CHALMERS
UNIVERSITY OF TECHNOLOGY

Precision Exploration and Underflow Mitigation for BAPS Digital Predistortion

Master's thesis in Embedded Electronic System Design

Yiran Duan

Department of Microtechnology and Nanoscience
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

Master's thesis 2026

Precision Exploration and Underflow Mitigation for BAPS Digital Predistortion

Yiran Duan

Department of Microtechnology and Nanoscience
Chalmers University of Technology
Gothenburg, Sweden 2026

Precision Exploration and Underflow Mitigation for BAPS Digital Predistortion
Yiran Duan

Supervisor: Per Larsson-Edefors, Department of Microtechnology and Nanoscience
Examiner: Lena Peterson, Department of Microtechnology and Nanoscience

Master's Thesis 2026
Department of Microtechnology and Nanoscience
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Precision Exploration and Underflow Mitigation for BAPS Digital Predistortion
Yiran Duan
Department of Microtechnology and Nanoscience
Chalmers University of Technology

Abstract

Digital predistortion (DPD) is widely used to compensate for the nonlinear behaviour of radio-frequency power amplifiers (PAs). The basis-propagating selection (BAPS) algorithm builds DPD models by reusing previously computed basis functions, making it attractive for hardware implementation. However, when reduced-precision floating-point arithmetic is used, these sequential multiplication chains create dynamic range challenges. This thesis investigates how the number of mantissa bits (t), the number of exponent bits (w), and the number of BAPS basis functions (R) affect DPD linearisation performance, evaluated using normalised mean square error (NMSE) and adjacent channel power ratio (ACPR), and hardware cost.

A Python-based simulation framework using the APyTypes library is developed to evaluate custom floating-point configurations and generate test vectors for register-transfer-level (RTL) verification. A joint sweep of R and t produces a two-dimensional NMSE heatmap. For the selected BAPS sequence and test case, the performance plateau is reached at $R = 6$ and $t = 7$; larger values give no measurable NMSE improvement. This extends the fixed-R mantissa-width result reported in prior work by showing the joint effect of R and t. Reducing t from 23 to 7 reduces the synthesised area by approximately 78% without measurable loss in DPD linearisation performance.

The exponent-width study shows that, without scaling, intermediate Type II products underflow at $w = 5$, removing important basis functions and making the predistorter ineffective. This thesis therefore uses power-of-two scaling, which raises these products using exponent shifts only, without additional floating-point multipliers. At $t = 7$, the scaled $(w, t) = (5, 7)$ design recovers performance close to the reference level and uses 5.9% less area than the scaled $(6, 7)$ design. At $w = 4$, the model coefficients underflow, so $w = 5$ is the practical lower bound for the studied signal and PA operating point.

The configuration $(w, t) = (5, 7)$ with power-of-two scaling reduces the synthesised total area to 21% of the single-precision baseline, while meeting the common 5 ns clock-period constraint used for the synthesis of all configurations, including the single-precision baseline. Selected configurations are also validated using the RF WebLab measurement platform and PA gate-voltage robustness tests. The results show that reduced-precision BAPS DPD can provide substantial hardware savings while maintaining linearisation performance close to the full-precision reference.

Keywords: digital predistortion, BAPS, custom floating-point, hardware implementation, precision exploration, underflow, RF power amplifier

Acknowledgements

I would like to thank my supervisor, Per Larsson-Edefors, for his guidance and support throughout this project. His feedback and advice helped me stay on track and improve my work at every stage. I would also like to thank my examiner, Lena Peterson, for her time and careful review of this thesis. I am also grateful to Lars Svensson for connecting me with my supervisor and helping me find this thesis project. Finally, I want to thank my family for their encouragement and support during my time studying abroad, and someone close to me for being a patient listener and for the kind words and feedback whenever I needed them. Their belief in me made this journey possible.

Yiran Duan, Gothenburg, July 2026

Contents

1

Introduction

Radio-frequency power amplifiers (PAs) consume a large share of the power in wireless transmitter systems [1]. To improve energy efficiency, a PA is usually operated close to its saturation point. At this point, however, the PA becomes nonlinear. The output signal is distorted, and spectral energy spreads into adjacent channels [2]. This is a serious problem in modern communication systems, where emission limits are strict and spectrum resources are limited.

Digital predistortion (DPD) is a common technique used to address this problem. A DPD block is placed before the PA and predistorts the input signal before it enters the PA. As a result, the combined response of the DPD and the PA becomes close to linear [3]. In this way, the PA can still operate in an efficient region while meeting linearity requirements.

In real systems, DPD must run in hardware at the full sample rate [4]. This creates a strict throughput requirement, so the DPD model must be accurate while also being simple enough to fit limited hardware resources and power budgets. The basis-propagating selection (BAPS) algorithm [5] was developed to reduce DPD computational complexity. Instead of using a large predefined model structure, BAPS builds the model step by step by reusing previously computed basis functions. This reduces the number of arithmetic operations needed at runtime, and the operation sequence is fixed offline, so the hardware only has to execute a compact set of predetermined computations. These properties make BAPS attractive for area-efficient hardware implementation, while high data throughput depends on the selected RTL architecture and its degree of pipelining.

Even when the model is computationally efficient, a hardware DPD system still has to use finite-precision arithmetic. The floating-point wordlength affects both hardware area and DPD performance. Too few bits can cause numerical errors and reduce linearisation quality. Too many bits increase hardware cost. Choosing a suitable precision is therefore an important design problem.

For BAPS, this problem is more complicated than for direct-form DPD models such as the memory polynomial (MP) and generalized memory polynomial (GMP) models. In these models, basis functions are computed directly from the input signal. In BAPS, they are built through a sequence of multiplications, where each new function can use values computed earlier. As the number of basis functions R increases, the outputs of Type II basis-function operations can be reused as operands in later operations, and their magnitudes can become much smaller or much larger. This increases the dynamic range requirement. This thesis studies how the BAPS

basis-function sequence affects floating-point precision requirements. The sequence is determined offline by a greedy search. The study focuses on exponent widths below the range explored in earlier work.

1.1 Related Work

This section reviews earlier work on DPD models, hardware-oriented precision design, and BAPS-based implementations. It then explains the research gap that this thesis aims to address.

DPD Modeling and Complexity Reduction

Many DPD models are based on the Volterra series, which is a general way to describe nonlinear systems with memory [6]. In practice, however, the full Volterra series contains too many terms for practical DPD implementation. Two common simplified Volterra-based models are the memory polynomial (MP) model [7] and the generalized memory polynomial (GMP) model [8]. These models offer a useful balance between accuracy and complexity. However, as signal bandwidth increases in 5G and beyond, memory effects become more significant. More terms are then needed to model the PA accurately, which increases the hardware cost.

To reduce complexity, several model-selection and pruning methods have been studied. Orthogonal matching pursuit (OMP) [9], doubly OMP (DOMP) [10], and the least absolute shrinkage and selection operator (LASSO) [11] select a small number of terms from a large candidate set. Dimensionality-reduction methods based on principal component analysis (PCA) [12] have also been used to reduce model complexity. These methods reduce the number of coefficients, but they do not change the basic structure of the remaining computations.

The BAPS algorithm [5] uses a different approach. Instead of starting from a large predefined model structure, it builds the model step by step and reuses previously computed results. The operation sequence is decided offline by a greedy search and then fixed for hardware use. At runtime, the hardware only needs to carry out a compact set of predetermined computations. At equal computational cost, BAPS has been shown to outperform MP, GMP, OMP, and DOMP models across different PA types [5].

Floating-Point Precision in Hardware DPD

Hardware DPD requires a suitable numerical format. Fixed-point arithmetic is area-efficient, but it needs careful scaling to avoid overflow and underflow. This becomes difficult when the signal has a high peak-to-average power ratio (PAPR) [13], because the peaks are much larger than the average and a fixed scaling cannot cover both well. Floating-point arithmetic can handle a wider dynamic range more easily, but it usually requires more hardware area. Custom floating-point formats, which use fewer bits than IEEE 754 single precision, have therefore attracted interest as a way to reduce area while keeping the flexibility of floating-point arithmetic [14, 15].

In some hardware DPD studies, model coefficients are first optimised in full floating-point precision and then quantised for hardware use [15]. In other words, precision is usually considered only after optimisation. This is reasonable for models whose basis functions are computed directly and independently from the input signal. For BAPS, however, the situation is different. Its basis functions are built step by step from earlier intermediate values, so quantisation effects can propagate through the construction process and affect later basis functions.

Precision Challenges Specific to BAPS

In BAPS, each basis function depends on previously computed ones. When the number of basis functions R is large, later basis functions may be the result of several chained multiplications. The magnitude of these intermediate values depends on both the input signal level and the depth of the computation chain. With too few exponent bits, some values may fall outside the representable range and underflow to zero before the final output is formed. This dynamic range problem is exacerbated by the sequential structure of BAPS and can become more significant as R increases.

Yu et al. [16] presented a recent systematic study of floating-point precision for BAPS-based DPD. They showed that the mantissa width t is the main factor for performance, with a critical degradation threshold at $t = 7$ bits, while the exponent width w could be reduced to 5 bits with little performance loss. The configuration $(w, t) = (5, 7)$ was reported as a useful trade-off, with about 80% area reduction compared with single precision and less than 0.4 dB loss in normalized mean square error (NMSE) performance.

However, that study evaluated only fixed BAPS configurations and did not map how performance changes continuously with R. The failure mechanism at low exponent widths also remains insufficiently explained. Because of this, two important questions form the starting point of this thesis. First, it is not clear how precision sensitivity changes when R changes. Second, the mechanism behind performance degradation at low exponent widths needs further investigation.

1.2 Purpose and Goal

The purpose of this thesis is to improve the understanding of floating-point precision requirements in DPD implementations based on BAPS. The goal is to support robust and hardware-efficient implementations across different model sizes and precision settings.

This thesis extends the work of [16] in two directions. The first direction concerns the number of active basis functions R. By sweeping R and mantissa width t together, this work builds an (R, t) heatmap of NMSE performance. This makes it possible to study how precision sensitivity changes with model size.

The second direction concerns the lower limit of the exponent width. This thesis studies the dynamic range behaviour of BAPS basis functions at low exponent widths. Based on this analysis, a power-of-two scaling method is proposed for the

basis function values, with the aim of reducing underflow and recovering performance at low hardware cost. The work also includes a Verilog implementation of the BAPS DPD pipeline, validation with real PA measurements on the RF WebLab testbed [17], and a gate-voltage robustness test to assess performance when the PA operating point changes.

Based on these goals, this thesis addresses the following research questions:

- How does DPD performance change when the number of basis functions R and the mantissa width t are varied together?
- What causes performance degradation at low exponent widths, and can scaling the basis function values recover this loss at low hardware cost?

1.3 Thesis Outline

The rest of this thesis is organised as follows.

Chapter 2 presents the technical background for this work. It introduces PA nonlinearity and DPD, explains the BAPS algorithm and its basis construction process, and reviews floating-point representation for hardware DPD design.

Chapter 3 describes the methodology used in this work. It explains the experimental workflow, the tools selected for simulation and synthesis, the precision exploration strategy for the (R, t) and exponent-width sweeps, and the approach used for hardware complexity analysis.

Chapter 4 presents the preliminary study. It describes the Python simulation framework used for coefficient identification and precision evaluation. It also analyses the dynamic range problem that arises in Type II basis function computation and proposes the power-of-two scaling method as a solution.

Chapter 5 describes the hardware design of the BAPS DPD pipeline. It presents the register-transfer-level (RTL) architecture, the finite state machine design for both pipeline stages, and the hardware implementation of the power-of-two scaler and the Type II operator module.

Chapter 6 presents the results. These include the software–RTL consistency verification, the (R, t) heatmap from the precision sweep, the effect of exponent-width reduction and power-of-two scaling on DPD performance, the PA operating point robustness analysis, and the hardware complexity results from logic synthesis.

Chapter 7 discusses the main findings and their meaning for hardware design. It also covers the limitations of the work and directions for future research.

Chapter 8 concludes the thesis.

2

Technical Background

This chapter introduces the background needed to understand the work presented in this thesis. It starts with the nonlinear behaviour of power amplifiers and how digital predistortion is used to compensate for it. It then describes the BAPS algorithm and explains why its sequential structure creates specific numerical challenges. Finally, it covers floating-point number representation and the use of floating-point intellectual property (IP) cores in hardware design.

2.1 Power Amplifier Nonlinearity

A power amplifier (PA) is used in a wireless transmitter to increase the power of a signal before it is sent over the air [3]. In practice, the PA is usually one of the most power-consuming parts of the transmitter. To improve energy efficiency, it should operate close to its saturation point, where the output power is high.

The problem is that a PA does not behave linearly in this region. At low input amplitudes, the output is approximately a scaled version of the input. As the input amplitude increases, the gain starts to decrease. This effect is known as gain compression, or AM/AM distortion. The output phase can also shift with the input amplitude, which is called AM/PM distortion. Both effects distort the transmitted signal and can cause energy to spread into neighbouring frequency channels, an effect known as spectral regrowth [2].

Modern communication signals often have a high peak-to-average power ratio (PAPR). This means that the signal contains short peaks that are much larger than its average level. To avoid severe distortion of these peaks, the PA must operate with some backoff, that is, with its average output power kept well below the maximum, which reduces efficiency. In general, a higher PAPR requires more backoff and leads to lower efficiency [2].

Figure 2.1 shows the measured AM/AM and AM/PM characteristics of the PA used in this thesis. The measurements used for this figure were obtained from the RF WebLab platform, which provides remote access to a real PA measurement setup. The platform itself is described in the methodology chapter. The left plot shows clear gain compression at higher input amplitudes, where the output deviates from the ideal linear response. The right plot shows the phase shift as a function of input amplitude. The spread at low amplitudes is mainly caused by measurement noise. At higher amplitudes, the phase shift remains small, which indicates that AM/PM distortion is not the dominant effect for this PA.

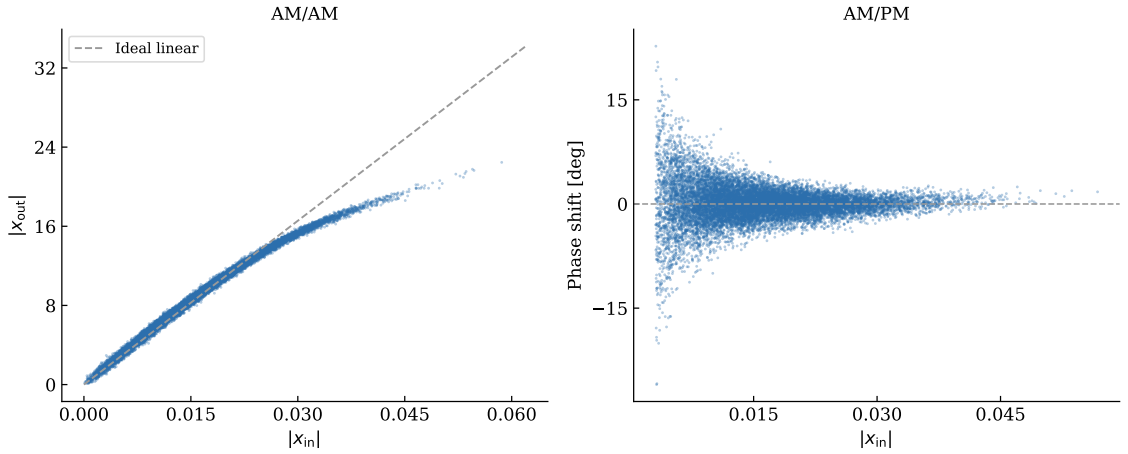


Figure 2.1: Measured AM/AM (left) and AM/PM (right) characteristics of the PA, obtained using the RF WebLab platform.

2.1.1 Memory Effects

Beyond the instantaneous nonlinearity described above, PAs also show memory effects. This means that the output at a given time depends not only on the current input, but also on inputs from earlier time steps. Memory effects come from two main sources. Thermal memory is caused by temperature changes in the device as the signal power varies over time. Electrical memory comes from reactive components in the bias network and from charge trapping in the semiconductor material [3]. These effects become more significant when the signal bandwidth is large, as is common in modern communication systems.

Due to nonlinearity and memory effects, a PA without correction will distort the signal and cause spectral regrowth into neighbouring channels. This motivates the use of digital predistortion, which is described in the next section.

2.1.2 PA Operating Point

The nonlinear behaviour of a PA also depends on its bias conditions [18]. In a field-effect transistor (FET) based PA, the gate voltage V_g controls the operating point of the device. A higher gate voltage typically moves the PA towards a more linear region, while a lower gate voltage pushes it closer to its saturation point and increases the degree of nonlinearity. As a result, the AM/AM and AM/PM characteristics change with V_g, and a DPD model identified at one operating point may not perform equally well at another.

