q^2)^2$. This module outputs one feature per clock cycle over a span of four cycles.

Figure 3.17: Feature extractor block diagram

Based on the functions and computational characteristics of this module, we split the computation of features to enhance the usage of data and circuits and employ a finite-state machine to control the output. The arithmetic unit for $I^2 + Q^2$ comprises two high-order multipliers and one high-order adder. Considering that the output of this feature is scheduled for the third cycle, the module distributes the calculation over the first two cycles. In the first cycle, I^2 and Q^2 are computed separately. In the second

cycle, the sum of these two products is calculated. The output of $(I^2 + Q^2)^2$ is scheduled for the fourth cycle, and its computation is based on the already computed $I^2 + Q^2$. Therefore, the module performs the exponentiation of $I^2 + Q^2$ in the third cycle.

This design effectively balances the system's operating frequency and resource utilization. The operation and output in different cycles are summarized in Table 3.2.

Cycle	**1**	**2**	**3**	**4**
Operation	I^2 , Q^2	$I^2 + Q^2$	$(I^2 + Q^2)^2$	
Output	I	Q	$I^2 + Q^2$	$(I^2 + Q^2)^2$

Table 3.2: Operation and output of each cycle of feature extractor module

3.2.5. Activation Function

The sigmoid and tanh modules are responsible for activating signals, and their hardware implementation are similar.

Truncation Optimization

The activation function module's first challenge is truncation. Similar to how the bit width of multipliers affects timing and area, wide comparators increase hardware area and reduce system clock frequency. This is also one reason why look-up tables are not used to implement activation functions. Therefore, this module optimizes the truncation first.

For the sigmoid function, the input is a 16-bit $Q6.10$ format, which needs to be constrained to the range of -2 to 2, resulting in a $Q2.10$ format. For the tanh function, the input is a 13-bit $Q3.10$ format, which needs to be constrained within the range of -1 to 1, also resulting in a $Q2.10$ format.

To determine whether the input value is not less than 2 or not greater than -2, only the integer part needs to be compared. The input value can be split into two parts: $Value_{integer}$ and $Value_{fraction}$.

$$Value = Value_{integer} + Value_{fraction} \tag{3.22}$$

Here, $Value_{integer}$ can be a positive or negative number with a precision of 1, such as 0, 1, 2, -1, -2, etc, while $Value_{fraction}$ is a number between 0 and 1, representing the fractional part.

- When $Value_{integer}$ is greater than 1, it indicates that the original input is a number greater than or equal to 2, then this input is truncated to 12'b01.1111_1111_11.
- When $Value_{integer}$ is smaller than -2, it indicates that the original input is a number less than -2, and this input is truncated to 12'b10.0000_0000_00.

This optimization reduces a 16-bit comparator to a 6-bit comparator.

Hardsigmoid and Hardtanh Implementation

The second part of implementation in the modules is the Hardsigmoid/Hardtanh function. The difficulty of hardware implementation was considered when designing the linear function in software. In the hardware circuit, a simple shift operation can represent operation $\times 2^n$. For the sigmoid function, defined as $sigmoid = \frac{x}{4} + \frac{1}{2}$, the simplified calculation process is as follows :

- Right shift and truncation: Shift the input value to the right by two bits, discarding the lower two bits.
- Addition: There are two possible cases for the first three bits of the result after the right shift.
  * Case1: The original input is a positive number, and the first three bits after the shift are 3'b00.0. Adding 3'b00.1 (0.5) to this result yields 3'b00.1.
  * Case2: The original input is a negative number, and the first three bits after the shift are 3'b1.11. Adding 00.1 (0.5) to this result yields 3'b00.0 (overflow is ignored because the output range is (0,1)).

Therefore, the optimized logic can be expressed by the following formula:

$$data_{out} = \left\{00, \overline{In[width-1]}, In[width-2:2]\right\} \tag{3.23}$$

In is the 12-bit signal. $[width-2:2]$ means truncating input signal from the 3rd lower bit to the 11th lower bit. $\overline{In[width-1]}$ means doing not operation of the highest bit of In signal.

4 Results

4.1. Experimental Setup

4.1.1. Software Design Setup

The training for the experiment was conducted on an NVIDIA RTX 2050 Laptop GPU, utilizing the ADAMW optimizer for 100 epochs for the floating-point model and 120 epochs for QAT quantization strategy. The $ReduceLROnPlateau$ scheduler was employed, starting with an initial learning rate of 1 × 10^{-3}. The batch size was set to 64, with a frame length of 50 and a stride of 1.

The 200 MHz dataset from OpenDPD was used in this experiment. It was partitioned into three subsets: 60% for training, 20% for validation, and 20% for testing.

The basic framework of this design consists of a GRU neural network and a fully connected neural network. The GRU neural network is configured with an input feature size of 4, a hidden size of 10, and a single hidden layer. The total number of parameters in the network is 502.

4.1.2. Hardware Design Setup

Considering the complexity of the ASIC design process, including the time-consuming and intricate stages of simulation, synthesis, and place and route, using FPGA offers advantages due to its re-programmability. To enhance efficiency, this experiment employs FPGA for functional simulation and timing synthesis during the hardware structure optimization phase. Once the hardware structure design is finalized, functional simulation, gate-level simulation, and post-layout simulation, as well as place and route, are performed on the ASIC.

The platform this work utilized is Vivado 2022.2. For the FPGA simulation on Vivado, the chip used is xc7z020clg400-1, which contains abundant resources to meet the design requirements. The available resources of this chip are shown in Table 4.1.

	Slice LUTs	**Slice Registers**	**Slice**	**LUT as Logic**	**DSP**
Available	53200	106400	13300	53200	220

Table 4.1: Available on-board resources of FPGA

For ASIC, this work uses Cadence Genus to synthesize the RTL design, Xcelium to simulate the design, and Innovus to place and route the design.

Both ASIC and FPGA front-end designs are developed using Verilog HDL, though they differ in synthesis and routing methodologies. In ASIC design, synthesis converts Verilog code into gate-level circuits, followed by routing these logic gates. In contrast, FPGA uses LUTs as the basic logic units. During FPGA synthesis, Verilog code is transformed into LUTs, which can cause some discrepancies in critical path analysis between the two platforms.

In hardware circuit design, PPA (Power, Performance, and Area) are crucial metrics for evaluating the quality of the design, defined as follows:

- Performance: Performance in hardware circuit design evaluates the functionality of the circuit, including metrics such as clock frequency, data processing cycles, throughput, and accuracy.
- Power: Power measures the amount of electrical energy consumed by the circuit during operation. Due to the structure of the FPGA, power estimation is not precise and can only serve as a reference.
- Area: The area evaluates the size of the hardware designs. One of the objectives of this study is to design a compact hardware circuit structure. On FPGA, the number of LUT represents area size, while on ASIC, the area is the number of physical areas.

4.2. Software

The software results dictate the selection of hardware parameters, including feature size, hidden size, and precision bits. In the software, this work first compared the accuracy variations under different feature selections and hidden size settings of the float model. Based on these results, the hardware configuration, feature size, and hidden size were determined. Subsequently, the software quantized the inputs and parameters of the float model to different precisions and quantized the activation function using the LUT method. These results informed the determination of the bit width for the LUT method (baseline) in the hardware design. Finally, the software further quantized the float model to different precisions and used Hardsimoid/Hardtanh to quantize the activation function. The bit width of output of the Hardsimoid/Hardtanh was determined based on the accuracy of the results.