This sensitivity to the operating point is relevant for practical deployment, where the PA bias may drift due to temperature changes, ageing, or intentional adjustment for efficiency. A DPD model identified at one operating point may not maintain the same linearisation performance when V_g changes. Therefore, operating-point robustness should be evaluated when assessing a practical DPD implementation. The operating-point evaluation procedure used in this thesis is described in Chapter 3.

2.2 Digital Predistortion

Digital predistortion is a technique used to compensate for PA nonlinearity. The idea is to pre-modify the input signal before it enters the PA so that the combined response of the DPD and the PA becomes close to linear [3]. In this way, the PA can still operate in an efficient region while meeting linearity requirements. In practice, the DPD model must run at the full signal sample rate, which means it needs to be accurate and computationally simple enough for hardware implementation.

2.2.1 System Architecture

A typical DPD system operates as follows. The input signal first passes through the DPD block, which modifies it based on a model of the PA. The modified signal is then converted to analogue, upconverted to radio frequency, and fed into the PA. The PA output is measured, downconverted back to baseband, and digitised. This measured output is compared with the desired output to identify or update the DPD model coefficients. Figure 2.2 shows a simplified block diagram of this architecture [5].

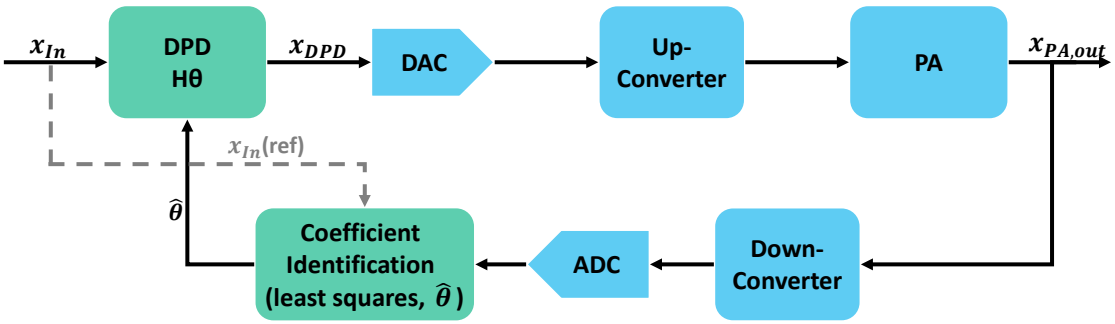


Figure 2.2: Simplified block diagram of a DPD system. The input signal is predistorted before the PA, and the PA output is fed back to identify the model coefficients.

2.2.2 Coefficient Identification

The DPD model is described by a set of coefficients θ. These coefficients are found using a least-squares method. The basis functions of the model are collected into a matrix H, where each column contains one basis function evaluated over all time samples and each row corresponds to one time instant. The desired output is stored in a vector y. The coefficients are then estimated as

$$\hat{\theta} = (H^H H)^{-1} H^H y, \tag{2.1}$$

where $(\cdot)^H$ denotes the conjugate transpose. This gives the coefficients that minimise the mean square error between the model output and the desired output.

A common approach for identifying these coefficients in practice is the indirect learning architecture (ILA) [6]. In the ILA, the PA output is used as the input to

a post-inverse model, and the coefficients are identified by fitting this post-inverse to the desired signal. The identified coefficients are then copied to the pre-inverse DPD block. This approach avoids the need to directly invert the PA model and is widely used because of its simplicity and reliable convergence.

2.2.3 DPD Models

Many DPD models are based on the Volterra series, which can describe a broad class of nonlinear systems with memory [6]. Because the full Volterra series is too large to use directly, simpler models are commonly used instead. The MP model is one of the most widely used [7]:

$$y(n) = \sum_{k=1}^{K} \sum_{m=0}^{M} a_{km} x(n-m)|x(n-m)|^{k-1}, \tag{2.2}$$

where $x(n)$ is the input signal, K is the nonlinearity order, M is the memory depth, and a_{km} are the model coefficients.

The GMP model extends the MP model by adding cross-terms between the signal at different time instants [8]:

$$\begin{aligned} y(n) = & \sum_{k=1}^{K_a} \sum_{m=0}^{M_a} a_{km} x(n-m)|x(n-m)|^{k-1} \\ & + \sum_{k=1}^{K_b} \sum_{m=0}^{M_b} \sum_{l=1}^{L_b} b_{kml} x(n-m)|x(n-m-l)|^{k-1} \\ & + \sum_{k=1}^{K_c} \sum_{m=0}^{M_c} \sum_{l=1}^{L_c} c_{kml} x(n-m)|x(n-m+l)|^{k-1}. \end{aligned} \tag{2.3}$$

The GMP model can capture more complex memory effects, but it also has a larger number of terms and a higher hardware cost.

2.2.4 Performance Metrics

Two metrics are used in this thesis to measure how well a DPD system corrects PA distortion. The normalized mean square error (NMSE) measures the difference between the desired output and the actual PA output in the time domain [2]:

$$\mathrm{NMSE} = 10 \log_{10} \left(\frac{\sum_n |y_{\mathrm{desired}}(n) - y_{\mathrm{actual}}(n)|^2}{\sum_n |y_{\mathrm{desired}}(n)|^2} \right) \quad [\mathrm{dB}]. \tag{2.4}$$

A lower NMSE value means that the output is closer to the desired signal.

The adjacent channel power ratio (ACPR) measures how much signal power leaks into the frequency channels next to the main channel [2]:

$$\mathrm{ACPR} = 10 \log_{10} \left(\frac{P_{\mathrm{adjacent}}}{P_{\mathrm{main}}} \right) \quad [\mathrm{dBc}]. \tag{2.5}$$

A lower ACPR value means less spectral leakage. The unit dBc expresses power relative to the main channel power, so it is a ratio between two power levels rather than an absolute value. NMSE and ACPR are both reported on a decibel scale, and lower values indicate better linearisation performance.

2.3 The BAPS Algorithm

The BAPS algorithm [5] was proposed as a way to build DPD models that are both accurate and simple enough for hardware implementation. Rather than starting from a large fixed set of terms, BAPS builds a small set of basis functions step by step. Each new basis function is formed from functions that have already been computed, so repeated computation can be avoided.

2.3.1 Basis Function Construction

The first basis function is always the input signal itself:

$$\phi_1(n) = x(n) \tag{2.6}$$

Each subsequent basis function is created using one of the two following operation types.

A Type I operation applies a delay to an existing basis function:

$$\phi_r = q^{-m}\phi_i, \quad i < r \tag{2.7}$$

where q^{-m} denotes a delay of m samples.

A Type II operation forms a new basis function by multiplying three existing ones:

$$\phi_r = \phi_i\phi_j\phi_k^*, \quad i, j, k < r \tag{2.8}$$

where $(\cdot)^*$ denotes complex conjugation.

The output of the BAPS model is a weighted sum of all basis functions:

$$y(n) = \sum_{r=1}^{R} \theta_r\phi_r(n) \tag{2.9}$$

where θ_r are the model coefficients and R is the total number of basis functions [5].

2.3.2 Greedy Search

The BAPS basis-function configuration is selected offline using a greedy search algorithm [5]. The search starts with the input signal as the only basis function. At each step, candidate Type I and Type II basis functions are formed from the basis functions that have already been selected. The model error is evaluated for each candidate. The candidate that gives the largest reduction in model error is selected as the next basis function. This process continues until the desired number of basis functions, R, is reached.

The result of the greedy search is a BAPS basis-function configuration. It defines the selected basis functions and the operation used to generate each one. For a Type I basis function, it defines the source basis function and delay. For a Type II basis function, it defines the indices i, j, and k in $\phi_r = \phi_i\phi_j\phi_k^*$. The search requires

significant computation because many candidates are evaluated at each step. However, it only needs to be performed once for a given PA and signal condition. In the RTL implementation, the selected BAPS basis-function configuration is stored as a fixed operation table. The hardware then constructs the basis functions sequentially from this table, without online search or adaptation.

The greedy search reduces the model error step by step, but it does not guarantee the best possible global solution. In practice, however, the selected sequences have been shown to provide good linearisation performance for different PA types [5].

A consequence of this sequential construction is that some basis functions, especially those created later in the sequence, are the result of several chained Type II multiplications. The magnitude of these intermediate values depends on both the input signal level and the depth of the chain. When R is large, the chains become longer and the dynamic range demands on the floating-point format increase. If the exponent width is too small, values deep in the chain can underflow to zero. This causes a loss of information before the final output is computed. This sensitivity to dynamic range is an important challenge for reduced-precision hardware implementation as a main focus of this thesis.

2.4 Floating-Point Representation

Floating-point numbers are used in hardware and software to represent a wide range of values with a compact bit format. This section first introduces the IEEE 754 standard and then discusses custom precision formats for reduced-cost hardware implementation.

2.4.1 IEEE 754 Standard

The IEEE 754 standard defines a floating-point number using three fields: a sign bit S, a w-bit exponent E, and a t-bit mantissa M [19]. The value of a normalised number is

$$V = (-1)^S \cdot (1 + M) \cdot 2^{E-\text{Bias}}, \tag{2.10}$$

where $\text{Bias} = 2^{w-1} - 1$. The mantissa M stores the fractional part of the significand, and a leading 1 is assumed but not stored. Single precision uses $w = 8$ and $t = 23$, giving a total of 32 bits. Figure 2.3 shows the bit field layout of this format.

Figure 2.3: Bit field layout of the IEEE 754 single-precision floating-point format.

The exponent field determines the dynamic range of the format, that is, the largest and smallest values that can be represented. The mantissa field determines the

precision, that is, how accurately a value is stored. These two roles are largely independent of each other.

For normalised numbers, the exponent field is not zero and the leading bit of the significand is assumed to be 1. However, IEEE 754 also defines subnormal numbers for values very close to zero. A subnormal number is used when the exponent field is zero. In this case, the hidden leading bit is 0 rather than 1, and the significand becomes purely fractional. Its value can be written as

$$V_{\text{sub}} = (-1)^S \cdot (0.M) \cdot 2^{1-\text{Bias}}. \tag{2.11}$$

Subnormal numbers extend the representable range below the smallest normalised value. They provide a gradual transition towards zero, instead of rounding all values below the normalised range directly to zero.

The smallest positive normalised value is

$$V_{\text{min,normal}} = 2^{1-\text{Bias}}. \tag{2.12}$$

If subnormal numbers are supported and the mantissa has t bits, the smallest positive subnormal value is approximately

$$V_{\text{min,sub}} = 2^{1-\text{Bias}-t}. \tag{2.13}$$

Therefore, subnormal numbers extend the lower end of the dynamic range, but they also require additional hardware support for handling very small values.

2.4.2 Custom Precision Formats

In hardware design, floating-point formats with fewer bits than IEEE 754 single precision can be used. Reducing the total bit width lowers the area and power cost of each arithmetic operation. The key question is how much the bit width can be reduced before performance starts to degrade.

Reducing the mantissa width introduces rounding errors at each arithmetic step. In a system with many sequential operations, these errors can accumulate. Reducing the exponent width limits the dynamic range by increasing the lower bound of representable normalised values. As a result, small intermediate values are more likely to fall below the representable range of the chosen format. This is especially important for BAPS, where Type II operations multiply several basis functions together and can produce very small intermediate products.

If a value falls below the representable lower bound of the chosen format, underflow occurs. If a value becomes too large to be represented, overflow occurs. Both cases cause a loss of numerical information. In reduced-precision hardware, supporting very small values through subnormal numbers can increase the complexity of the arithmetic units. Therefore, the treatment of underflow is an important design choice when custom floating-point formats are used.

2.5 Floating-Point IP Cores

Implementing floating-point arithmetic directly in hardware requires complex logic for handling rounding, normalisation, and special cases. Rather than designing this logic from scratch, hardware designers typically use floating-point IP cores. These are pre-designed and pre-verified hardware modules that implement floating-point operations such as addition, subtraction, and multiplication for a given precision configuration.

IP cores are provided by tool vendors and can be integrated into a design with a standard interface. Their precision is often configurable through parameters, which makes it possible to try different bit widths without changing the surrounding logic. This is especially useful for precision exploration studies like this one.

Many floating-point IP cores also allow the designer to relax full IEEE 754 compliance, which can reduce hardware area and shorten the critical path. Some IP cores are implemented as purely combinational modules, where outputs are produced through combinational logic rather than stored in internal pipeline registers. This simplifies the surrounding control logic, since no additional pipeline-valid signals are required. However, it also places the full floating-point operation delay in the same timing path, which must be checked against the target clock constraint during logic synthesis. Other IP cores use internal pipeline registers to improve throughput, but require the surrounding logic to account for the fixed output latency.

3

Method

This chapter describes the approach used in this thesis to evaluate floating-point precision requirements for BAPS-based DPD. The work follows a three-phase workflow: system simulation, RTL implementation, and experimental validation on real PA hardware. The methodology is designed to connect numerical precision, DPD performance, and hardware complexity in a consistent way.

3.1 Experimental Workflow

In the first phase, a Python-based framework is used to generate BAPS models, evaluate custom floating-point arithmetic, and prepare predistorted signals for measurement and RTL verification. Custom floating-point formats with different mantissa and exponent widths are used in the BAPS basis-function computation. NMSE is recorded for each configuration as the main performance metric. This phase covers the (R, t) precision sweep, the analysis of dynamic range behaviour at low exponent widths, and the evaluation of the proposed power-of-two scaling method.

In the second phase, the BAPS DPD pipeline is implemented in Verilog with parameterisable floating-point precision. The RTL implementation is verified by comparing its output with the Python reference model using the same input signals. This confirms that the hardware implementation is consistent with the software reference model.

In the third phase, the generated predistorted signals are evaluated on the RF WebLab platform for the main precision sweeps and selected RTL validation cases. A baseband input signal is sent to the platform, transmitted through the real PA, and the measured output is returned for analysis. NMSE and ACPR are then computed and compared with the simulation results to assess whether the same performance trends are observed on the real PA. A gate-voltage robustness test is also performed to evaluate whether the selected reduced-precision configuration remains effective when the PA operating point changes.

3.2 Tool Selection

The Python simulation framework uses the APyTypes library [20] to simulate custom floating-point arithmetic. APyTypes allows arbitrary mantissa and exponent

widths to be specified, which makes it straightforward to test any precision configurations. The BAPS model and greedy search are implemented in Python, following the operation structure described in [5].

The RTL implementation is written in Verilog and uses Cadence ChipWare [21] floating-point IP cores for the arithmetic operations. These cores are not configured for full IEEE 754 compliance, which reduces hardware area. Cadence ChipWare provides configurable floating-point multipliers and adders with selectable mantissa and exponent widths, which allows the same RTL design to be reused for different precision settings. The ChipWare adder and subtractor support exponent widths down to $w = 5$ bits, while the ChipWare multiplier supports smaller exponent widths. For the configurations from $w = 5$ to $w = 8$, all arithmetic units therefore use ChipWare components. At $w = 4$, only the adder and subtractor fall below their supported range. In this case, the floating-point adder and subtractor are replaced by operators generated using FloPoCo [22], an open-source tool for producing custom floating-point arithmetic units, while the multiplier remains a ChipWare component. Functional simulation is carried out using Cadence Xcelium, and logic synthesis is performed using Cadence Genus. Test vectors are generated from the Python reference model and used as input to the RTL testbench. The resulting outputs are then compared numerically.

Experimental validation uses the RF WebLab remote measurement platform [17], which provides access to a real PA over the internet. Input signals are prepared in Python and sent to the platform through MATLAB. The platform transmits the signal through the PA and returns the measured output for analysis. This allows selected precision configurations to be verified on real hardware without requiring physical access to the measurement setup. NMSE and ACPR are used as the evaluation metrics. Figure 3.1 shows the physical setup of the platform.

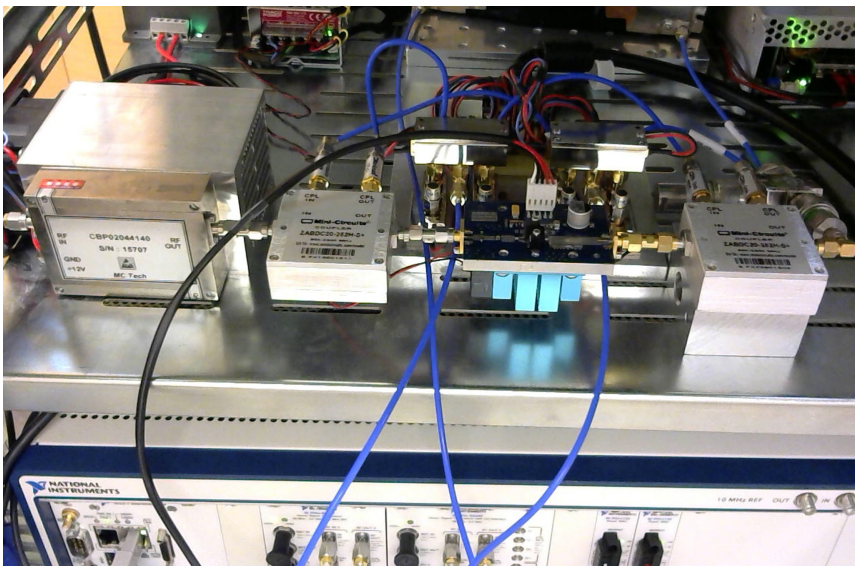


Figure 3.1: Physical setup of the RF WebLab platform, consisting of a National Instruments vector signal transceiver, RF couplers, and a power amplifier board [17].

3.3 Precision Exploration Strategy

This thesis extends the precision study in [16] in two directions. The first direction is a joint sweep of the number of basis functions R and the mantissa width t. The second direction is an investigation of low exponent widths and the proposed power-of-two scaling method.

For the joint sweep, R is varied from 4 to 12, and t is swept across 5, 6, 7, 8, 9, 10, and 23 bits. The exponent width is fixed at $w = 8$ during this sweep, so that only the effect of R and t on DPD performance is studied. The BAPS operation sequence is first determined by the offline greedy search at the maximum evaluated model size, $R = 12$. For smaller values of R, the first R basis functions of this sequence are used. This keeps the operation order consistent across the sweep and avoids introducing a new greedy-search result at every sweep point.

The DPD output is computed using the Python simulation framework with APyTypes [20] quantisation. The PA output NMSE is then measured using the RF WebLab platform [17] and recorded for each configuration. The results are arranged into a two-dimensional heatmap, where each cell shows the measured NMSE value. This makes it possible to identify the minimum R and t needed to reach the performance plateau, defined here as the region where further increases in R or t give no measurable improvement in NMSE.

For the exponent study, the exponent width is reduced from $w = 8$ to $w = 4$ across multiple mantissa widths. At each configuration, NMSE is recorded, and the amplitude distribution of each intermediate basis function ϕ_r is analysed. The minimum value of $|\phi_r(n)|$ across all time samples is compared with the normalised lower bound of the target floating-point format. This identifies which basis functions are at risk of underflow and at which exponent width the first failures occur. The findings from this analysis motivate the power-of-two scaling method described in Chapter 4.

3.3.1 PA Operating-Point Evaluation

A gate-voltage robustness test was performed to evaluate the effect of PA operating-point variation. The BAPS mem1 configuration was used for this test. For each floating-point configuration, the DPD coefficients were identified at the reference gate voltage $V_g = -2.30$ V. These coefficients were then kept fixed and used to generate predistorted signals for gate voltages from -2.10 V to -2.50 V.

No coefficient retraining or re-identification was performed at the other operating points. At each gate voltage, the predistorted signal was applied to the RF WebLab PA, and the measured output was used to calculate NMSE and ACPR. This procedure evaluates whether the DPD model and the reduced-precision floating-point configurations can maintain linearisation performance when the PA operating point changes.

3.4 Hardware Complexity Analysis

Hardware complexity is evaluated by synthesising the complete `baps_dpd` module for each (w, t) configuration under study. Logic synthesis is performed using Cadence Genus, targeting the ASAP7 predictive 7 nm FinFET PDK [23] with a clock period constraint of 5 ns (200 MHz). This constraint follows the synthesis setup used in prior work [16]. The synthesised area of each instance is recorded and normalised to the single-precision result ($w = 8, t = 23, \text{conv}$), where `conv` indicates that the power-of-two scaling method described in Section 4.3 is disabled. This expresses the area reduction relative to the full-precision baseline and makes it possible to relate hardware cost directly to the DPD performance results presented in Chapter 6.

The `type2_operator` module contains most of the floating-point arithmetic in the design and is therefore the most area-sensitive component. The remaining modules, including the FSM, shift registers, coefficient ROM, and accumulator, contribute a smaller and largely precision-independent part of the total area. Reporting the full system area gives a complete view of the hardware cost at each precision setting.

4

System Exploration and Conceptual Design

This chapter presents the software-based analysis carried out before the RTL implementation. It first describes the system simulation framework used for coefficient identification, precision evaluation, and RTL test-vector generation. It then analyses the dynamic range problem that may arise in Type II basis-function computation. Finally, it introduces a power-of-two scaling method to reduce underflow in narrow-exponent floating-point formats.

4.1 System Simulation Framework

The system simulation framework is implemented in Python and serves two main purposes. First, it identifies the BAPS DPD model coefficients and evaluates performance across different precision configurations. Second, it generates the input and reference data used for RTL verification. Figure 4.1 gives an overview of the framework.

The input signal is a bandpass Gaussian signal with 16,384 samples. It has an oversampling ratio of 5 and a PAPR of approximately 10.69 dB. Before transmission, the signal amplitude is scaled to 80% of the PA peak input limit. The signal is prepared in Python and sent to the RF WebLab platform through a MATLAB script. The MATLAB script handles the platform interface and returns the measured PA output. The sensitivity of the results to signal parameters such as PAPR and input scaling is discussed in Section 7.4.

Coefficient identification follows the indirect learning architecture (ILA) [6]. The PA is first measured without DPD, and the measured output is saved as a checkpoint. All later coefficient-identification and precision-evaluation steps load this checkpoint and run offline. This avoids changes in the PA state between different WebLab sessions and makes repeated experiments more consistent.

In the ILA approach used here, the measured PA output is treated as the input to a post-inverse model, and the original reference signal is treated as the desired output. Before model identification, the measured PA output $y_{\text{no-DPD}}$ is gain-aligned to produce

$$y_{\text{norm}} = G_{\text{PA}} \cdot y_{\text{no-DPD}}, \tag{4.1}$$

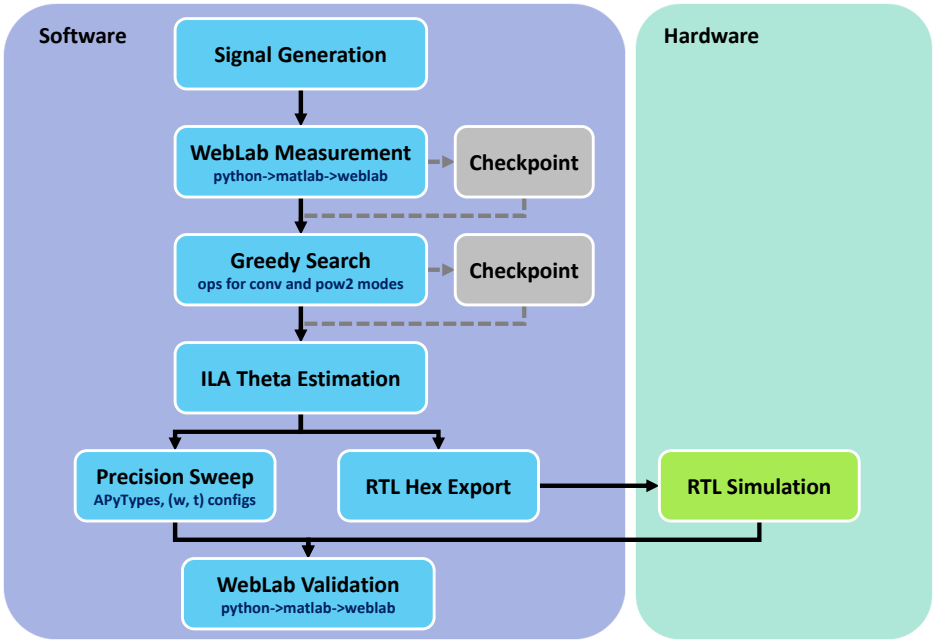


Figure 4.1: Overview of the system simulation and verification framework.

where G_{PA} is estimated from the cross-correlation between the input and output signals. The greedy search is then run using the pair $(y_{\mathrm{norm}}, x_{\mathrm{ref}})$ to determine the BAPS operation sequence.

The greedy search is performed up to a maximum of $R = 12$ basis functions. The resulting operation sequence is saved as a checkpoint. The greedy search result has a nested structure: for any $R' < R$, the first R' operations of the $R = 12$ sequence form a valid BAPS model with R' basis functions. Therefore, the (R, t) precision sweep does not require a separate greedy search for each value of R. It instead reuses the same $R = 12$ sequence and truncates it to the desired model size at each sweep point.

Based on the NMSE results from the software precision sweep, the performance plateau, defined here as the region where further increases in R or t give no measurable improvement in NMSE, is reached at around $R = 6$. The corresponding NMSE results are presented in Section 6.2. The RTL implementation therefore uses $R = 8$ basis functions (referred to as BAPS8) as a representative configuration within this plateau region. The operation sequence used in the RTL implementation is obtained from the greedy search performed in this thesis and is not copied from prior BAPS8 configurations. The greedy search is run on the mem1 configuration, which uses a memory depth of one sample. Table 4.1 shows the full operation sequence determined by the greedy search.

After the operation sequence is fixed, the model coefficients $\hat{\theta}$ are estimated by least squares. This coefficient-estimation step is performed separately for each scaling mode, because the basis-function magnitudes differ between modes. The scaling modes are introduced in Section 4.3.

Reduced-precision floating-point arithmetic is simulated using the APyTypes li-

Table 4.1: BAPS operation sequence for the mem1 configuration, determined by offline greedy search with $R = 12$.

r	Operation	Type
1	$\phi_1 = x(n)$	Init
2	$\phi_2 = \phi_1 \cdot \lvert\phi_1\rvert^2$	Type II
3	$\phi_3 = \phi_2 \cdot \lvert\phi_1\rvert^2$	Type II
4	$\phi_4 = q^{-1}\phi_1$	Type I
5	$\phi_5 = q^{-1}\phi_4$	Type I
6	$\phi_6 = q^{-1}\phi_5$	Type I
7	$\phi_7 = q^{-1}\phi_6$	Type I
8	$\phi_8 = \phi_4 \cdot \phi_2 \cdot \phi_4^*$	Type II
9	$\phi_9 = \phi_7 \cdot \phi_7 \cdot \phi_4^*$	Type II
10	$\phi_{10} = q^{-1}\phi_8$	Type I
11	$\phi_{11} = \phi_7 \cdot \lvert\phi_7\rvert^2$	Type II
12	$\phi_{12} = \phi_6 \cdot \phi_4 \cdot \phi_2^*$	Type II

brary [20]. In the high-level software sweep, quantisation is applied at the output of each Type II basis-function computation. The RTL implementation is stricter because the two multiplication stages inside the `type2_operator` (the hardware module that computes Type II operations, described in Chapter 5) are quantised separately, and each arithmetic unit produces a result in the target (w, t) format. A per-operator quantisation mode is also available in the software framework, but the output-level mode is used for the precision sweep because it runs faster across a large number of configurations. Therefore, the software sweep is used to identify the main precision trends, while RTL verification is used to confirm configurations that may be affected by intermediate-product underflow.

After coefficient estimation, the framework splits into two paths. The first path performs a precision sweep using APyTypes across a range of (w, t) configurations. The second path exports the input signals as hex files, which are used as test vectors for RTL simulation in Cadence Xcelium. The RTL simulation output is then returned to the software side for numerical comparison. Both paths feed into a final WebLab validation step, where selected configurations are verified on the real PA.

4.2 Underflow Analysis

When the exponent width is reduced, a numerical problem may arise in the Type II computation chain. This section explains the root cause of this problem using ϕ_2 as a concrete example.

4.2.1 Type II Underflow: Problem Analysis

As discussed in Section 2.4, reducing the exponent width reduces the dynamic range of a floating-point format and increases the risk of underflow. In the reduced-precision hardware configurations considered in this work, subnormal numbers are

not used. Supporting them would not resolve the underflow problem, since the minimum values of the Type II basis functions fall well below the subnormal range of narrow-exponent formats. Therefore, the smallest positive normalised value is used as the practical lower bound in the following analysis.

For a floating-point format with exponent width w, the bias is $\text{Bias} = 2^{w-1} - 1$, and the smallest positive normalised value is

$$V_{\text{min,normal}} = 2^{1-\text{Bias}}. \tag{4.2}$$

For $w = 5$, this gives $\text{Bias} = 15$ and a smallest normalised magnitude of

$$2^{-14} \approx 6.1 \times 10^{-5}. \tag{4.3}$$

Values below this lower bound are treated as underflowed and are represented as zero in both the hardware implementation and the APyTypes simulation.

The root cause lies in the sequential structure of BAPS. Each Type II operation multiplies three basis functions together, so the magnitude of the result scales roughly as the product of their magnitudes. To see this concretely, consider

$$\phi_2 = \phi_1 \cdot |\phi_1|^2. \tag{4.4}$$

The input signal $\phi_1 = x(n)$ has a root-mean-square (RMS) amplitude of approximately 1.8×10^{-2}. Since the magnitude of ϕ_2 is proportional to $|x|^3$, its RMS value is approximately 1.4×10^{-5}. The minimum value across all samples is around 1.1×10^{-13}.

The RMS value of ϕ_2 is already below the $w = 5$ normalised lower bound, and its minimum sample value is far below it. Therefore, many samples of ϕ_2 are at risk of underflow when normalised-only reduced-precision arithmetic is used. More importantly, the intermediate products inside the Type II operator can fall below this lower bound before the final basis function is formed. In hardware, these underflowed intermediate values are represented as zero, which can make the computed basis function collapse to zero for the affected samples.

In general, a Type II function with nonlinear order p has a magnitude proportional to $|x|^p$, which decreases rapidly as p increases. Table 4.2 shows the amplitude statistics for all basis functions in the `conv` scaling mode. The RMS values of all Type II basis functions are below the $w = 5$ normalised lower bound, although some peak samples remain above it. This indicates that these basis functions spend a significant part of their time close to or below the lower representable range of the target format. As a result, they are sensitive to underflow at $w = 5$.

Figures 4.2 and 4.3 illustrate the Type II computation inside the `type2_operator`. The operation is computed in two multiplication steps. In the first step, ϕ_i and ϕ_j are multiplied to produce an intermediate value. In the second step, this intermediate value is multiplied by ϕ_k^* to give the final result. Without pre-scaling, the intermediate value can fall below the normalised lower bound of a narrow-exponent format and be rounded to zero. Once this value becomes zero, the second multiplication also produces zero and the entire basis function is lost. Figure 4.3 shows

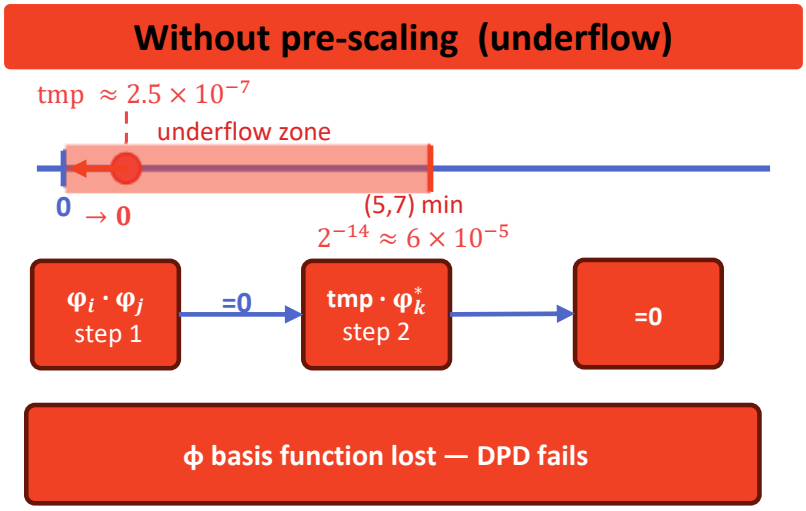


Figure 4.2: Underflow in the Type II operator without pre-scaling. The first intermediate product can underflow before the second multiplication.