4.2.1. Floating-point Model

In this section, the paper compares the output accuracy for different feature combinations and hidden sizes: I and Q, I and Q with square of amplitude, and I and Q with square and fourth power of amplitude.

From Table 4.2, It can be observed that when the selected feature extraction includes only I and Q, the accuracy improvement is minimal, even with an increased hidden size and model parameters. In this scenario, the bottleneck lies in the limited features. When three features are selected, the ACPR of the model approaches close to -50 only when the parameters are very large, which results in significant hardware overhead and is not cost-effective. Conversely, when four features are selected, the ACPR increases gradually with the growth of the hidden size.

To balance the number of parameters with accuracy, this work makes a compromise to employ a GRU network model with a feature extraction of **four**, namely I, Q, $|x|^2$, $|x|^4$, and a hidden size of **10** for DPD applications.

Feature	Hidden Size	VAL-NMSE (dB)	VAL-EVM (dB)	VAL-ACPR (dBc)	TEST-NMSE (dB)	TEST-EVM (dB)	TEST-ACPR (dBc)	Parameter
I,Q	10	-33.15	-35.48	-44.08	-32.87	-35.65	-43.87	442
I,Q	12	-39.76	-42.38	-45.37	-39.76	-42.45	-44.47	602
I,Q	14	-39.92	-42.85	-46.23	-39.96	-42.98	-45.20	786
I,Q,$\|x\|^2$	10	-34.26	-36.43	-45.42	-33.95	-36.61	-44.33	472
I,Q,$\|x\|^2$	12	-43.76	-46.95	-49.70	-43.66	-46.91	-49.56	638
I,Q,$\|x\|^2$,$\|x\|^4$	8	-43.60	-46.77	-49.51	-43.57	-46.32	-49.62	354
I,Q,$\|x\|^2$,$\|x\|^4$	**10**	**-43.95**	**-46.93**	**-50.20**	**-43.91**	**-46.70**	**-49.58**	**502**
I,Q,$\|x\|^2$,$\|x\|^4$	12	-44.44	-47.85	-51.37	-44.71	-48.01	-51.38	674

Table 4.2: Performance of GRU-based DPD with various feature sets and hidden sizes

4.2.2. Quantization Model

Quantization with Lookup Tables Method

Once the configuration of GRU neural network is determined, this work quantizes the inputs and parameters and uses the LUT method to quantize the activation functions, sigmoid and tanh, based on the floating-point models from the previous section.

As shown in Table 4.3, the precision of input data, parameters, and LUTs significantly impacts the performance metrics. When the precision of input data and parameters exceeds that of the LUTs, the

ACPR visibly increases with higher LUTs' precision, indicating that the bottleneck is the LUTs' precision. Conversely, when the precision of input data and parameters is lower than the LUTs' precision, the ACPR remains almost unchanged, suggesting that the bottleneck is the precision of the input data and parameters.

In the LUTs method, the bit-width selection strategy aims to choose larger bit-widths for inputs and weights while keeping the bit-width of the LUTs smaller, ensuring the ACPR degradation remains within 1 dBc relative to the floating-point model (50.2dBc/49.58dBc). Therefore, in the hardware LUT method (baseline), the data formats are **Q2.14** for I, Q signals, **Q2.7** for Sigmoid, and **Q2.8** for Tanh.

Weight	LUT	VAL-NMSE (dB)	VAL-EVM (dB)	VAL-ACPR (dBc)	TEST-NMSE (dB)	TEST-EVM (dB)	TEST-ACPR (dBc)
Q2.9	Q2.10	-40.34	-44.02	-46.91	-40.15	-43.64	-46.14
Q2.9	Q2.11	-39.33	-43.68	-47.26	-39.29	-43.39	-46.72
Q2.9	Q2.12	-38.47	-43.47	-47.24	-38.59	-43.33	-46.51
Q2.10	Q2.7	-32.11	-37.32	-39.08	-32.33	-37.59	-39.15
Q2.10	Q2.8	-35.91	-40.72	-43.43	-36.19	-41.08	-42.87
Q2.10	Q2.9	-39.50	-44.19	-46.40	-39.53	-43.92	-45.98
Q2.10	Q2.10	-41.50	-45.21	-47.86	-41.53	-45.05	-47.61
Q2.10	Q2.11	-42.65	-46.02	-48.74	-42.59	-45.84	-48.36
Q2.10	Q2.12	-42.22	-45.68	-48.74	-42.34	-45.78	-48.44
Q2.14	Q2.7	-41.50	-45.08	-48.12	-41.46	-45.12	-47.76
Q2.14	**Sigmoid:Q2.7 Tanh:Q2.8**	**-42.92**	**-46.47**	**-49.42**	**-42.91**	**-46.65**	**-48.84**
Q2.14	Q2.8	-43.05	-46.54	-49.69	-43.10	-46.69	-48.77

Table 4.3: Performance of GRU-based DPD with different precision using LUT method

Quantization with Hardsimoid/Hardtanh Function

Due to the imbalanced resource distribution of hardware design implementing the activation function using the LUT method, this paper proposes a new method in which the activation function is quantized by Hardsimoid/Hardtanh.

As shown in Table 4.4, When the quantization precision is more than 10 bits, the reduction of ACPR is negligible compared with that of the floating-point model, and in some cases, the performance can even surpass that of the floating-point model. This work selects a precision of **10** bits after point for both weight and the Hardsimoid/Hardtanh function as the configuration for the hardware data stream.

Weight	Activation Function	VAL-NMSE (dB)	VAL-EVM (dB)	VAL-ACPR (dBc)	TEST-NMSE (dB)	TEST-EVM (dB)	TEST-ACPR (dBc)
Q2.9	Q2.10	-41.16	-45.15	-48.45	-41.07	-44.83	-47.82
Q2.9	Q2.11	-41.33	-45.69	-49.09	-41.47	-45.38	-48.68
Q2.9	Q2.12	-38.29	-44.99	-48.77	-38.44	-44.73	-48.30
Q2.10	Q2.7	-33.06	-38.61	-40.09	-33.21	-38.69	-39.96
Q2.10	Q2.8	-37.33	-42.47	-44.59	-37.38	-42.33	-44.44
Q2.10	Q2.9	-40.14	-45.04	-47.30	-40.12	-44.73	-46.88
Q2.10	**Q2.10**	**-42.44**	**-46.44**	**-49.42**	**-42.23**	**-45.93**	**-48.74**
Q2.10	Q2.11	-42.14	-46.55	-50.10	-42.00	-46.16	-49.34
Q2.10	Q2.12	-43.59	-47.05	-50.16	-43.42	-46.60	-49.60
Q2.11	Q2.10	-42.36	-46.82	-49.83	-42.29	-46.35	-48.73
Q2.11	Q2.11	-43.97	-47.66	-50.49	-43.79	-47.25	-50.30

Table 4.4: Performance of GRU-based DPD with different precision using Hardsimoid/Hardtanh Function

Summary
Comparing the experimental results of the model using LUT method in Table 4.3 and the Hardsimoid/Hardtanh function in Table 4.4, as well as the bar chart shown in Figure 4.1, It can be concluded that under the same precision, the accuracy of the Hardsimoid/Hardtanh function is much higher than that of the LUT method, among which the difference of ACPR can be **1-2 dBc**. When ACPR is almost the same, the Hardsimoid/Hardtanh function can quantize the floating point number to a lower bit width. For example, the accuracy of the model using the Hardsimoid/Hardtanh function with a weight format of **Q2.9** and a Hardsimoid/Hardtanh format of **Q2.11** is almost the same as that of the model using the LUT method with a weight format of **Q2.10** and a lut format of **Q2.12**. From the perspective of hardware design, the use of the Hardsimoid/Hardtanh function can save hardware resources and reduce chip area while maintaining accuracy.