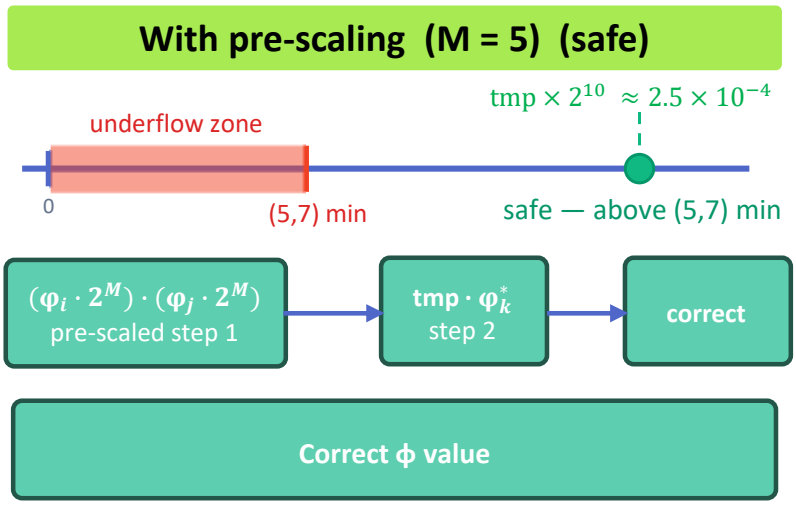


Figure 4.3: Pre-scaling in the Type II operator with $M = 5$. The first intermediate product is raised above the underflow threshold before the second multiplication.

Table 4.2: Amplitude statistics of all basis functions in the `conv` mode. The $w = 5$ normalised lower bound is $2^{-14} \approx 6.1 \times 10^{-5}$.

Basis	Type	RMS	Min $\lvert\phi_r\rvert$	Peak $\lvert\phi_r\rvert$
ϕ_1	Init	1.81×10^{-2}	4.86×10^{-5}	6.20×10^{-2}
ϕ_2	Type II	1.43×10^{-5}	1.15×10^{-13}	2.38×10^{-4}
ϕ_3	Type II	3.77×10^{-8}	2.72×10^{-22}	9.14×10^{-7}
ϕ_4	Type I	1.81×10^{-2}	4.86×10^{-5}	6.20×10^{-2}
ϕ_5	Type I	1.81×10^{-2}	4.86×10^{-5}	6.20×10^{-2}
ϕ_6	Type I	1.81×10^{-2}	4.86×10^{-5}	6.20×10^{-2}
ϕ_7	Type I	1.81×10^{-2}	4.86×10^{-5}	6.20×10^{-2}
ϕ_8	Type II	3.69×10^{-8}	2.43×10^{-18}	8.41×10^{-7}
ϕ_9	Type II	1.04×10^{-5}	1.36×10^{-11}	1.61×10^{-4}
ϕ_{10}	Type I	3.69×10^{-8}	2.43×10^{-18}	8.41×10^{-7}
ϕ_{11}	Type II	1.43×10^{-5}	1.15×10^{-13}	2.38×10^{-4}
ϕ_{12}	Type II	3.45×10^{-8}	6.66×10^{-18}	6.12×10^{-7}

how pre-scaling with $M = 5$ raises the intermediate product above the underflow threshold.

The problem is made worse by the dependency structure of the operation sequence. When ϕ_2 underflows to zero, any later basis function that uses it as an operand is also affected. From Table 4.1, the functions ϕ_3, ϕ_8, and ϕ_{12} all depend on ϕ_2. When ϕ_8 is lost, ϕ_{10} is also lost because it is a delayed copy of ϕ_8. A single underflow event can therefore silently remove multiple basis functions, causing a large and sudden drop in DPD performance.

Yu et al. [16] reported that $w = 5$ was sufficient for the BAPS configurations they studied. However, that work evaluated exponent widths only down to $w = 5$ and did not analyse the intermediate-product underflow that can occur inside Type II operations. The results in this thesis show that the safe exponent width depends on the signal scaling, PA operating point, selected BAPS basis-function configuration, and hardware treatment of very small values. A different PA can lead to a different selected BAPS basis-function configuration, while a different signal or operating point can change the magnitude range of the Type II intermediate products. For the input signal used here, even ϕ_1 has samples that fall below the $w = 5$ normalised lower bound, with a minimum value of 4.86×10^{-5}, which is below the bound of 6.1×10^{-5}. The Type II basis functions derived from it are then much more likely to fall below this bound. This motivates the power-of-two scaling method described in the following section.

4.3 Power-of-Two Scaling Method

The underflow problem requires a solution that locally raises the magnitude of Type II basis functions into the representable range of the target format. A natural first approach is RMS normalisation, where each basis function is divided by its RMS amplitude after it is computed. This keeps all values near unity and avoids underflow. However, RMS normalisation requires computing the RMS over the full input

signal before processing begins. This is not compatible with a streaming hardware design, where samples are processed one at a time. A fixed normalisation factor from the training signal may also become inaccurate if the input signal statistics change. For these reasons, RMS normalisation is not suitable for this implementation.

Instead, this thesis proposes a power-of-two scaling method. For clarity, two modes are distinguished throughout the remainder of this chapter: `conv`, the conventional mode without power-of-two pre-scaling or post-scaling compensation, and `pow2`, the proposed mode with both operations enabled. The detailed `pow2` computation is described below.

For each Type II basis function ϕ_r, a fixed integer scaling exponent k_r is computed offline from the training signal. The exponent is selected from the RMS amplitude of the corresponding Type II output. The aim is to move the typical magnitude of each Type II basis function to an order-one range while keeping the scaling factor as a power of two:

$$k_r \approx -\log_2\Big(\mathrm{RMS}(|\phi_r|)\Big). \tag{4.5}$$

The final value of k_r is chosen as an integer shift applied to the exponent field, with a small upward adjustment when needed to ensure the scaled RMS remains comfortably within the representable range. Since k_r depends only on the operation sequence and the signal statistics, both of which are fixed after the offline greedy search, it does not need to be recomputed at runtime. Unlike exact RMS normalisation, the power-of-two scaling only needs to place each value in the correct order of magnitude, so it is much less sensitive to changes in the input signal statistics. In practice, the selected k_r values remain stable, and a moderate change in signal level changes each k_r by at most one integer value.

The scaling is applied sequentially. If a later Type II operation uses an earlier scaled basis function as an operand, the later operation is based on that scaled value. Therefore, the "RMS before" column in Table 4.3 refers to the RMS of the raw Type II output before applying the final 2^{k_r} output scaling for that operation. It is not always the same as the RMS value in the `conv` scaling mode. Table 4.3 lists the scaling exponents used for each Type II basis function.

Table 4.3: Power-of-two scaling exponents k_r for each Type II basis function, and the resulting RMS after scaling. All values are computed offline from the training signal in the Python framework.

Basis	Operation	k_r	RMS before	RMS after
ϕ_2	$\phi_1 \cdot \lvert\phi_1\rvert^2$	+16	1.43×10^{-5}	9.39×10^{-1}
ϕ_3	$\phi_2 \cdot \lvert\phi_1\rvert^2$	+11	1.35×10^{-3}	2.76×10^{0}
ϕ_8	$\phi_4 \cdot \phi_2 \cdot \phi_4^*$	+11	1.25×10^{-3}	2.56×10^{0}
ϕ_9	$\phi_7 \cdot \phi_7 \cdot \phi_4^*$	+17	1.04×10^{-5}	1.36×10^{0}
ϕ_{11}	$\phi_7 \cdot \lvert\phi_7\rvert^2$	+16	1.43×10^{-5}	9.39×10^{-1}
ϕ_{12}	$\phi_6 \cdot \phi_4 \cdot \phi_2^*$	+11	8.97×10^{-4}	1.84×10^{0}

Applying 2^{k_r} to a floating-point number is equivalent to adding k_r to its exponent field, which is a simple integer addition. No mantissa multiplication is needed. This

makes the scaling method low-cost to implement in hardware, as described in the hardware design chapter.

4.3.1 Pre-scaling inside the Type II Operator

The output scaling described above raises the magnitude of the completed basis function. However, it does not by itself protect the intermediate product inside the `type2_operator`. To prevent underflow inside this operator, the first two operands ϕ_i and ϕ_j are multiplied by a fixed factor 2^M before the first multiplication, where $M = 5$. The first multiplication therefore computes

$$(2^M\phi_i)(2^M\phi_j) = 2^{2M}\phi_i\phi_j. \tag{4.6}$$

This increases the first intermediate product by a factor of 2^{2M}. For $M = 5$, the increase is 2^{10}. Figure 4.4 shows the complete Type II computation flow.

This pre-scaling is needed because the first intermediate product $\phi_i\phi_j$ can be much smaller than either operand individually. The most vulnerable cases occur when at least one operand is already a nonlinear basis function with a small magnitude. In those cases, the first intermediate product can fall below the normalised lower bound of a narrow-exponent format before the second multiplication is applied. Once this intermediate value becomes zero, the final Type II output is also zero.

The value $M = 5$ is selected after checking the operand amplitude range in the target BAPS sequence. It provides sufficient margin for the $w = 5$ configurations studied in this thesis while avoiding excessive growth of the scaled operands. It is important to note that the product of two peak input samples is not the limiting case. The input peak is approximately 0.062, close to 2^{-4}, so the product of two such samples is around 2^{-8}, which is well above the $w = 5$ lower bound of 2^{-14}. The real underflow risk comes from products involving small Type II basis values. Pre-scaling by $M = 5$ shifts these vulnerable intermediate products upward by a factor of 2^{10} and prevents the main Type II computation from collapsing to zero.

In the present implementation, the Type II operator computes $(\phi_i\phi_j)\phi_k^*$. Therefore, ϕ_i and ϕ_j are pre-scaled before the first multiplication, while ϕ_k^* enters the second multiplication without pre-scaling. In exact arithmetic, the three operands can be multiplied in a different order. In reduced-precision hardware, however, the chosen order can affect the magnitude of the first intermediate product and its underflow risk. The effect of alternative multiplication orders is not discussed in this thesis.

4.3.2 Post-Scaling Compensation

After the Type II operator produces its result, a compensation step is applied. Since both operands ϕ_i and ϕ_j are pre-scaled by 2^M, the raw Type II output contains an extra factor of 2^{2M}:

$$\tilde{\phi}_r = (2^M\phi_i)(2^M\phi_j)\phi_k^* = 2^{2M}\phi_i\phi_j\phi_k^*. \tag{4.7}$$

The intended scaled basis function is

$$\phi_{r,\text{scaled}} = 2^{k_r}\phi_i\phi_j\phi_k^*, \tag{4.8}$$

where k_r is the offline scaling exponent for the current Type II basis function. Therefore, the post-scaling compensation applied after the Type II operator is

$$k_{\text{eff},r} = k_r - 2M. \tag{4.9}$$

This compensation is implemented by adding $k_{\text{eff},r}$ to the exponent field of the Type II output. Since this is also a power-of-two scaling operation, no mantissa multiplication is required.

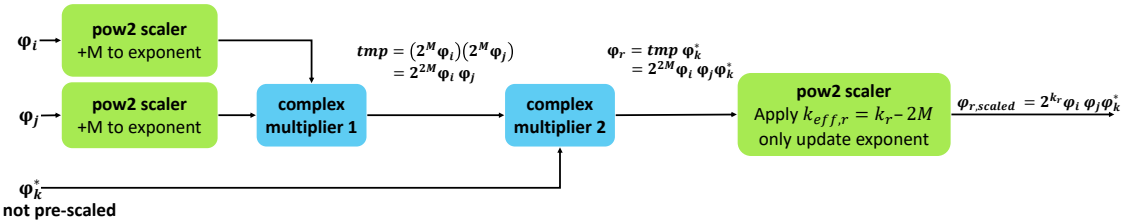


Figure 4.4: Computation flow of a Type II operation with power-of-two pre-scaling and post-scaling compensation. The operands ϕ_i and ϕ_j are pre-scaled by 2^M, while ϕ_k^* is not pre-scaled. The final exponent update applies $k_{\text{eff},r} = k_r - 2M$.

Table 4.4 shows the amplitude statistics after `pow2` scaling is applied. The RMS values of the Type II basis functions are raised close to unity, which places their typical magnitudes well above the $w = 5$ normalised lower bound. Although some individual samples may still be very small, the scaling method prevents the main Type II intermediate products from collapsing to zero during hardware computation. The peak values after scaling remain well below the upper representable limit of the $w = 5$ formats used in this work, so the scaling does not introduce an overflow problem for this dataset.

Table 4.4: Amplitude statistics of all basis functions in the `pow2` scaling mode.

Basis	Type	RMS	Min $\lvert\phi_r\rvert$	Peak $\lvert\phi_r\rvert$
ϕ_1	Init	1.81×10^{-2}	4.86×10^{-5}	6.20×10^{-2}
ϕ_2	Type II	9.39×10^{-1}	7.53×10^{-9}	1.56×10^{1}
ϕ_3	Type II	2.76×10^{0}	3.65×10^{-14}	1.23×10^{2}
ϕ_4	Type I	1.81×10^{-2}	4.86×10^{-5}	6.20×10^{-2}
ϕ_5	Type I	1.81×10^{-2}	4.86×10^{-5}	6.20×10^{-2}
ϕ_6	Type I	1.81×10^{-2}	4.86×10^{-5}	6.20×10^{-2}
ϕ_7	Type I	1.81×10^{-2}	4.86×10^{-5}	6.20×10^{-2}
ϕ_8	Type II	2.56×10^{0}	3.26×10^{-10}	1.13×10^{2}
ϕ_9	Type II	1.36×10^{0}	1.78×10^{-6}	2.11×10^{1}
ϕ_{10}	Type I	2.56×10^{0}	3.26×10^{-10}	1.13×10^{2}
ϕ_{11}	Type II	9.39×10^{-1}	7.53×10^{-9}	1.56×10^{1}
ϕ_{12}	Type II	1.84×10^{0}	8.94×10^{-10}	8.22×10^{1}

4.3.3 Underflow Reduction Across Precision Configurations

The amplitude statistics above describe typical magnitudes, but they do not directly show how much basis information is lost in hardware. To make this concrete, the

number of basis-function samples that are flushed to zero is counted for four precision configurations. Each configuration is evaluated against the normalised lower bound of its own floating-point format. For $w = 8$ this bound is 2^{-126}, for $w = 6$ it is $2^{-30} \approx 9.3 \times 10^{-10}$, and for $w = 5$ it is $2^{-14} \approx 6.1 \times 10^{-5}$. The counts are aggregated over the input signals from all gate-voltage operating points evaluated in Chapter 6, so they describe the average underflow behaviour across the full operating range rather than a single point.

The following analysis evaluates two scaling modes: `conv`, in which no scaling is applied, and `pow2`, in which power-of-two pre-scaling is enabled. Table 4.5 reports the results. The $(w, t) = (8, 7)$ configuration loses no samples, because its exponent range is wide enough to represent even the smallest Type II values. The $(w, t) = (6, 7)$ `conv` configuration loses about 9.9% of its samples to flush-to-zero. This loss is consistent with the small performance gap observed for this configuration in the robustness results of Chapter 6. The $(w, t) = (5, 7)$ `conv` configuration is much worse and loses about 26.5% of its samples, which confirms that $w = 5$ cannot be used without scaling.

The most important result is the $(w, t) = (5, 7)$ `pow2` configuration. With pre-scaling added in the RTL, only 0.4% of the samples are flushed to zero. This value is lower than the 9.9% of the $(w, t) = (6, 7)$ `conv` configuration, even though the $w = 5$ lower bound is about five orders of magnitude higher than the $w = 6$ lower bound. This means that a fixed power-of-two shift keeps more usable basis information than a wider exponent field does on its own. It is the direct numerical reason why the proposed $(5, 7)$ `pow2` configuration matches the high-precision references in the PA measurements of Chapter 6.

Table 4.5: Fraction of basis-function samples flushed to zero, for four precision configurations. Each configuration is evaluated against the normalised lower bound of its own floating-point format. A sample flushed to zero contributes nothing to the basis function. The counts are aggregated over the input signals from all operating points evaluated in Chapter 6.

Configuration	Mode	Samples flushed to zero
$(w, t) = (8, 7)$	`conv`	0.0%
$(w, t) = (6, 7)$	`conv`	9.9%
$(w, t) = (5, 7)$	`conv`	26.5%
$(w, t) = (5, 7)$	`pow2`	0.4%

4.3.4 Lower Bound: Why Pow2 Scaling Cannot Recover $w = 4$

Pow2 scaling successfully lifts the typical magnitudes of the Type II basis functions into the representable range of $w = 5$ formats. However, it does not remove the exponent-width constraint completely. After the basis functions are scaled, the final DPD output still depends on whether the corresponding model coefficients can be represented in the selected floating-point format.

In this thesis, the coefficients are estimated directly for the scaled basis set using least squares. Therefore, the stored coefficient values already reflect the effect of the basis-function scaling. These stored coefficients are the values that must be represented by the reduced-precision hardware.

For $w = 5$, the normalised lower bound is $2^{-14} \approx 6.1 \times 10^{-5}$. Eleven of the twelve stored coefficients remain above this bound. The one exception is ϕ_9, whose coefficient of 4.97×10^{-5} is slightly below the bound. However, the RTL results show that the overall DPD performance is still recovered at $w = 5$, which indicates that the loss of this single term has only a limited effect. For $w = 4$, the normalised lower bound becomes $2^{-6} \approx 1.56 \times 10^{-2}$, which is much larger. In this case, seven coefficients fall below the lower bound and are rounded to zero. Six of these seven coefficients are representable at $w = 5$, while ϕ_9 is already marginal at $w = 5$.

Table 4.6 compares the stored coefficient magnitudes after `pow2` scaling with the normalised lower bound of the $(w, t) = (4, 10)$ format. Seven of the twelve coefficients fall below this bound and are represented as zero in the hardware-level analysis. These include all six Type II basis functions and ϕ_{10}, which is a delayed copy of the scaled ϕ_8 basis function. A zero coefficient removes the corresponding term from the output sum, so those basis functions make no contribution to the DPD output even if the basis functions themselves are computed correctly.

Table 4.6: Stored coefficient magnitudes after `pow2` scaling, compared with the normalised lower bound of the $(w, t) = (4, 10)$ format, $2^{-6} \approx 1.56 \times 10^{-2}$. The coefficient of ϕ_9 is also slightly below the $w = 5$ normalised lower bound, but this single marginal term has limited effect on the final DPD performance.

Basis	Type	k_r	Stored coefficient magnitude	Underflow at $w = 4$?
ϕ_1	Init	+0	8.50×10^{-1}	no
ϕ_2	Type II	+16	6.27×10^{-4}	**yes**
ϕ_3	Type II	+11	1.84×10^{-3}	**yes**
ϕ_4	Type I	+0	7.05×10^{-2}	no
ϕ_5	Type I	+0	6.54×10^{-2}	no
ϕ_6	Type I	+0	2.62×10^{-2}	no
ϕ_7	Type I	+0	8.40×10^{-2}	no
ϕ_8	Type II	+11	8.80×10^{-4}	**yes**
ϕ_9	Type II	+17	4.97×10^{-5}	**yes**
ϕ_{10}	Type I	+0	2.87×10^{-4}	**yes**
ϕ_{11}	Type II	+16	1.58×10^{-4}	**yes**
ϕ_{12}	Type II	+11	8.96×10^{-5}	**yes**

The underlying cause is that pow2 scaling resolves the underflow problem in the basis-function computation, but the low input signal amplitude requires large scaling exponents k_r to lift the Type II basis functions into the representable range. These large k_r values result in small stored coefficients for the Type II terms. At $w = 5$, all but one of these coefficients remain representable. At $w = 4$, the normalised lower bound is too large to accommodate them. Therefore, $w = 5$ is identified as the practical lower bound on exponent width for this signal and PA operating point. This conclusion is confirmed by the RTL simulation results presented in Chapter 6.

5

Hardware Design

This chapter describes the RTL hardware design of the BAPS DPD pipeline. It translates the fixed BAPS operation sequence and the power-of-two scaling method from Chapter 4 into a parameterised Verilog implementation. The chapter first presents the top-level architecture. It then explains the finite state machines used to control the two computation stages. Finally, it describes the hardware implementation of the power-of-two scaler and the Type II operator.

5.1 RTL Architecture Overview

The `baps_dpd` module implements the forward computation path of the BAPS DPD model. Coefficient identification is performed offline in Python and is not part of the RTL implementation. It takes one complex input sample $x(n)$ and produces the predistorted output sample $y(n)$. The operation sequence is hardcoded in the FSM operation table, following the offline greedy-search result shown in Table 4.1. The design is parameterised by the exponent width w and the mantissa width t, so the same RTL structure can be reused for different floating-point precision configurations.

The top-level design consists of two computation stages. Stage 1 is the basis builder, which computes the basis functions ϕ_1 to ϕ_R sequentially and stores them in a register bank. Stage 2 performs coefficient multiplication and accumulation to produce the final DPD output. Figure 5.1 shows the block diagram of the RTL design.

For a Type I operation, the result is a delayed copy of an existing basis function. This operation reads the required value from the ϕ register bank and does not require floating-point arithmetic. For a Type II operation, the result is computed as

$$\phi_r = \phi_i \phi_j \phi_k^*, \tag{5.1}$$

where $(\cdot)^*$ denotes complex conjugation. This operation is implemented by the `type2_operator` module described in Section 5.4. The Type II operator contains the main floating-point arithmetic in the basis builder and is therefore the most precision-sensitive part of the design.

After all R basis functions have been computed, Stage 2 starts the final weighted sum. A single coefficient-multiplier path is reused sequentially. For each r from 1 to R, the design computes $\theta_r \phi_r(n)$ and adds the result to an accumulator:

$$y(n) = \sum_{r=1}^{R} \theta_r \phi_r(n). \tag{5.2}$$

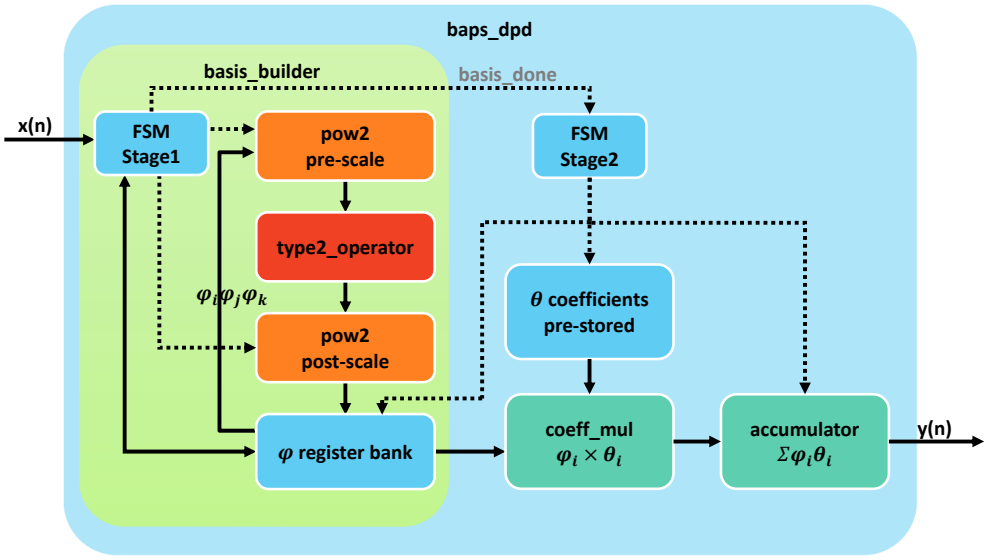


Figure 5.1: Block diagram of the `baps_dpd` RTL design.

The coefficients are estimated offline using the ILA framework described in Section 4.1. They are then stored as constants in the RTL design before synthesis. Separate coefficient sets are generated for the `conv` and `pow2` scaling modes because the basis-function magnitudes are different in the two cases.

This architecture is intended for precision exploration rather than maximum throughput. The basis functions and coefficient products are computed sequentially, which reduces hardware duplication. The trade-off is that one output sample requires several FSM cycles to complete.

5.2 Finite State Machine Design

The pipeline is controlled by two finite state machines (FSMs). The Stage 1 FSM controls basis-function construction. The Stage 2 FSM controls coefficient multiplication and accumulation. Stage 2 starts only after Stage 1 has completed all basis functions for the current input sample.

5.2.1 Stage 1 FSM

The Stage 1 FSM has seven states. Figure 5.2 shows the state transition diagram. The FSM starts in the `idle` state and waits for a start signal. When a new input sample arrives, it moves to the `init` state, where ϕ_1 is set to $x(n)$ and the basis-function counter is reset.

The FSM then enters a loop for $r = 2, \ldots, R$. In the `load_op` state, it reads the operation type and operand indices for the current basis function from the hardcoded operation table. In the `exec` state, it executes the selected operation. For a Type I operation, the FSM reads the required basis value from the previous-sample

basis register array. In the current BAPS8-mem1 implementation, this provides a one-sample delay, $q^{-1}\phi_i$. For a Type II operation, the FSM applies the required pre-scaling to the selected operands and sends them to the `type2_operator`. The resulting value is then passed to the next state.

The `scale` state applies the power-of-two post-scaling for Type II operations. This is done by adding the effective exponent shift k_{eff} to the exponent field of the Type II result. Type I results bypass this scaling step because they are delayed copies of already computed basis functions. In the `store` state, the result is written back to the basis-function register bank. When all R basis functions have been computed, the FSM moves to the `done` state, sends a completion signal to Stage 2, and then returns to the `idle` state.

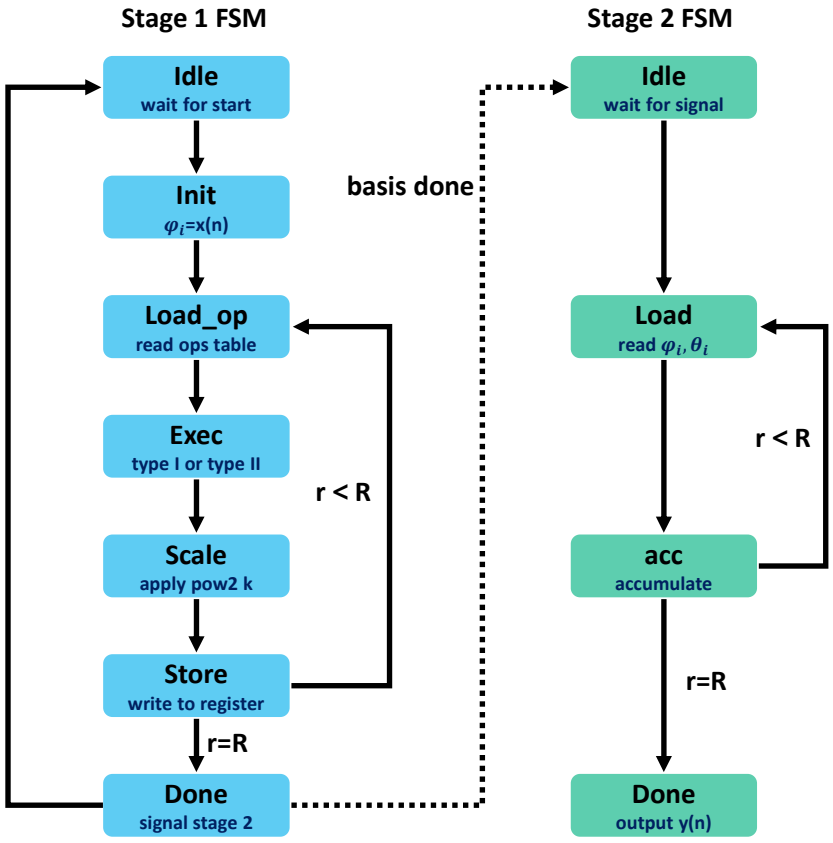


Figure 5.2: State transition diagrams of the Stage 1 and Stage 2 FSMs.

5.2.2 Stage 2 FSM

The Stage 2 FSM has four states, also shown in Figure 5.2. It starts in the `idle` state and waits for the completion signal from Stage 1. When this signal arrives, the FSM moves to the `load` state, where the current basis function ϕ_r and coefficient θ_r are selected.

The coefficient multiplier is implemented as a combinational floating-point path. Its output is then added to the accumulator in the `acc` state. This loop repeats

for $r = 1, \ldots, R$. When all coefficient products have been accumulated, the FSM enters the `done` state. The final value $y(n)$ is made available at the output, the accumulator is reset, and the FSM returns to the `idle` state.

5.3 Power-of-Two Scaler Implementation

The power-of-two scaling method described in Section 4.3 is implemented using exponent-field updates. Pre-scaling and post-scaling use the same exponent-shift principle, but they are implemented separately in the current RTL.

For pre-scaling, a local combinational function adds the fixed shift M to the exponent fields of the selected Type II operands. The function is applied separately to the real and imaginary parts of ϕ_i and ϕ_j. Zero inputs remain zero.

For post-scaling, the `pow2_scaler` module extracts the sign, exponent, and mantissa fields of the Type II result, adds the signed scaling value, and reassembles the floating-point word. The sign and mantissa fields remain unchanged. Therefore, both operations use exponent updates only and do not require an additional floating-point multiplier.

In the Type II computation path, pre-scaling is applied before the first complex multiplication. The operands ϕ_i and ϕ_j are both scaled by 2^M, where $M = 5$ in this implementation. This raises the typical magnitude of the vulnerable intermediate product $\phi_i\phi_j$ and reduces the risk of underflow inside the Type II operator, as discussed in Section 4.2.

After the Type II operator produces its output, post-scaling compensation is applied in the `scale` state of the Stage 1 FSM. The effective shift is

$$k_{\text{eff},r} = k_r - 2M, \tag{5.3}$$

where k_r is the target scaling exponent for the current Type II basis function. The term $2M$ compensates for the two pre-scaled operands used in the first multiplication step. Since k_r and M are fixed constants for a given operation sequence, the hardware implementation only needs fixed signed exponent additions. This keeps the scaling overhead small compared with the floating-point multipliers and adders in the Type II operator.

5.4 Type II Operator Module

The `type2_operator` module implements the Type II operation

$$\phi_r = \phi_i\phi_j\phi_k^*. \tag{5.4}$$

The module is parameterised by `exp_width` and `sig_width`. In the notation used in this thesis, `exp_width` corresponds to the exponent width w, while `sig_width` controls the significand precision associated with the mantissa width t. This parameterisation allows the same module interface to be used across the precision configurations studied in this work.

The computation is organised into two combinational complex-multiplication stages. First, ϕ_i and ϕ_j are multiplied to form an intermediate value

$$\text{tmp} = \phi_i \phi_j. \tag{5.5}$$

The real and imaginary parts are

$$\text{tmp}_{\text{re}} = \phi_{i,\text{re}}\phi_{j,\text{re}} - \phi_{i,\text{im}}\phi_{j,\text{im}}, \tag{5.6}$$
$$\text{tmp}_{\text{im}} = \phi_{i,\text{re}}\phi_{j,\text{im}} + \phi_{i,\text{im}}\phi_{j,\text{re}}. \tag{5.7}$$

Second, the intermediate value is multiplied by ϕ_k^*. Since complex conjugation changes the sign of the imaginary part of ϕ_k, the output components can be written as

$$\phi_{r,\text{re}} = \text{tmp}_{\text{re}}\phi_{k,\text{re}} + \text{tmp}_{\text{im}}\phi_{k,\text{im}}, \tag{5.8}$$
$$\phi_{r,\text{im}} = \text{tmp}_{\text{im}}\phi_{k,\text{re}} - \text{tmp}_{\text{re}}\phi_{k,\text{im}}. \tag{5.9}$$

No separate sign-inversion block is required for the conjugate because the sign change is included in these arithmetic expressions.

The full Type II operation requires eight real floating-point multiplications, two real additions, and two real subtractions. For the configurations supported by Cadence ChipWare, these operations use configurable ChipWare floating-point multiplier, adder, and subtractor components [21]. For the $w = 4$ adder/subtractor case, the FloPoCo-generated operator described in Section 3.2 is used where the ChipWare component does not support the required exponent width. The surrounding `type2_operator` interface remains unchanged.

The module is implemented without internal pipeline registers. Therefore, the output is produced through a combinational arithmetic path. This keeps the control logic simple, but it also means that the delay of the Type II operator contributes directly to the timing path. This is one reason why timing-constrained synthesis is used in the hardware complexity analysis.

The intermediate values tmp_{re} and tmp_{im} are produced by floating-point arithmetic before the second multiplication level. They are rounded to the target precision at this point. This introduces an additional rounding event inside each Type II operation. It is also the location where underflow can occur without pre-scaling, which motivates the power-of-two scaling method introduced in Section 4.3 and implemented in this chapter.

6

Results

This chapter presents the verification, measurement, and synthesis results. It first verifies the consistency between the Python reference model and the RTL implementation, then presents the joint (R, t) precision sweep, the exponent-width sweep and the evaluation of the power-of-two scaling method, and finally the PA operating-point robustness test and the hardware complexity results.

All NMSE and ACPR results come from separate WebLab measurement sessions, and the no-DPD baseline is re-measured in each session, so the baseline values differ by a small amount between tables and figures. These differences are within the measurement repeatability of about 0.12 dB reported in Section 6.4, and they do not affect the comparison between configurations.

6.1 Python–RTL Consistency Verification

Before the precision exploration is started, the RTL implementation was verified against the Python reference model at two configurations, $(w, t) = (8, 23)$ and $(w, t) = (8, 7)$. For each, the same input signal and coefficient set were applied to both the Python model and the RTL simulation, and the predistorted signals from both paths were evaluated on the RF WebLab platform [17]. The resulting PA output NMSE and ACPR were used to check whether the RTL implementation preserves the behaviour of the Python reference.