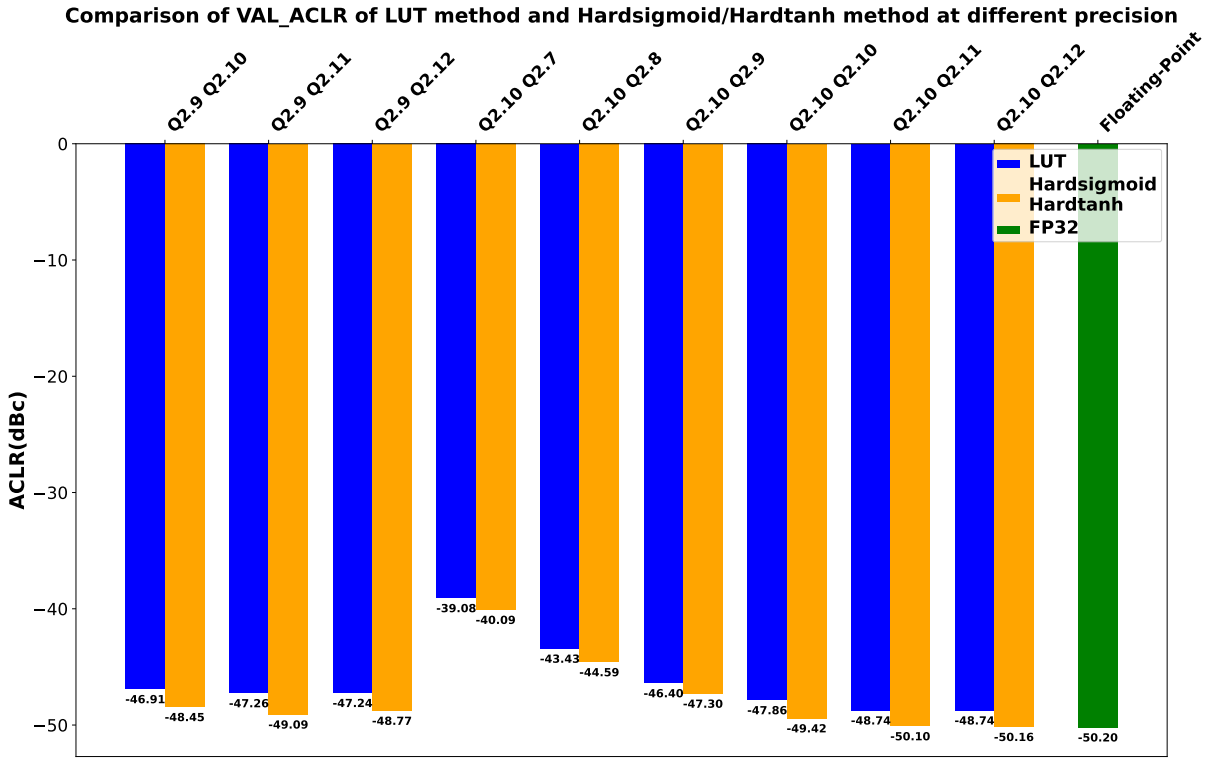


Figure 4.1: Comparison of results of LUT method and Hardsimoid/Hardtanh function at different precision

4.3. Hardware

Hardware implementation is divided into two stages. The first stage, the optimization phase, involves optimizing the hardware area as the primary step, followed by timing, and finally, operational cycles. The hardware design of this stage is implemented on the FPGA platform, where an analysis and comparison of area and power consumption are performed. The second stage is to perform the optimized hardware design on ASIC. In this stage, the experiment evaluates various aspects, including power consumption, area, timing, and throughput.

4.3.1. FPGA

In this section, the paper first implements a model using LUTs method for the activation function. To address the issue of excessive area, the Hardsimoid/Hardtanh function is employed for the activation function to optimize the hardware circuit. After resolving the area issue, the hardware circuit is further optimized by increasing the parallelism of matrix multiplication to address timing issues and reduce the operational cycles. The bit weight in hardware design is determined by software experimental results. The synthesis adopts strategy 'Flow_PerfThresholdCarry' and the implementation adopts strategy 'Vi-

vado Implementation Defaults'.

LUT Method

Based on the result of the software, the hardware is configured with parameters with a precision of 14 bits after point and the output of sigmoid and tanh with a precision of 8 bits and 7 bits after point, separately. The device diagram of hardware design using LUT Method is shown in Figure 4.2. Because the number of LUTs for Sigmoid and Tanh is extremely huge and the computation of the design is in high parallelism that **20** Sigmoid modules and **10** Tanh modules are instanced simultaneously, even this neural network with small parameters take up about half of resources in hardware implementation.

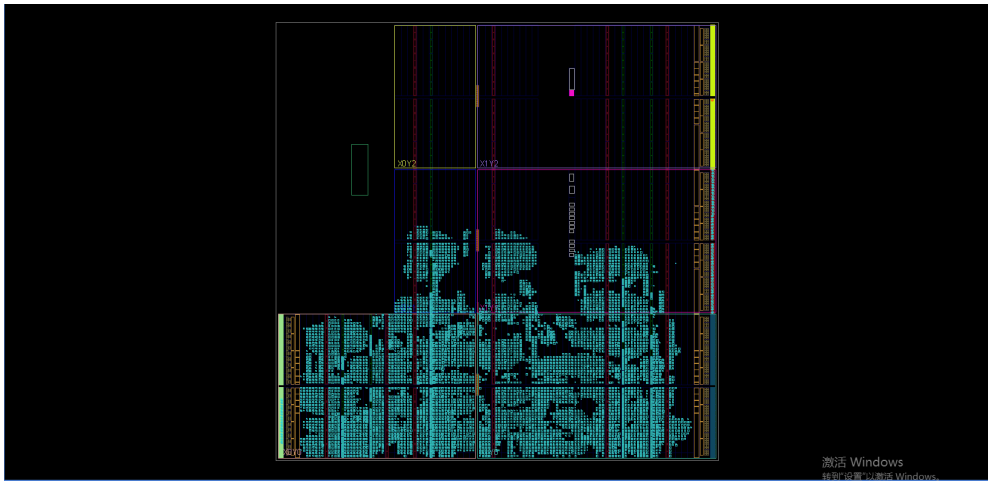


Figure 4.2: Device diagram of hardware design using LUT method

To be specific, Table 4.5 illustrates the resource consumption and distribution for each module. The majority of the slice LUT consumption is attributed to the implementation of the LUTs of activation functions, while the resources allocated for MAC and other neural network operations account for only about one-quarter of the used resources. Notably, almost all the mux resources are utilized for the sigmoid LUT and tanh LUT. From an area perspective, this resource allocation is highly inefficient for the hardware accelerator design of this small GRU network. The activation function, which constitutes a very small part of the algorithm, incurs three times the hardware overhead of MAC and all other operations, including multiplication, addition, and truncation. The main bottleneck in the hardware design lies in the implementation of the activation function. Therefore, this paper proposes using the Hardsimoid/Hardtanh function to quantize the activation function. In a hardware-software co-design, the optimization process should verify the accuracy at first in software, followed by hardware implementation.

	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Slice	LUT as Logic	DSP
Available	53200	106400	26600	13300	13300	53200	220
Sigmoid LUT	6598(32.2%)	140(3.5%)	502(28.2%)	82(57.7%)	1936(34.4%)	6598(32.2%)	0
Tanh LUT	8461(41.2%)	250(6.3%)	1260(70.9%)	60(42.3%)	2328(41.4%)	8461(41.2%)	0
MAC and Other Ops	5463(26.6%)	3579(90.2%)	16(0.9%)	0	1358(24.2%)	5463(26.6%)	85(100%)
Used Totally	20522	3969	1778	142	5622	20522	85

Table 4.5: Hardware utilization of design using LUT method

Table 4.6 and Table 4.7 illustrates the power consumption distribution. Similar to the resource allocation, the power consumed by the activation function exceeds that of MAC and all other operations combined. So the majority of energy is consumed by sigmoid LUT and tanh LUT.

In timing analysis, due to the constraints of the FPGA chip, the clock frequency cannot reach a very high frequency. Therefore, the timing analysis here focuses on the critical path, while the clock frequency serves as a reference. In this design, the clock period is set to 11ns. However, there is still

	Static	Dynamic	Total
Power (W)	0.112	0.472	0.584
Percentage	19.2%	80.8%	100%

Table 4.6: Power consumption of design using LUT method

	Sigmoid	Tanh	MAC and Other Ops	Dynamic
Power (W)	0.213	0.057	0.202	0.472
Percentage	45.1%	12.1%	42.8%	100%

Table 4.7: Distribution of dynamic power consumption of design using LUT method

a timing violation -0.322ns. And as it is shown in Figure 4.3, the critical path is primarily located in the Tanh module.

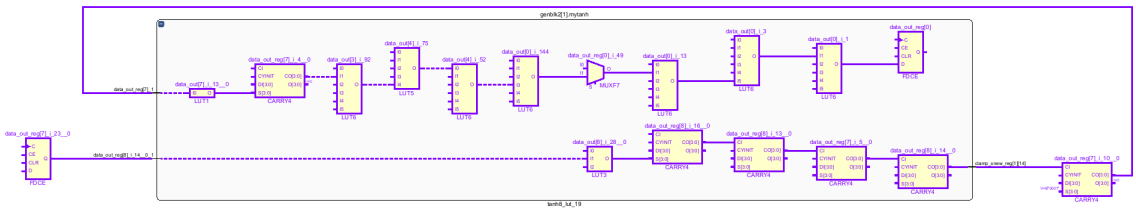


Figure 4.3: Critical path of hardware design using LUT method

Hardsimoid/Hardtanh Function

Based on the result of the software, the hardware is configured with a parameter with a precision of 10 bits after the point and the intermediate variable with a precision of 10 bits after the point. The device diagram of hardware design with the Hardsimoid/Hardtanh function is shown in Figure 4.4. The device diagram appears less congested compared to before, indicating a significant reduction in hardware resource usage.

Table 4.8 illustrates the resource utilization of each module. Comparing Table 4.5 and Table 4.8, it is obvious that the allocation of resources to the activation functions decreases dramatically. The number of Slice LUTs of Sigmoid decreases from **6598** to **350** while that of Tanh decreases from **8461** to **240**. The significant reductions have resulted in almost **18.85x** and **35.25x** saving of hardware resources. Meanwhile, it also mitigates the issue of excessive mux usage due to longer address lines and wider data bit widths. Moreover, as a neural network accelerator, its resource allocation is highly balanced, where the majority of resources are allocated for parameter storage and computation. And the reduction in area will lead to a decrease in power consumption and an improvement in timing.

Due to the very low power consumption in the Sigmoid and Tanh modules, Vivado statistics show the figure is less than 1% of the total power. Table 4.9 lists their dynamic power, static power, and overall power. By comparing Table 4.6 and Table 4.9, it can be concluded that optimizing the activation function with the Hardsimoid/Hardtanh function has significantly reduced the corresponding power consumption to negligible levels.

In timing analysis, the replacement of the long-bit comparators with shift operations will shorten the critical path, leading to a noticeable increase in clock frequency. In this design, the clock period is set to 7ns, and there is no timing violation, with WNS, 0.017ns.

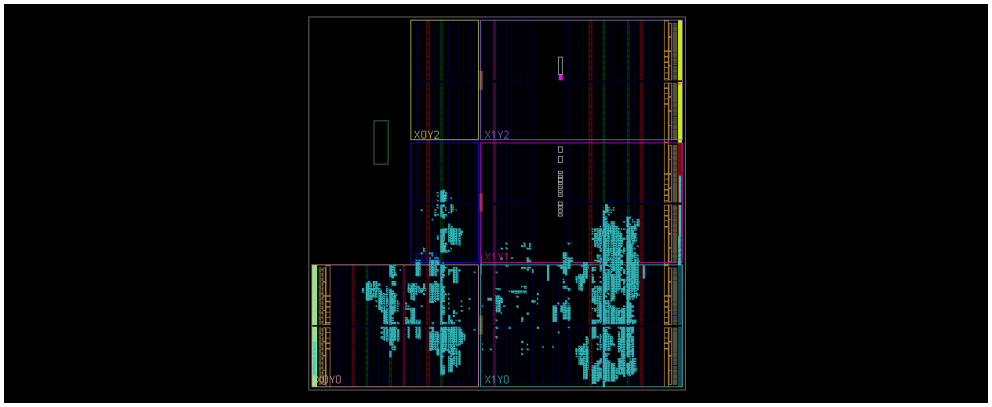


Figure 4.4: Device diagram of hardware design using Hardsimoid/Hardtanh function

	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Slice	LUT as Logic	DSP
Available	53200	106400	26600	13300	13300	53200	220
Sigmoid LUT	350(6.4%)	200(6.3%)	0	0	105(6.4%)	350(6.5%)	0
Tanh LUT	240(4.4%)	120(3.8%)	0	0	82(5.0%)	240(4.5%)	0
Other Ops	4849(89.2%)	2836(89.9%)	12(100%)	0	1452(88.6%)	4849(89.0%)	95(100%)
Used Totally	5439	3156	12	0	1639	5349	95

Table 4.8: Hardware utilization of design using Hardsimoid/Hardtanh function

Increase Parallelism and Retiming

As mentioned in the methodology chapter, in order to increase the data-processing frequency, the final optimization of this design is to further parallel the matrix multiplication, with a total of eight cycles to process a pair of I and Q signals. The device diagram is shown in Figure 4.5.