Figures 6.1 and 6.2 show the predistorted signal spectrum and PA output spectrum for the two configurations. In both cases, the Python and RTL curves are nearly indistinguishable, indicating that the RTL datapath follows the reference model closely.

Table 6.1 summarises the quantitative results. The difference between the Python model and the RTL implementation is at most 0.18 dB in NMSE and 0.11 dBc in ACPR across the two tested configurations. This confirms that the RTL design correctly implements the BAPS DPD pipeline and can be used for the reduced-precision hardware evaluation in the following sections.

6.2 Precision Sweep: (R, t) Heatmap

This section presents the results of the joint sweep of the number of active basis functions R and the mantissa width t. The exponent width is fixed at $w = 8$, and

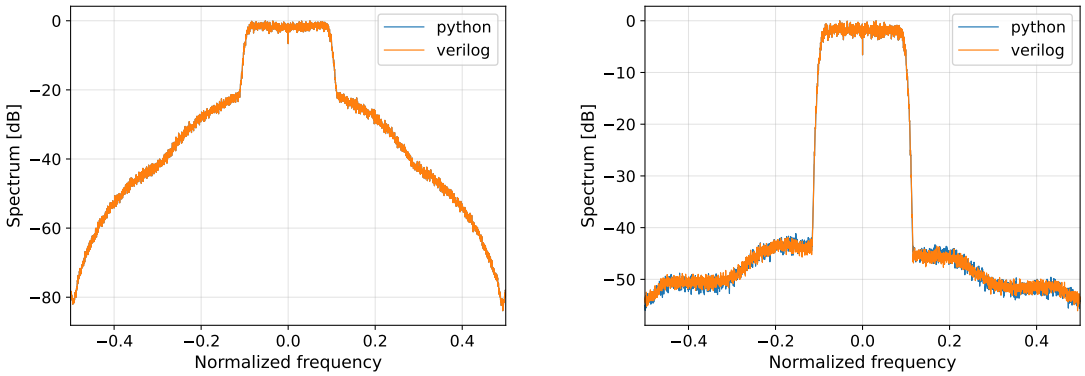


Figure 6.1: Predistorted signal spectrum (left) and PA output spectrum (right) comparing the Python reference and the RTL implementation at $(w, t) = (8, 23)$.

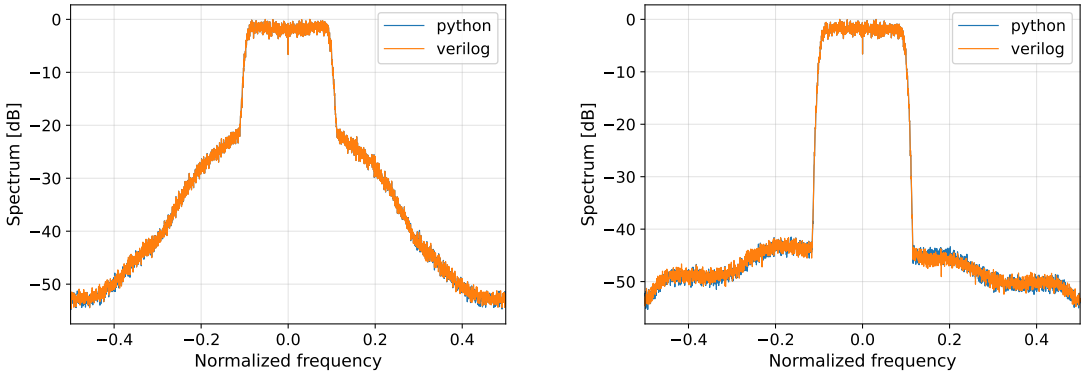


Figure 6.2: Predistorted signal spectrum (left) and PA output spectrum (right) comparing the Python reference and the RTL implementation at $(w, t) = (8, 7)$.

Table 6.1: PA output NMSE and ACPR for the Python reference model and the RTL implementation at two precision configurations. The no-DPD baseline is −22.36 dB NMSE and −32.91 dBc ACPR.

Configuration		Python		RTL	
w	t	NMSE (dB)	ACPR (dBc)	NMSE (dB)	ACPR (dBc)
8	23	−36.60	−42.31	−36.69	−42.40
8	7	−36.12	−42.10	−36.30	−41.99

`pow2` scaling is disabled. Therefore, the results isolate the effect of R and t on DPD performance. Figure 6.3 shows the measured PA NMSE and ACPR heatmaps for the mem1 configuration (see Table 4.1), with R ranging from 4 to 12 and t taking values 5, 6, 7, 8, 9, 10, and 23.

The NMSE heatmap shows two clear trends. In the R direction, the $R = 4$ row is strongly degraded at around -28 dB for all mantissa widths. The $R = 5$ row improves but has not yet reached the performance plateau. For $R \geq 6$, the NMSE enters a stable region at approximately -36 to -36.9 dB, which is close to the full-precision level for this experiment. In the t direction, the $t = 5$ column is consistently the weakest across all values of R, while $t \geq 7$ is sufficient to reach the full-precision ceiling for $R \geq 6$. This suggests that $(R, t) = (6, 7)$ is the smallest tested configuration that achieves near-optimal NMSE performance.

The ACPR heatmap gives a related but not identical view. The $R = 4$ row, which shows severe NMSE degradation, still achieves ACPR values ranging from about -40 to -43 dBc. Except for the $t = 5$ case, most values in this row remain around -42 to -43 dBc. This can be explained by the model structure. At $R = 4$, the model contains two Type II basis functions (see Table 4.1), $\phi_2 = \phi_1|\phi_1|^2$ and $\phi_3 = \phi_2|\phi_1|^2$, with the third-order term ϕ_2 being dominant. These terms capture the main low-order nonlinearity and are sufficient to reduce the spectral regrowth into adjacent channels. However, the model does not have enough basis functions to reconstruct the time-domain waveform accurately, which explains the poorer NMSE.

A second observation is that, for $t = 5$ and $R \geq 8$, ACPR degrades to around -38 to -39 dBc. Larger R introduces longer computation chains in the basis builder, and the limited mantissa precision at $t = 5$ causes accumulated numerical error that eventually affects spectral performance.

These observations show that ACPR can be less sensitive than NMSE to some forms of model degradation. A configuration that appears acceptable under ACPR may still have significant time-domain error. For this reason, NMSE was used as the primary performance metric in the precision exploration. Based on both heatmaps, $(R, t) = (6, 7)$ was identified as the minimum tested setting that gives good performance under both metrics.

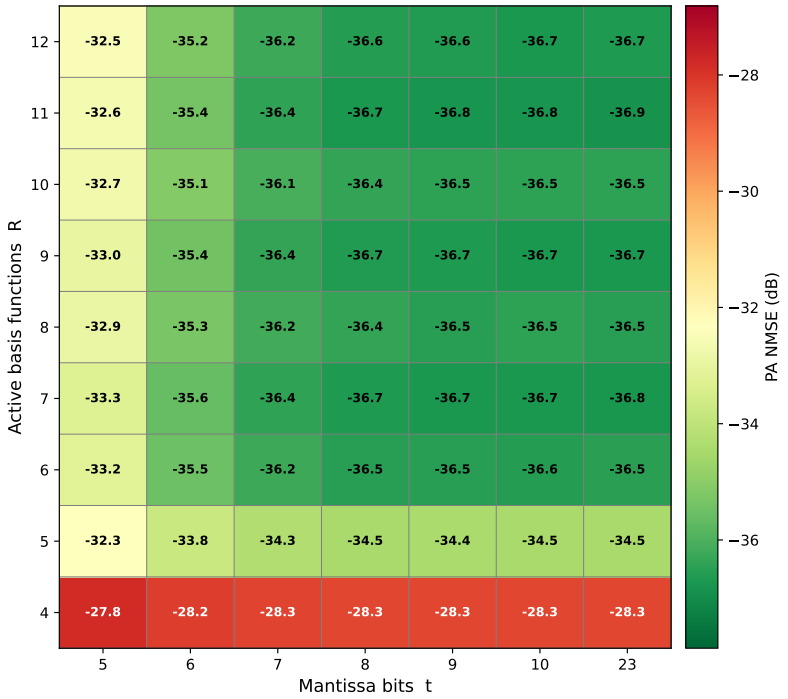
6.3 Exponent Width and Power-of-Two Scaling

This section evaluates the effect of reducing the exponent width and the effectiveness of the power-of-two scaling method. The results were obtained from RTL simulation followed by WebLab PA measurements.

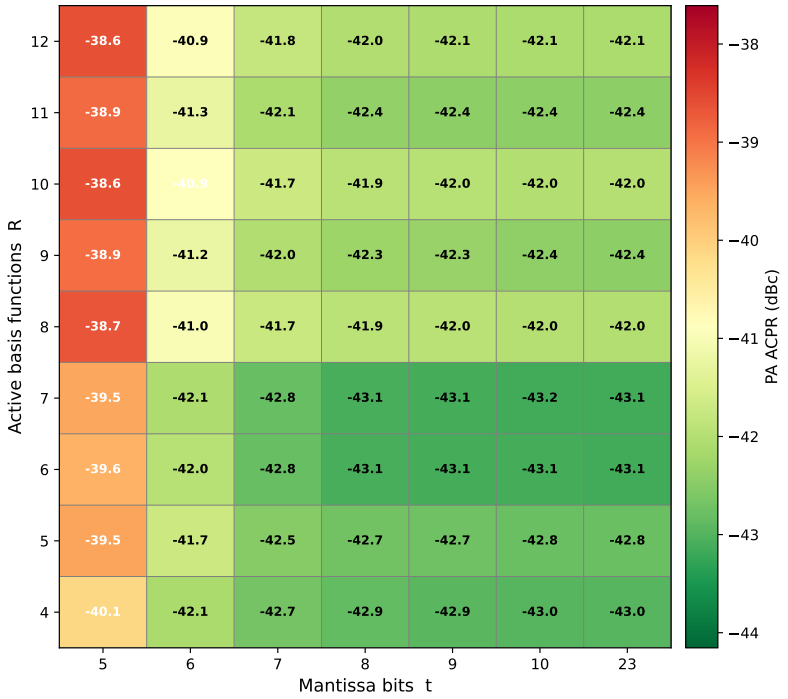
6.3.1 Effect of `pow2` Scaling on Spectral Performance

Figure 6.4 compares the PA output spectra for two representative precision settings.

At $(w, t) = (8, 23)$, both `conv` and `pow2` modes match the float64 reference closely. This shows that the scaling method does not introduce a visible penalty when the exponent range is already sufficient. At $(w, t) = (5, 23)$, the `conv` mode shows

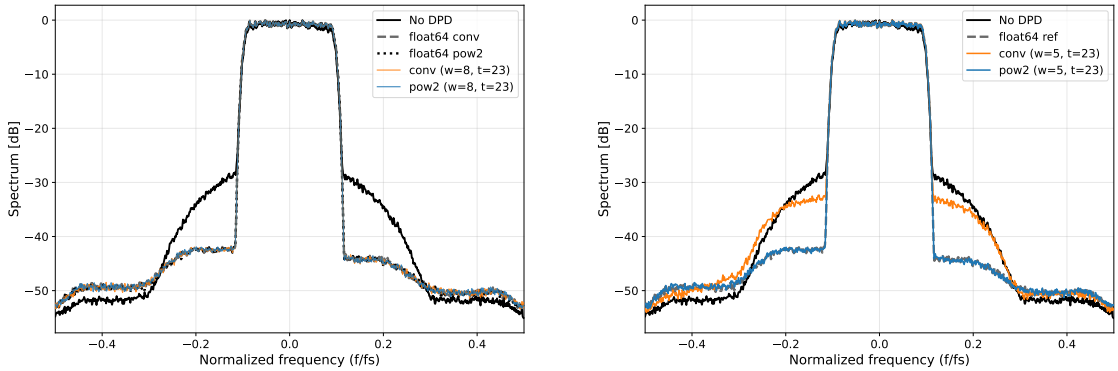


(a) PA NMSE heatmap (no-DPD baseline: −22.27 dB).



(b) PA ACPR heatmap (no-DPD baseline: −32.85 dBc).

Figure 6.3: Measured PA NMSE and ACPR as functions of the number of active basis functions R and mantissa width t, with exponent width $w = 8$ and `pow2` scaling disabled.



(a) $(w, t) = (8, 23)$: `conv` and `pow2` modes both match the float64 reference closely.

(b) $(w, t) = (5, 23)$: `conv` mode shows severe spectral regrowth, while `pow2` mode recovers to the float64 reference level.

Figure 6.4: PA output spectra comparing `conv` and `pow2` scaling modes at two precision configurations.

significant spectral regrowth in the adjacent channels, indicating that the DPD correction has failed. With `pow2` scaling enabled, the spectrum recovers to the float64 reference level. This confirms that the failure is caused mainly by exponent-range limitation, and that the `pow2` method substantially mitigates the underflow problem described in Section 4.2.

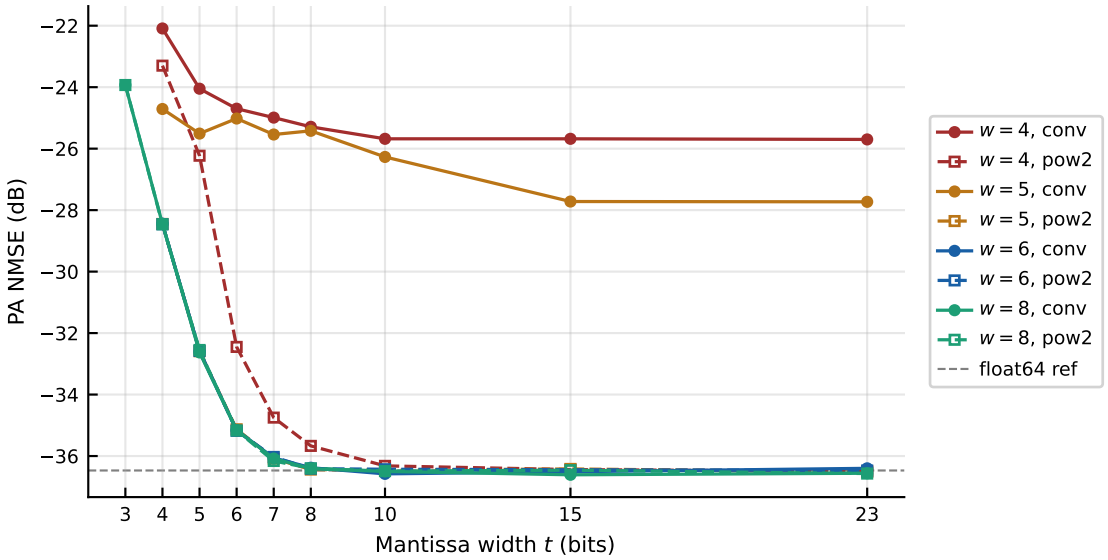
6.3.2 Precision Sweep: Exponent and Mantissa Width

Figure 6.5 shows the PA NMSE and ACPR as functions of mantissa width t for different exponent widths and scaling modes.

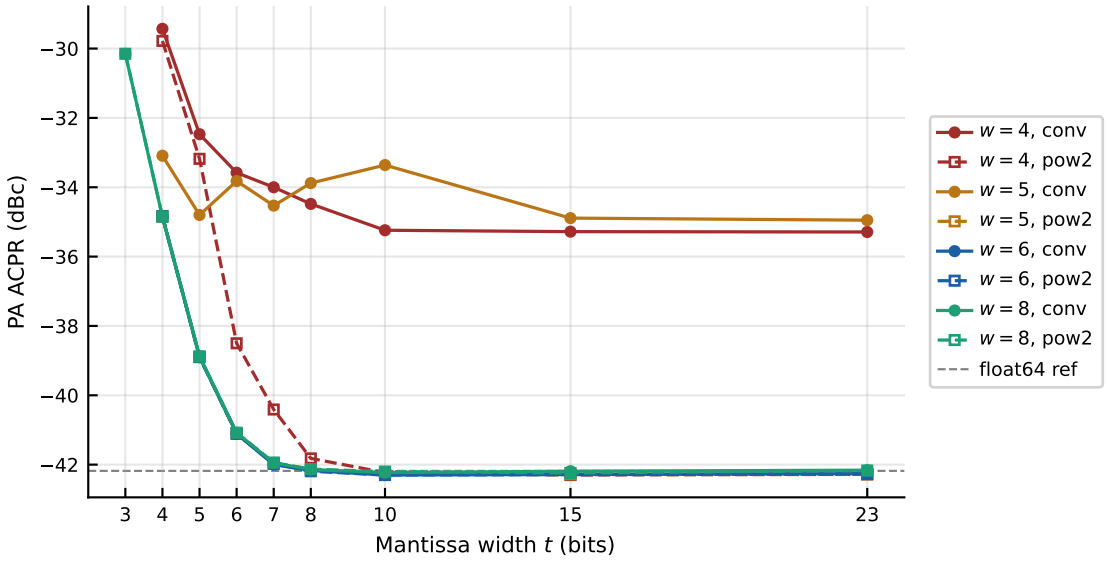
For $w = 8$ and $w = 6$, the `conv` and `pow2` curves are almost identical across all values of t, agreeing with the spectral result in Figure 6.4a. At these exponent widths the available dynamic range is sufficient for the tested signal and PA operating point, so the scaling method gives no observable benefit.