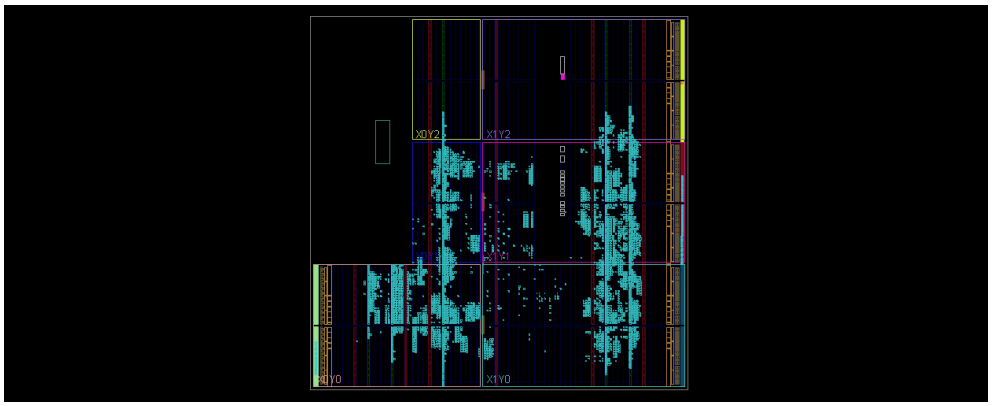


Figure 4.5: Device diagram of hardware design with increased parallelism and retiming

After optimizing the activation function, the hardware circuit consumes significantly fewer resources. To reduce the computation cycles, a space-for-time strategy is adopted, which will increase the hardware resources used. Table 4.10 shows the resource allocation for hardware design with increased

	Static	Dynamic	Total
Power (W)	0.107	0.224	0.331
Percentage	32%	68%	100%

Table 4.9: Power consumption of design using Hardsimoid/Hardtanh function

parallelism.

	Slice LUTs	Slice Registers	Slice	LUT as Logic	DSP
Available	53200	106400	13300	53200	220
Sigmoid LUT	566(8.4%)	200(7.1%)	212(9.5%)	566(8.4%)	0
Tanh LUT	216(3.2%)	120(4.2%)	82(3.7%)	216(3.2%)	0
MAC and Other Ops	5985(88.4%)	2506(88.7%)	1929(86.8%)	5985(88.4%)	159(100%)
Used Totally	6767	2826	2223	6767	159

Table 4.10: Hardware utilization of design with retiming

The increase in resource consumption and computational parallelism will also lead to higher power consumption, as shown in Table 4.11.

	Static	Dynamic	Total
Power(W)	0.110	0.410	0.520
Percentage	21%	79%	100%

Table 4.11: Power consumption of hardware design with retiming

In timing analysis, the clock period is set to 8.3ns, and WNS is -0.073ns. As mentioned before, the timing analysis on the FPGA is only a reference to see whether the circuit optimization is effective in increasing the data processing frequency. The clock period will be reset on the ASIC platform.

Summary
After two rounds of optimization, the hardware design exhibits a dramatic decrease in resource consumption while ensuring favorable timing. And increasing parallelism will lead to the higher power consumption. These trends are illustrated in Table 4.12 and Table 4.13. Specifically, when activation function implementation changes to the Hardsimoid/Hardtanh function, the sigmoid module, tanh module, and overall system resource utilization achieved savings of **18.85X**, **35.25X**, and **3.77X**, respectively, compared to the LUT method (baseline). Based upon this optimization, additional parallelism and retiming yielded savings of **11.66X**, **39.17X**, and **3.03X** for the sigmoid module, tanh module, and overall system resource utilization, respectively, relative to the LUT method (baseline).

Optimization Strategies	**Slice LUTs**			**Reduction**		
	Sigmoid	**Tanh**	**Total**	**Sigmoid**	**Tanh**	**Total**
LUT Method (Baseline)	6598	8461	20522	1x	1x	1x
Hardsigmoid/Hardtanh Function	350	240	5439	**18.85x**	**35.25x**	**3.77x**
Parallel and Retiming	566	216	6767	**11.66x**	**39.17x**	**3.03x**

Table 4.12: Comparison of resources consumption of baseline and two optimizations

4.3.2. ASIC

On the ASIC platform, this work focuses on evaluating the performance of the hardware design. And this design utilizes the optimized code from FPGA and then does simulation, synthesis, and place and route.

Performance
This design can achieve very high frequencies on the ASIC platform. In this experiment, the clock period is set to **0.5ns**, which means the clock frequency is **2GHz**. And there are no timing violations

Optimization Strategies	Power (W)		
	Sigmoid	Tanh	Total
LUT Method (Baseline)	0.213	0.057	0.472
Hardsigmoid/Hardtanh Function	<1%	<1%	0.331
Parallel and Retiming	<1%	<1%	0.520

* note: <1% means the figure is less than 1% of the total power consumption

Table 4.13: Comparison of power consumption of baseline and two optimizations

during synthesis and routing. Therefore, with **8** cycles processing a pair of I and Q signals, the data processing frequency can be calculated as follows:

$$f_{data} = \frac{f_{clk}}{cycles} = \frac{2GHz}{8} = 250MHz \tag{4.1}$$

Due to the Nyquist theorem and the issue of frequency band leakage, the sampling frequency needs to be 3-4 times the data frequency for processing a signal [44]. Therefore, this hardware design can handle wireless communication signals with frequencies up to **70MHz**.

According to the design outlined in the methodology chapter, the fully-connected computation is delayed by one data processing cycle of the GRU, 8 clock cycles. Combined with the overhead, the input-output latency is **15** cycles, or **7.5ns**. And the layout is shown in Figure 4.6.

Table 4.14 lists the frequency, latency, power consumption, and area of this design. Table 4.15 illustrates the power consumption of three components, internal power, switching power, and leakage power. Table 4.16 illustrates the accuracy of DPD on hardware and software.

Frequency (GHz)	Processing Cycles	Latency (ns)	Post-Layout Power (mW)	Area (mm^2)
2.0	**8**	**7.5**	**194.726**	**0.047**

Table 4.14: Performance summary of GRU-based DPD chip

	Internal Power (mW)	Switching Power (mW)	Leakage Power (mW)	Total Power (mW)
Number	105.044	81.529	8.153	194.726
Percentage	53.94%	41.87%	4.19%	100%

Table 4.15: Distribution of power consumption of Post-Layout

	Data Type	Params/Units	TEST-NMSE (dB)	TEST-EVM (dB)	TEST-ACPR (dBc)
Software	32-bit floating-point	502	-43.91	-46.70	**-49.58**
Hardware	10-bit fixed-point	158	-42.79	-46.05	**-49.48**

Table 4.16: Comparison of performance between FP32 model on software and quantized model on hardware

Throughput

Throughput refers to the number of tasks that a system can process or the amount of data that it can transmit per unit of time. In terms of evaluation of neural networks, throughput typically refers to the number of multiplication and accumulation (MAC) operations performed per unit of time, measured in operations per second (Op/s). And 2 operations are performed in 1 MAC operation. In the hardware design of DPD applying the GRU neural network, the number of operations for a part of the I and Q signal primarily consists of three parts, as shown below:

$$Ops_{total} = Ops_{Feature} + Ops_{GRU} + Ops_{Linear} \tag{4.2}$$

The number of operations for the feature is 4, which includes three multiplications and one addition. The number of operations for the GRU is 980, consisting of four parts:

- rt: 10×4+10×10 MAC operations, 10+10 bias additions, and 10 additions for combining two matrix multiplication results.
- zt: 10×4+10×10 MAC operations, 10+10 bias additions, and 10 additions for combining two matrix multiplication results.
- nt: 10×4+10×10 MAC operations, 10+10 bias additions, 10 multiplications of rt with the matrix result, and 10 additions for combining two matrix multiplication results.
- ht: 20 multiplication operations and 20 addition operations.

The fully connected layer has 42 operations, including 2 times 10 MAC operations and 2 bias additions. As a result, the total number of operations are:

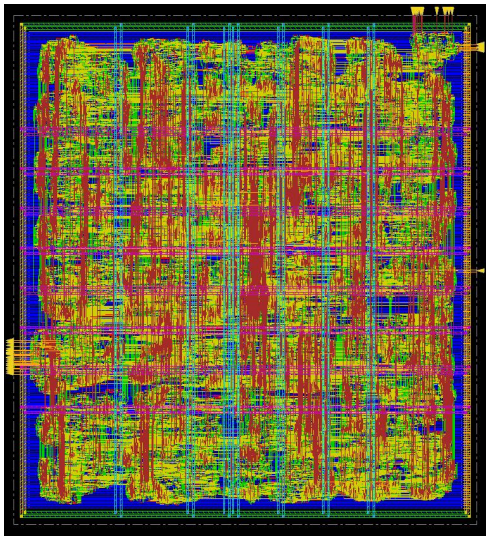
$$Ops_{total} = 4 + 980 + 42 = 1026 \tag{4.3}$$

The throughput can be calculated as follows:

$$Throughput = \frac{Ops_{total}}{cycles} \times frequency \tag{4.4}$$

In this design, a pair of I and Q signals is processed within 8 cycles and the clock frequency is 2GHz. So the ideal throughput should be:

$$Throughput = \frac{1026\ Op}{8\ cycles} \times 2\ GHz = 256.5\ GOp/s \tag{4.5}$$



	Specification			
Technology	22nm CMOS			
Area	0.047 mm^2			
Supply Voltage	0.9 V			
Latency	7.5 ns			
Clock Frequency	2 GHz			
Sample Rate	250 MSPS			
Power Comsumption	Internal Power	Switching Power	Leakage Power	Total Power
	105.044 mW	81.529 mW	8.153 mW	194.726 mW
Throughput	256.5 GOp/s			
Power Efficiency	1.3154 TOp/s/W			

Figure 4.6: Chip layout and performance summary

Power Efficiency

Power efficiency is defined as the ratio of data processing throughput to the system's energy consumption. In neural network accelerators, it indicates how many operations can be achieved per unit power consumption. Its calculation formula is as follows:

$$Power\_Efficiency = \frac{Throughput}{Power} \tag{4.6}$$

In this design, the ideal throughput is 256.5 GOp/s, and the power consumption is 0.195 Watts. Therefore, the power efficiency should be calculated as follows:

$$Power\_Efficiency = \frac{256.5\ GOp/s}{0.195\ W} = 1.3154\ TOp/s/W \tag{4.7}$$

Summary
The DPD chip post-layout, shown in Figure 4.6 is simulated under 22nm CMOS technology and it occupies **0.047 mm^2** area. The DPD chip is the first GRU SoC with the highest throughput, **256.5 GOp/s** and power efficiency, **1.3154 TOp/s/W**. What is more, the chip can work at **2GHz** and can deal with wireless communication signals with bandwidth up to **70MHz**. The chip has a minimum delay as low as **7.5ns**. In the experimental test, the TEST_ACPR can reach up to **-49.48 dBc**.

4.3.3. Comparison With Previous Work

Table 4.17 shows the comparison of performance and power consumption between the proposed accelerator and other state-of-the-art RNN accelerators. The accelerator proposed in this paper demonstrates superior performance compared with the RNN accelerator based on network compression mentioned in paper [50]. Specifically, the proposed design achieves a 38.3% reduction in the number of PEs, while increasing peak throughput by 2.5 times and enhancing power efficiency by 2.2 times. When compared to the accelerator introduced at ISSCC 2017 [40], which operates at a frequency of 50MHz with a 4-bit data width, this design shows a substantial difference in power consumption due to its higher operating frequency and wider data width. However, the difference in peak throughput is relatively small, approximately 8.5%. This work effectively balances a trade-off among power consumption, throughput and operating frequency.

	This work	**ICCS 2020 [50]**	**ISSCC 2017 [40]**
Architecture	ASIC	ASIC	ASIC
Process (nm)	22	28	65
Clock frequency (MHz)	2000	200	50
PEs	158	256	N/A
Voltage (V)	0.9	1.1	0.77
Bit precision (bits)	12	16	4
Peak throughput (GOp/s)	256.5	102.4	280.26
Power consumption (mW)	194.7	166.8	34.6
Power efficiency (TOp/s/W)	1.3154	0.6	8.1

Table 4.17: Comparison of performance between the proposed accelerator and state-of-the-art RNN Accelerators

	This work	**IMC-5G 2019 [21]**	**IMS 2023 [4]**
Architecture	ASIC	FPGA	FPGA
Process (nm)	22	16	16
Clock frequency (MHz)	2000	312.5	300
Bit precision (bits)	12	16	14
Area/ Slice registers	0.047 mm^2	11258	110337
Power consumption (W)	0.194	0.374	12.9
Test-ACPR (dBc)	-49.48	-46.2	-47.1
Bandwidth (MHz)	75	400	75

Table 4.18: Comparison of resource consumption and performance between GRU-based DPD hardware design and state-of-the-art DPD hardware designs

Table 4.18 shows the comparison of resource consumption and performance between the proposed

GRU-based DPD hardware design and other state-of-the-art DPD hardware designs implemented on FPGA. The DPD hardware implementation in paper [21] is based on the Volterra series, while implementation in paper [4] is a PWL-based model. Both implementations employ a parallel-processing DPD structure, where multiple DPD units are duplicated to widen the signal bandwidth. As indicated in Table 4.18, the proposed design demonstrates a significant difference in area/slice register consumption compared to the other designs, while the bandwidth of the processed signal is relatively smaller. Nevertheless, the accuracy of the proposed design is much higher, with an improvement of 3.28 dBc compared to the implementation in paper [21] and 2.38 dBc compared to implementation in paper [4], respectively. Therefore, the DPD hardware structure presented in this work is well-suited for embedded applications that require low bandwidth and power consumption but demand high accuracy.

5

Conclusions and Future works

5.1. Conclusion

PAs are a critical component of wireless communication systems, but their efficiency and linearity are often at odds. DPD technology compensates for the non-linearity of PAs, making it widely used in wireless communication systems. However, as bandwidth increases, traditional DPD models become more complex, and the miniaturization of base stations creates a growing demand for low-power, high-efficiency DPD chips. Inspired by OpenDPD, the use of GRU neural networks in DPD research has shown prominent results. This paper is dedicated to designing a high-performance GRU-based hardware system to meet these demands.

In software, to simplify the computational complexity of the hardware, this paper employs QAT quantization strategy to convert the floating-point model to a fixed-point model. During the retraining process, the minor deviations introduced by the quantization of the rounded activation functions result in negligible parameter changes during backpropagation, hindering the fixed-point model from achieving the original accuracy of the floating-point model. To mitigate this issue, this paper proposes the use of linear functions, Hardsigmoid and Hardtanh, to quantize the activation functions. These functions amplify the variations caused by precision loss, making the parameter updates more sensitive to accuracy loss. This approach facilitates the quantization of the floating-point model to a lower bit-width fixed-point model while preserving higher accuracy.