For $w = 5$ in `conv` mode, the NMSE stays near -28 dB regardless of t, so increasing the mantissa width does not recover performance. This confirms that the ceiling is caused by underflow in the exponent field, not by mantissa precision. With `pow2` scaling enabled, the $w = 5$ curve recovers and follows the $w = 6$ and $w = 8$ trend, reaching the float64 reference level at $t \geq 8$.

For $w = 4$, performance stays poor even with `pow2` scaling. The measured NMSE remains around -24 to -25 dB, close to the no-DPD baseline and far worse than the $w = 5$ result of about -36 dB, so the predistorter provides almost no useful linearisation. As discussed in Section 4.3, several stored coefficients of the scaled basis functions fall below the normalised lower bound at $w = 4$ and are rounded to zero. These losses remove the corresponding nonlinear terms and cannot be corrected by pre-scaling the Type II computation alone.



(a) PA NMSE as a function of mantissa width t.



(b) PA ACPR as a function of mantissa width t.

Figure 6.5: PA NMSE and ACPR versus mantissa width t for different exponent widths and scaling modes. Predistorted signals are generated by the Python reference model and measured on WebLab. The dashed line shows the float64 reference.

6.3.3 RTL Validation at $w = 5$ with `pow2` Scaling

The RTL implementation with `pow2` scaling was validated on the RF WebLab platform across a range of mantissa widths at $w = 5$. Figure 6.6 shows the measured PA NMSE as a function of t. Unlike the software precision sweep, which uses the Python FP64 result as a high-precision reference, this RTL validation uses the FP32 configuration ((w,t)=(8,23)) as the reference. FP32 is the full-precision configuration implemented in the RTL design. This comparison therefore isolates the effect of reducing the RTL floating-point precision.

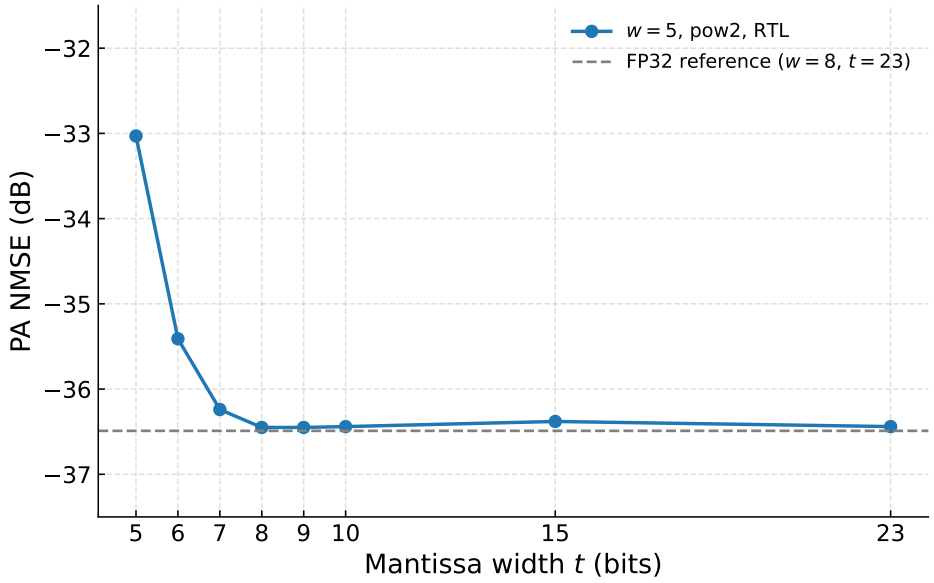


Figure 6.6: PA NMSE as a function of mantissa width t for the RTL implementation at $w = 5$ with power-of-two scaling. The dashed line shows the FP32 RTL reference, $(w, t) = (8, 23)$.

The results confirm the trend observed in the precision sweep. At $t = 7$, the NMSE reaches -36.24 dB, which is 0.26 dB above the FP32 reference of -36.50 dB for this sweep. For $t \geq 8$, the performance stabilises at approximately -36.4 to -36.5 dB, within 0.1 dB of the FP32 reference.

For comparison, the $w = 5$ cases in `conv` mode do not recover when the mantissa width is increased, as shown in Figure 6.5. This confirms that the main performance loss is caused by exponent-range limitation rather than mantissa rounding. In the RTL flow, vulnerable Type II intermediate products can collapse to zero without `pow2` scaling. Once key basis functions are lost, the final DPD output no longer contains the required nonlinear correction terms, and the predistortion performance is lost.

6.4 PA Operating-Point Robustness

This section evaluates how the DPD performance changes when the PA gate voltage V_g is varied. The BAPS mem1 configuration was used for this test. For each

Python floating-point configuration, the model coefficients were identified at the reference operating point $V_g = -2.30$ V. The identified coefficients were then applied across a range of V_g values from -2.10 V to -2.50 V without retraining. Five Python floating-point configurations were compared: the FP32 reference $(w, t) = (8, 23)$, a reduced-mantissa variant $(w, t) = (8, 7)$, two reduced-exponent variants $(w, t) = (6, 10)$ and $(w, t) = (6, 7)$, and the proposed configuration $(w, t) = (5, 7)$ with power-of-two pre-scaling enabled. The first four configurations use `conv` mode and serve as references. The proposed $(5, 7)$ configuration uses the smallest exponent width supported by the floating-point IP, and it relies on pre-scaling to keep the intermediate Type II values inside the representable range.

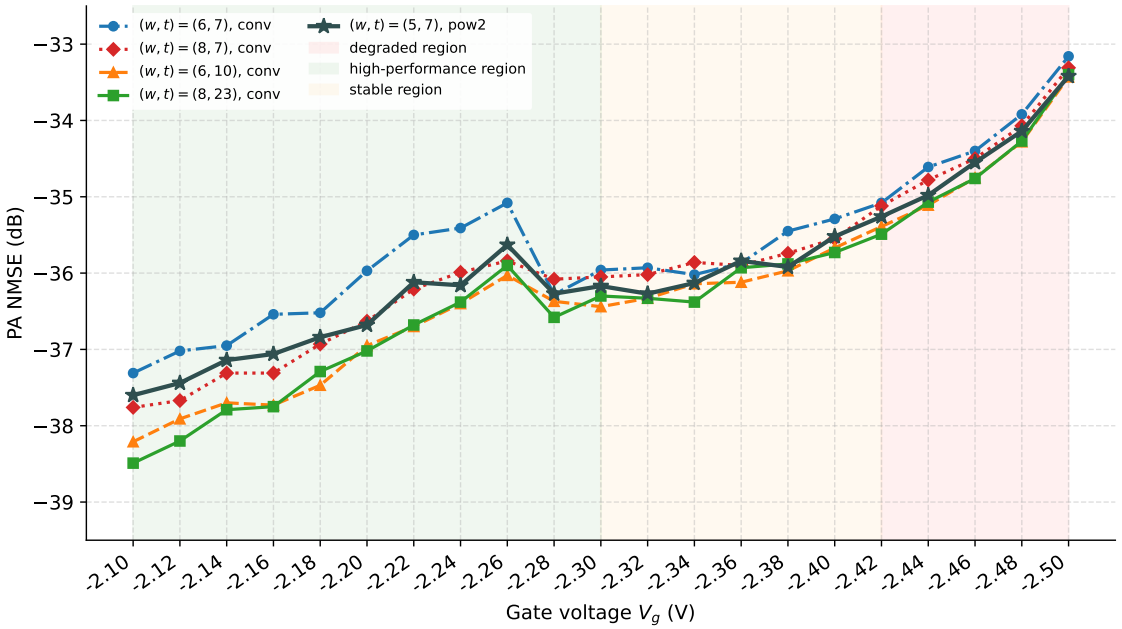


Figure 6.7: PA NMSE versus gate voltage V_g for five precision configurations, with the model identified at $V_g = -2.30$ V and applied without retraining. Four `conv`-mode references are compared with the proposed $(w, t) = (5, 7)$ configuration using power-of-two pre-scaling. Three regions are marked: high-performance (green), stable (orange), and degraded (red).

Figure 6.7 shows the PA NMSE as a function of V_g for all five configurations. Three regions can be identified. In the high-performance region, from $V_g = -2.10$ V to -2.30 V, the PA operates in a more favourable region for the identified DPD model and the measured NMSE is lower. In this region, the $(w, t) = (6, 7)$ `conv` configuration shows a gap of approximately 1 dB relative to the FP32 reference, while $(w, t) = (8, 7)$ and $(w, t) = (6, 10)$ follow the FP32 reference more closely. The proposed $(5, 7)$ configuration with pre-scaling stays inside this high-precision group, and it is consistently better than $(6, 7)$ `conv`, even though it uses a smaller exponent width. This result is consistent with the underflow analysis in Section 4.3. Although the $(5, 7)$ configuration has a narrower exponent range, power-of-two pre-scaling preserves more useful Type II basis-function values than the $(6, 7)$ `conv` configuration. It therefore avoids the loss caused by underflow in the wider $(6, 7)$

`conv` configuration.

In the stable region, from $V_g = -2.30$ V to -2.42 V, all configurations remain close to each other. The gap between $(w, t) = (6, 7)$ `conv` and the FP32 reference narrows to less than 0.5 dB, and the proposed $(5, 7)$ configuration overlaps with the high-precision references. This confirms that in the operating range around the identification point, reduced precision does not cause significant performance degradation.

Beyond $V_g = -2.42$ V, all configurations degrade together. The gap between the precision-reduced configurations and the FP32 reference continues to shrink, reaching less than 0.3 dB at $V_g = -2.50$ V. This indicates that the performance loss in this region is caused mainly by model–PA mismatch rather than by precision limitation, and it affects all configurations in the same way.

Two observations summarise the result. First, $(w, t) = (8, 7)$ and $(w, t) = (6, 10)$ produce nearly identical results across the full V_g range, which shows that reducing the exponent width from 8 to 6 has no measurable effect when the mantissa width is sufficient. Second, and more importantly, the proposed $(5, 7)$ configuration with pre-scaling follows the high-precision references across the complete gate-voltage range, including the high-performance region where the $(6, 7)$ `conv` configuration loses about 1 dB. The proposed configuration is therefore robust to operating-point variation while using the smallest exponent width, which gives the largest hardware saving. This result confirms, directly on PA measurements and across many operating points, the pre-scaling behaviour predicted in Section 4.3.

To check that the observed variation is reproducible and not caused by measurement noise, three repeated measurements were performed at selected V_g values for the $(w, t) = (6, 7)$ configuration. The standard deviation across repeats is at most 0.12 dB, confirming that the trends shown in Figure 6.7 reflect PA behaviour rather than measurement artefacts.

6.5 Hardware Complexity

Table 6.2 summarises the synthesis results for selected precision configurations. All results target the ASAP7 predictive 7 nm FinFET process design kit [23] at a clock period of 5 ns, corresponding to 200 MHz, using Cadence Genus. Area is reported as total area, including both cell area and net area, and is normalised to the FP32 baseline ($w = 8$, $t = 23$, pow2 off) of 114,309 μm^2.

Figure 6.8 shows the synthesised BAPS DPD cell area as a function of mantissa width t for exponent widths $w \in 6, 7, 8$ in `conv` mode, to illustrate the scaling trend of the logic cells. The area comparison in Table 6.2 instead uses total area, including both cell area and net area. The three curves are tightly clustered at every t value, confirming that exponent width has only a minor influence on area. In contrast, the area increases by roughly a factor of six as t grows from 5 to 23. This shows that mantissa width is the dominant factor in hardware complexity. This trend is expected because the `type2_operator` module contains eight real floating-point multipliers whose logic depth and cell count scale strongly with mantissa width.

Table 6.2: Synthesis area results for selected precision configurations targeting the ASAP7 7 nm process at 200 MHz, normalised to the FP32 baseline ($w = 8$, $t = 23$, pow2 off).

w	t	mode	Total area (μm^2)	Norm. area	Slack (ps)
8	23	conv	114,309	1.00	0
6	23	conv	109,270	0.96	0
6	23	pow2	111,603	0.98	0
6	7	conv	24,746	0.22	0
6	7	pow2	25,910	0.23	0
5	7	pow2	24,389	**0.21**	0
5	5	pow2	18,111	0.16	6

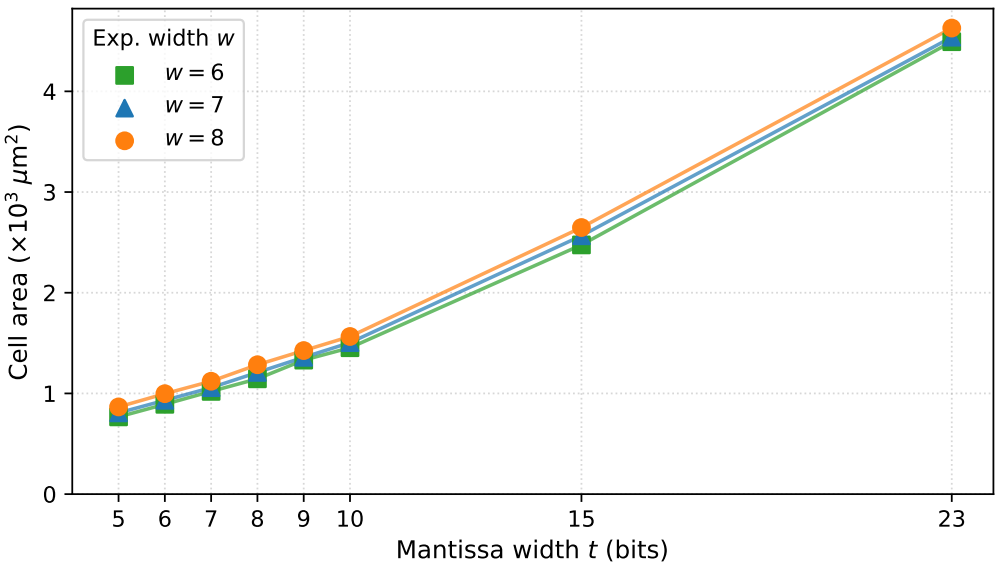


Figure 6.8: Synthesised BAPS DPD cell area versus mantissa width t for $w \in \{6, 7, 8\}$ with `pow2` disabled. The three curves are closely spaced, indicating that exponent width has a secondary effect on area compared with mantissa width.

Reducing the mantissa width from $t = 23$ to $t = 7$ at $w = 6$ in `pow2`-off mode reduces the normalised total area from 0.96 to 0.22, which is about a 78% reduction relative to the FP32 baseline for the current RTL architecture. For configurations that include the `pow2` scaling logic, reducing the exponent width from $w = 6$ to $w = 5$ at $t = 7$ reduces the total area from 25,910 μm^2 to 24,389 μm^2, giving an additional saving of about 5.9%. The configuration ($w = 5, t = 7$, pow2 on) achieves a normalised area of 0.21. All selected configurations were synthesised with a 5 ns clock-period constraint, corresponding to 200 MHz. Their critical paths pass through the Type II operator, and zero timing slack means that the 200 MHz constraint is met exactly.

Adding the pow2 scaling method adds a small but consistent area overhead. The `conv` and `pow2` on results in Table 6.3 are obtained from separately synthesised RTL configurations. The `conv` configuration does not include the pre-scaling and post-scaling compensation logic. Therefore, the area difference represents the total additional hardware required by the `pow2` implementation, including the scaling-related datapath, registers, and control logic.

Table 6.3: Area overhead of separately synthesised `conv` and `pow2` on RTL configurations at $w = 5$. Total area values are synthesis results in μm^2.

t	conv (μm^2)	pow2 (μm^2)	Δ (μm^2)	Overhead
5	17,047	18,111	1,064	6.2%
6	19,983	21,162	1,179	5.9%
7	23,184	24,389	1,205	5.2%
8	26,375	27,773	1,398	5.3%
9	30,390	31,605	1,215	4.0%
10	33,657	34,977	1,320	3.9%
15	58,467	60,698	2,231	3.8%
23	106,693	109,353	2,660	2.5%

The relative overhead decreases from 6.2% at $t = 5$ to 2.5% at $t = 23$, because the floating-point datapath becomes larger as the mantissa width increases. This confirms that the `pow2` method introduces a small hardware cost relative to the area savings it enables.