In hardware, given the high frequency and low latency requirements for processing wireless communication signals, this design highly enhances the parallelism of matrix multiplication and employs inter-layer pipeline and retiming techniques for timing optimization. To optimize resource utilization, this design proposes an efficient data flow. Additionally, given the small number of parameters in the GRU neural network and the high degree of hardware parallelism, the use of LUTs method for implementing activation functions results in an imbalanced resource distribution, with over 73.4% of resources allocated to activation functions. This disproportionate allocation leads to significant resource wastage. To address this issue, this paper introduces the use of linear functions, Hardsigmoid and Hardtanh, to optimize the activation functions.

The experimental results illustrate that selecting features I, Q, $|x|^2$, $|x|^4$, and a hidden size of **10** for GRU neural network effectively trades off parameter count and accuracy. Under the same parameter precision, quantizing activation functions with Hardsigmoid/Hardtanh results in an ACPR improvement of 1-2 dBc compared to LUT methods. The performance of the hardware on the FPGA indicates that the resource utilization using Hardsigmoid/Hardtanh shows significant improvement compared to the LUT method (baseline), specifically demonstrated as **18.85x** reduction on sigmoid module, **35.25x** reduction on tanh module and **3.77x** reduction of the total system. This paper evaluates the hardware design on the ASIC platform. Simulated under 22nm CMOS technology, the DPD chip occupies **0.047 mm^2**, with the highest throughput, **256.5 GOp/s** and power efficiency **1.3154 TOp/s/W**. In addition, the chip works at **2GHz**, capable of handling signals with bandwidth up to **70MHz**.

5.2. Future work

Future work will focus on several key areas to enhance the proposed GRU-based DPD chip further:

- Scalability and Flexibility: This work implements and evaluates the hardware for a single optimal data flow configuration. Future work could focus on the reconfigurability of the DPD chip to adapt to different application scenarios.
- Power Optimization: This work primarily focuses on optimizing area and timing while there remains significant potential for improvement in power consumption. Dynamic voltage and frequency scaling (DVFS) and power gating can be used to further reduce the power consumption of the DPD hardware system.
- Comprehensive Testing and Validation: This work only performs simulation when verifying the accuracy of the DPD chip and select the only data set provides by OpenDPD. So the test coverage is insufficient. Future work can focus on the robustness and reliability.

Bibliography

[1] Manal Talib Ali and Bassam H Abed. “The Proposition of Three Approaching Ways to Implement Tan-sigmoid Activation Function in FPGA”. In: *Engineering and Technology Journal* 40.2 (2022), pp. 311–321. DOI: 10.30684/etj.v40i2.2160.

[2] Jeffrey G. Andrews et al. “What Will 5G Be?” In: *IEEE Journal on Selected Areas in Communications* 32.6 (2014), pp. 1065–1082. DOI: 10.1109/JSAC.2014.2328098.

[3] Mart van Baalen et al. “Simulated Quantization, Real Power Savings”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2022, pp. 2756–2760. DOI: 10.1109/CVPRW56347.2022.00311.

[4] Hoda Barkhordar-Pour et al. “Real-Time FPGA-Based Implementation of Digital Predistorters for Fully Digital MIMO Transmitters”. In: *2023 IEEE/MTT-S International Microwave Symposium - IMS 2023*. 2023, pp. 263–266. DOI: 10.1109/IMS37964.2023.10188033.

[5] Federico Boccardi et al. “Five disruptive technology directions for 5G”. In: *IEEE Communications Magazine* 52.2 (2014), pp. 74–80. DOI: 10.1109/MCOM.2014.6736746.

[6] Tommaso Cappello et al. “Power Consumption and Linearization Performance of a Bit- and Frequency-Scalable AM/AM AM/PM Pre-Distortion on FPGA”. In: *2022 International Workshop on Integrated Nonlinear Microwave and Millimetre-Wave Circuits (INMMiC)*. 2022, pp. 1–3. DOI: 10.1109/INMMiC54248.2022.9762118.

[7] H.-H. Chen et al. “Joint Polynomial and Look-Up-Table Predistortion Power Amplifier Linearization”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 53.8 (2006), pp. 612–616. DOI: 10.1109/TCSII.2006.877278.

[8] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014). DOI: 10.48550/arXiv.1406.1078.

[9] Junyoung Chung et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014). DOI: 10.48550/arXiv.1412.3555.

[10] Rahul Dey and Fathi M. Salem. “Gate-variants of Gated Recurrent Unit (GRU) neural networks”. In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2017, pp. 1597–1600. DOI: 10.1109/MWSCAS.2017.8053243.

[11] Heba Mohammed Fadhil and Zinah Osamah Dawood. “Evolutionary Perspective of Mobile Communication Technologies”. In: *2018 International Conference on Computer and Applications (ICCA)*. 2018, pp. 80–84. DOI: 10.1109/COMAPP.2018.8460233.

[12] Fang Fang, Tsuhan Chen, and Rob A. Rutenbar. “Floating-point bit-width optimization for low-power signal processing applications”. In: *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 3. 2002, pp. III-3208-III–3211. DOI: 10.1109/ICASSP.2002.5745332.

[13] Wenji Fang et al. “Annotating Slack Directly on Your Verilog: Fine-Grained RTL Timing Evaluation for Early Optimization”. In: *arXiv preprint arXiv:2403.18453* (2024). DOI: 10.48550/arXiv.2403.18453.

[14] R. H. Flake. “Volterra series representation of nonlinear systems”. In: *Transactions of the American Institute of Electrical Engineers, Part II: Applications and Industry* 81.6 (1963), pp. 330–335. DOI: 10.1109/TAI.1963.6371765.

[15] Altaf Abdul Gaffar, Jonathan A. Clarke, and George A. Constantinides. “PowerBit - power aware arithmetic bit-width optimization”. In: *2006 IEEE International Conference on Field Programmable Technology*. 2006, pp. 289–292. DOI: 10.1109/FPT.2006.270330.

[16] Chang Gao et al. “DeltaRNN: A power-efficient recurrent neural network accelerator”. In: *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2018, pp. 21–30. DOI: 10.1145/3174243.3174261.

[17] Fadhel M. Ghannouchi and Oualid Hammi. “Behavioral modeling and predistortion”. In: *IEEE Microwave Magazine* 10.7 (2009), pp. 52–64. DOI: 10.1109/MMM.2009.934516.

[18] Estefanía Guillena et al. “Reconfigurable DPD Based on ANNs for Wideband Load Modulated Balanced Amplifiers Under Dynamic Operation From 1.8 to 2.4 GHz”. In: *IEEE Transactions on Microwave Theory and Techniques* 70.1 (2022), pp. 453–465. DOI: 10.1109/TMTT.2021.3091672.

[19] Song Han et al. “Ese: Efficient speech recognition engine with sparse lstm on fpga”. In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2017, pp. 75–84. DOI: 10.1145/3020078.3021745.

[20] Ziaul Hasan, Hamidreza Boostanimehr, and Vijay K. Bhargava. “Green Cellular Networks: A Survey, Some Research Issues and Challenges”. In: *IEEE Communications Surveys & Tutorials* 13.4 (2011), pp. 524–540. DOI: 10.1109/SURV.2011.092311.00031.

[21] Hai Huang, Jingjing Xia, and Slim Boumaiza. “Parallel-Processing-Based Digital Predistortion Architecture and FPGA Implementation for Wide-band 5G Transmitters”. In: *2019 IEEE MTT-S International Microwave Conference on Hardware and Systems for 5G and Beyond (IMC-5G)*. 2019, pp. 1–3. DOI: 10.1109/IMC-5G47857.2019.9160360.

[22] “IEEE Standard for Floating-Point Arithmetic”. In: *IEEE Std 754-2008* (2008), pp. 1–70. DOI: 10.1109/IEEESTD.2008.4610935.

[23] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. “An empirical exploration of recurrent network architectures”. In: *International conference on machine learning*. PMLR. 2015, pp. 2342–2350.

[24] Jaehyeong Kim and K Konstantinou. “Digital predistortion of wideband signals based on power amplifier model with memory”. In: *Electronics Letters* 37.23 (2001), p. 1.

[25] Tugce Kobal and Anding Zhu. “Digital Predistortion of RF Power Amplifiers With Decomposed Vector Rotation-Based Recurrent Neural Networks”. In: *IEEE Transactions on Microwave Theory and Techniques* 70.11 (2022), pp. 4900–4909. DOI: 10.1109/TMTT.2022.3209658.

[26] Gang Li et al. “Instant Gated Recurrent Neural Network Behavioral Model for Digital Predistortion of RF Power Amplifiers”. In: *IEEE Access* 8 (2020), pp. 67474–67483. DOI: 10.1109/ACCESS.2020.2986816.

[27] Hongmin Li et al. “Vector Decomposed Long Short-Term Memory Model for Behavioral Modeling and Digital Predistortion for Wideband RF Power Amplifiers”. In: *IEEE Access* 8 (2020), pp. 63780–63789. DOI: 10.1109/ACCESS.2020.2984682.

[28] Hao Lin et al. “Optimization design of FPGA-based look-up-tables for linearizing RF power amplifiers”. In: *2011 International Conference on Electronics, Communications and Control (ICECC)*. 2011, pp. 2731–2734. DOI: 10.1109/ICECC.2011.6066767.

[29] Tomas Mikolov et al. “Learning longer memory in recurrent neural networks”. In: *arXiv preprint arXiv:1412.7753* (2014). DOI: 10.48550/arXiv.1412.7753.

[30] D.R. Morgan et al. “A Generalized Memory Polynomial Model for Digital Predistortion of RF Power Amplifiers”. In: *IEEE Transactions on Signal Processing* 54.10 (2006), pp. 3852–3860. DOI: 10.1109/TSP.2006.879264.

[31] Daniel Neil et al. “Delta networks for optimized recurrent network computation”. In: *International conference on machine learning*. PMLR. 2017, pp. 2584–2593.

[32] F.H. Raab et al. “Power amplifiers and transmitters for RF and microwave”. In: *IEEE Transactions on Microwave Theory and Techniques* 50.3 (2002), pp. 814–826. DOI: 10.1109/22.989965.

[33] Theodore S. Rappaport et al. “Millimeter Wave Mobile Communications for 5G Cellular: It Will Work!” In: *IEEE Access* 1 (2013), pp. 335–349. DOI: 10.1109/ACCESS.2013.2260813.

[34] Meenakshi Rawat, Karun Rawat, and Fadhel M. Ghannouchi. “Adaptive Digital Predistortion of Wireless Power Amplifiers/Transmitters Using Dynamic Real-Valued Focused Time-Delay Line Neural Networks”. In: *IEEE Transactions on Microwave Theory and Techniques* 58.1 (2010), pp. 95–104. DOI: 10.1109/TMTT.2009.2036334.

[35] F. Richter and G. Fettweis. “Cellular Mobile Network Densification Utilizing Micro Base Stations”. In: *2010 IEEE International Conference on Communications*. 2010, pp. 1–6. DOI: 10.1109/ICC.2010.5502299.

[36] Fred Richter, Albrecht J. Fehske, and Gerhard P. Fettweis. “Energy Efficiency Aspects of Base Station Deployment Strategies for Cellular Networks”. In: *2009 IEEE 70th Vehicular Technology Conference Fall*. 2009, pp. 1–5. DOI: 10.1109/VETECF.2009.5379031.

[37] Parna Sabeti et al. “Frequency Synchronization for OFDM-Based Massive MIMO Systems”. In: *IEEE Transactions on Signal Processing* 67.11 (2019), pp. 2973–2986. DOI: 10.1109/TSP.2019.2911249.

[38] Robin M Schmidt. “Recurrent neural networks (rnns): A gentle introduction and overview”. In: *arXiv preprint arXiv:1912.05911* (2019). DOI: 10.48550/arXiv.1912.05911.

[39] A. F. M. Shahen Shah. “A Survey From 1G to 5G Including the Advent of 6G: Architectures, Multiple Access Techniques, and Emerging Technologies”. In: *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. 2022, pp. 1117–1123. DOI: 10.1109/CCWC54503.2022.9720781.

[40] Dongjoo Shin et al. “14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks”. In: *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. 2017, pp. 240–241. DOI: 10.1109/ISSCC.2017.7870350.

[41] Ying Fai Tong, Rob A Rutenbar, and David F Nagle. “Minimizing floating-point power dissipation via bit-width reduction”. In: *Power-Driven Microarchitecture Workshop*. Citeseer. 1998.

[42] Joel Vuolevi and Timo Rahkonen. *Distortion in RF power amplifiers*. Artech house, 2003.

[43] Shuo Wang et al. “C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs”. In: *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2018, pp. 11–20. DOI: 10.1145/3174243.3174253.

[44] Zonghao Wang et al. “Low Feedback Sampling Rate Digital Predistortion for Wideband Wireless Transmitters”. In: *IEEE Transactions on Microwave Theory and Techniques* 64.11 (2016), pp. 3528–3539. DOI: 10.1109/TMTT.2016.2602216.

[45] Dan Wu et al. “A Field Trial of f-OFDM toward 5G”. In: *2016 IEEE Globecom Workshops (GC Wkshps)*. 2016, pp. 1–6. DOI: 10.1109/GLOCOMW.2016.7848810.

[46] Shangbin Wu et al. “Frequency and quadrature amplitude modulation for 5G networks”. In: *2016 European Conference on Networks and Communications (EuCNC)*. 2016, pp. 1–5. DOI: 10.1109/EuCNC.2016.7560993.

[47] Yizhuo Wu et al. “MP-DPD: Low-Complexity Mixed-Precision Neural Networks for Energy-Efficient Digital Predistortion of Wideband Power Amplifiers”. In: *IEEE Microwave and Wireless Technology Letters* 34.6 (2024), pp. 817–820. DOI: 10.1109/LMWT.2024.3386330.

[48] Yizhuo Wu et al. “OpenDPD: An Open-Source End-to-End Learning & Benchmarking Framework for Wideband Power Amplifier Modeling and Digital Pre-Distortion”. In: *arXiv preprint arXiv:2401.08318* (2024). DOI: 10.48550/arXiv.2401.08318.

[49] Zehai Wu et al. “Development challenges for 5G base station antennas”. In: *2018 International Workshop on Antenna Technology (iWAT)*. 2018, pp. 1–3. DOI: 10.1109/IWAT.2018.8379163.

[50] Wentao Zhu et al. “Design of High Performance RNN Accelerator Based on Network Compression”. In: *2020 IEEE 2nd International Conference on Circuits and Systems (ICCS)*. 2020, pp. 6–10. DOI: 10.1109/ICCS51219.2020.9336599.