The configuration $(w, t) = (5, 5)$ with `pow2` scaling gives the smallest area, with a normalised value of 0.16. However, as shown in Section 6.3, $t = 5$ does not reach the DPD performance plateau and is therefore not a practical choice despite its area advantage.

Overall, the results identify $(w, t) = (5, 7)$ with `pow2` scaling as the best practical reduced-precision configuration in this thesis. It reduces the total area to 21% of the FP32 baseline while keeping the PA output NMSE within about 0.3 dB of the FP32 reference and meeting the 200 MHz timing constraint.

7

Discussion

This chapter discusses the main findings from the precision exploration, RTL validation, RF WebLab measurements, and synthesis results. The focus is on what the results mean for reduced-precision BAPS DPD hardware design.

7.1 Interpretation of Precision Sweep Results

The precision sweep results show that, for the evaluated DPD case, the FP32 format provides more mantissa precision and exponent range than required. In the (R, t) heatmap, the NMSE improvement reaches a clear plateau at approximately $t = 7$. Below this value, quantisation error becomes more significant and the DPD output begins to lose accuracy. This indicates that seven mantissa bits are sufficient for this signal and PA operating point, while further reduction to $t = 5$ causes a clear loss of linearisation performance.

The number of basis functions R also affects performance, but only up to a certain point. The results in Section 6.2 show that the performance plateau is reached at about $R = 6$ when $t = 7$. Increasing R beyond this point gives no measurable improvement in NMSE. This suggests that the dominant nonlinear behaviour of the measured PA is already captured by a relatively small BAPS model. For this reason, using a larger number of basis functions would mainly increase hardware area in the current RTL architecture without providing a clear performance benefit.

The exponent width has a different role. For the `conv` mode, no clear exponent-related performance loss is observed for $w \geq 6$. In this range, the basis-function values and coefficients remain within the representable dynamic range of the floating-point format. Therefore, once the mantissa width is large enough, increasing the exponent width from 6 to 8 gives little benefit for the evaluated signal. A larger exponent width may be needed for signals with a wider amplitude range, different PA operating points, or BAPS sequences that produce smaller intermediate values.

The hardware area results in Section 6.5 show that, in the current RTL architecture, reducing t gives the largest measured area reduction. The area scales strongly with t and only weakly with w. This is expected because the mantissa datapath dominates the area of floating-point multiplication. Reducing t from 23 to 7 gives approximately 78% area reduction without measurable loss in DPD performance in the main precision sweep. However, exponent-handling logic in floating-point adders can still affect the critical timing path and the synthesis result [24].

The measured area reduction also depends on the selected RTL architecture. The current design shares arithmetic resources and processes the basis construction and coefficient accumulation sequentially. A more parallel streaming implementation would use a different number of arithmetic units, registers, and buffers. Therefore, the 78% reduction should be interpreted as a result for the current RTL architecture and timing constraint, rather than a general area reduction for all BAPS DPD implementations.

These results also show why $t = 7$ is a useful practical design point. It is close to the lowest mantissa width that still reaches the performance plateau, and it provides a large area saving compared with single precision. Reducing t further may be possible for some signals or PA operating points, but this would require additional methods to control quantisation noise, for example quantisation-aware coefficient identification or a different basis-function sequence.

7.2 Role of the Power-of-Two Scaling Method

The exponent-width results show that reducing w is not always harmless. Although the `conv` mode works well for $w \geq 6$, it fails at $w = 5$ in the RTL implementation. The reason is not ordinary rounding noise, but underflow inside the Type II basis-function computation. When the two multiplication stages inside the `type2_operator` are quantised separately, some intermediate products can fall below the normalised lower bound of the target format and become zero before the final basis function is formed.

This behaviour is important because it is partly hidden in a high-level software model if only the final Type II output is quantised. The failure at $w = 5$ was therefore confirmed through RTL-level validation. This is one of the main findings of the work: for a sequential basis-construction algorithm like BAPS, precision analysis must consider not only the final model output, but also the internal arithmetic stages used to build each basis function.

The `pow2` scaling method addresses this problem by shifting the magnitude of Type II operands using powers of two. Since multiplication by a power of two can be implemented by modifying the exponent field, the method does not require additional floating-point multipliers. For the scaling operation itself, pre-scaling and post-scaling update only the exponent field using signed integer addition. The complete `pow2` RTL also includes the associated registers and control logic.

The method can also be viewed as a floating-point counterpart to explicit scaling in fixed-point hardware. In both cases, powers of two move values into a useful numerical range. The difference is that the present design changes the exponent field, whereas fixed-point hardware would use binary shifts and a selected binary-point position. The synthesis results show that this introduces only a small area overhead, between 2.5% and 6.2%, depending on the mantissa width.

With the `pow2` scaling logic included in the RTL, the $(w, t) = (5, 7)$ configuration recovers performance close to the higher-exponent reference while still keeping the hardware area low. This makes $w = 5$ a viable exponent-width target for the

evaluated BAPS DPD system. In other words, the scaling method extends the usable design space by making a format work that would otherwise fail because of Type II underflow.

However, the method does not remove the exponent-width limit completely. The analysis in Section 4.3.4 shows that $w = 4$ is too small for this signal and PA operating point. At this exponent width, the normalised lower bound is so high that several stored coefficients associated with the scaled Type II basis functions are rounded to zero. These basis functions therefore make no contribution to the final DPD output, even if the basis functions themselves are computed. This explains why `pow2` scaling can recover performance at $w = 5$, but not at $w = 4$.

The practical conclusion is that $w = 5$ is the lower exponent-width bound for the reduced-precision configuration studied here. The final useful design point is therefore not simply the shortest format, but the shortest format that still preserves the important basis functions and their coefficients.

This dynamic-range requirement can also be expressed in fixed-point terms. The required word length depends on which values must be preserved. For the present signal, a format that covers the scaled peak value below 2^7 and has a smallest step of 2^{-14} would need eight bits to the left of the binary point, including the sign bit, and fourteen fractional bits. This corresponds to about 22 bits in total. However, preserving every nonzero value in Table 4.4, including values close to 3.65×10^{-14}, would require about 45 fractional bits and approximately 53 bits in total. This shows why explicit scaling at different computation stages is important for a practical fixed-point implementation.

7.3 Comparison with Prior Work

The work of Yu et al. [16] showed that BAPS DPD can tolerate a large reduction in floating-point precision, especially in mantissa width. Their results identified $t = 7$ as an important threshold and reported a large area reduction compared with single precision. The present work supports this main observation. The results again show that mantissa width is the dominant precision parameter and that $t = 7$ is close to the lowest useful mantissa width for maintaining DPD performance.

This thesis extends the prior work in three main ways. First, the precision sweep is expanded to include the number of basis functions R. The resulting (R, t) heatmap shows not only the mantissa threshold, but also the minimum number of BAPS basis functions needed to reach the performance plateau for the selected BAPS sequence. This makes the design trade-off clearer because both arithmetic precision and BAPS model complexity are considered together.

Second, the low-exponent behaviour is studied in more detail. Earlier results suggested that exponent width could be reduced to $w = 5$ with little performance loss. In this thesis, the RTL analysis shows that this conclusion depends on the internal arithmetic behaviour of the Type II operator. Without scaling, $w = 5$ can fail because of intermediate-product underflow. The `pow2` method is introduced to address this failure mechanism and to make the $w = 5$ configuration practical.

Third, the synthesis target is the complete `baps_dpd` module rather than only the `type2_operator`. This gives a more complete estimate of hardware cost because it includes the FSM, basis register bank, coefficient ROM, multiplier, accumulator, and scaling logic. The result is useful for system-level design decisions, even though the `type2_operator` remains the most precision-sensitive part of the datapath.

The numerical results should not be interpreted as direct one-to-one replacements for those in prior work, because the signal, PA operating point, operation sequence, and synthesis target are not identical. Instead, the value of this work is that it confirms the main precision trend while explaining an additional hardware-level failure mechanism and proposing a low-cost correction for it.

7.4 Limitations

The main precision exploration is performed for one signal type and one main PA operating point. Although the RF WebLab validation and gate-voltage robustness test give useful evidence, the results may change for other signals, bandwidths, PAPR values, and PA bias conditions. A different input amplitude distribution could change the RMS values of the Type II basis functions and therefore change the required `pow2` scaling exponents.

The selected BAPS basis-function configuration is also fixed after the offline greedy search. This is suitable for hardware implementation, but it means that the design does not adapt its basis functions when the PA characteristics change. In practical deployment, the coefficients can be re-identified, but the selected BAPS basis-function configuration would normally remain fixed unless a new offline greedy search is performed. Therefore, the reported performance depends on the suitability of the selected BAPS sequence for the measured PA and signal condition.

The implementation focuses on the mem1 BAPS basis-function configuration used in this work. Deeper memory settings may produce different combinations of Type I and Type II operations. They may also create different dynamic range requirements because delayed and nonlinear terms can appear in different positions in the selected basis-function configuration. For those cases, the `pow2` exponents would need to be recomputed from the corresponding training signal and BAPS basis-function configuration. The generality of the method for deeper memory-depth settings is therefore not fully demonstrated here.

The synthesis results are based on logic synthesis using the ASAP7 predictive process design kit. They provide a useful estimate of relative hardware cost, but they do not include place-and-route effects, clock-tree insertion, routing congestion, or post-layout timing. The reported area values should therefore be interpreted as synthesis-level estimates rather than final silicon implementation results.

The reported 200 MHz value is the clock-frequency constraint used for synthesis, not the achievable input-sample throughput of the present RTL architecture. Because the basis construction and coefficient accumulation are performed sequentially, multiple FSM cycles are required for each output sample. The current design is therefore

intended for precision exploration rather than a 200 MS/s streaming implementation.

For each synthesised precision configuration, the real and imaginary parts of the complex coefficients are written as constant floating-point bit patterns in the `theta_re` and `theta_im` arrays in the top-level RTL module. During Stage 2, the coefficient-multiplication path reads these values sequentially according to the basis-function index. This is suitable for comparing arithmetic precision and data-path cost, but it does not include the additional memory, control logic, or update mechanism that would be required for on-chip coefficient adaptation. Power analysis is also outside the scope of this thesis. A complete power evaluation would require switching-activity data, for example from value change dump (VCD) files, and preferably post-layout simulation.

7.5 Future Work

Future work should first test the proposed method with a wider set of signals and PA operating points. Signals with different PAPR values, bandwidths, and input scaling levels may change the dynamic range of the Type II basis functions. Testing these cases would show whether the selected $(w, t) = (5, 7)$ configuration remains robust or whether the exponent and mantissa requirements need to be adjusted.

A second direction is to extend the precision sweep from two dimensions to three dimensions. In this thesis, the (R, t) heatmap is generated at a fixed exponent width. A future sweep over (R, w, t), with `pow2` scaling enabled, would provide a fuller view of the design space. This could identify whether other combinations of model size, exponent width, and mantissa width offer better trade-offs.

The `pow2` scaling method could also be made more adaptive. In the current design, the scaling exponents are selected offline for one signal and operation sequence. A more flexible flow could recompute these exponents during offline model identification whenever the PA operating point or input signal level changes. This would keep the hardware structure fixed while allowing the scaling parameters to follow the measured signal statistics.

Another useful extension is to evaluate deeper BAPS memory settings and other PA models. This would test whether the same underflow mechanism appears when the operation sequence contains more delay terms or a different balance of Type I and Type II operations. It would also help determine whether the observed $t = 7$ threshold is specific to this experiment or more generally valid for BAPS DPD.

Finally, future hardware work should include place-and-route and power analysis. Post-layout results would provide more accurate area and timing estimates, while switching-activity-based power analysis would show whether the reduced area also leads to meaningful dynamic power reduction.

A comparison with a fixed-point implementation would also be useful. For the reported scaled basis-function range, a single signed fixed-point format would need approximately 22 bits: eight bits to the left of the binary point, including the sign bit, and fourteen fractional bits. This estimate covers peak values below 2^7 and

retains values down to 2^{-14}. A complete fixed-point implementation would also need to consider Type II intermediate products, coefficients, and accumulation. It may therefore require stage-wise scaling or wider internal formats.

7.6 Ethical and Ecological Considerations

AI-assisted tools, including ChatGPT and Claude, were used during the development of this thesis. Their use was limited to language refinement, English grammar correction, LaTeX formatting support, and assistance with repetitive scripting tasks for simulation, verification, and data organisation.

AI tools helped improve writing clarity, correct grammar and syntax issues, and support routine coding or formatting tasks. They were not used to generate experimental data, perform RF WebLab measurements, choose the research direction, or determine the final conclusions.

All implementation work, experimental design, data analysis, result interpretation, and thesis conclusions were reviewed and controlled by the author. The author remains fully responsible for the accuracy, originality, and academic integrity of this work.

Beyond the use of AI tools, the wider aim of this work also has a positive ethical and ecological aspect. Power amplifiers are among the most power-consuming parts of a wireless transmitter. By reducing the hardware area of the BAPS DPD implementation by about 78%, this work supports more area- and power-efficient predistortion hardware. More efficient DPD allows the PA to operate closer to its efficient region while still meeting linearity requirements. This can help lower the energy consumption of wireless infrastructure and reduce its environmental impact, which is increasingly important as the number of deployed base stations grows.

8

Conclusion

This thesis investigated floating-point precision requirements for BAPS-based digital predistortion, with the aim of reducing hardware cost while maintaining linearisation performance. The study considered the mantissa width t, exponent width w, and number of basis functions R. Their effects were evaluated through software simulation, RTL verification, RF WebLab measurements, and logic synthesis.

The joint (R, t) precision sweep showed that the performance plateau is reached at approximately $R = 6$ basis functions and $t = 7$ mantissa bits. Increasing R or t beyond these values gives no measurable improvement for the signal and PA operating point studied in this work. This result confirms that a compact BAPS model with reduced mantissa precision can provide effective DPD performance.

The exponent-width study showed that reducing the exponent width from $w = 8$ to $w = 6$ has little effect on performance. Without scaling, reducing the exponent width to $w = 5$ causes intermediate Type II products to underflow in the RTL implementation. The underflow of these intermediate Type II products removes important basis-function contributions and makes the predistorter inaccurate. This failure mechanism was only exposed through RTL-level analysis, which highlights the importance of hardware-level verification for custom-precision designs. The proposed power-of-two scaling method substantially mitigates this problem and restores DPD performance close to the full-precision reference at $w = 5$, with an area overhead of 2.5%–6.2%, depending on mantissa width. At $w = 4$, several stored coefficients associated with the scaled basis functions fall below the normalised lower bound. Therefore, $w = 5$ remains the practical lower bound for this signal and PA operating point.

The hardware complexity results confirmed that mantissa width is the dominant factor for silicon area. Reducing t from 23 to 7 gives approximately 78% area reduction compared with the single-precision baseline while maintaining DPD performance close to the full-precision result. The configuration $(w, t) = (5, 7)$ with `pow2` scaling is the most area-efficient practical configuration identified in this work. It achieves a normalised area of 0.21 relative to the single-precision baseline while meeting the 200 MHz timing constraint.

Selected configurations were further validated using the RF WebLab platform and PA gate-voltage robustness tests. The results showed that reduced-precision BAPS DPD can maintain performance close to the full-precision reference while providing substantial hardware savings.

The reported area savings are specific to the PA, signal type, and operating conditions studied in this work. In a practical system, the PA and the expected range of transmitted signals should be characterised before selecting a reduced-precision configuration. Changes in bandwidth, PAPR, output power, waveform, or PA operating condition may change the basis-function amplitudes and coefficients. These changes may require further verification or a new DPD identification. Therefore, reduced precision can lower hardware area, but it also leaves less design margin and requires more careful validation over the intended operating range.

Overall, careful control of mantissa precision, exponent range, and basis-function scaling enables a more hardware-efficient BAPS DPD implementation without sacrificing practical predistortion performance.

Bibliography

[1] F. H. Raab, P. Asbeck, S. Cripps, P. B. Kenington, Z. B. Popovic, N. Pothecary, J. F. Sevic, and N. O. Sokal, "Power amplifiers and transmitters for RF and microwave," *IEEE Transactions on Microwave Theory and Techniques*, vol. 50, no. 3, pp. 814–826, 2002.

[2] A. Katz, J. Wood, and D. Chokola, "The evolution of PA linearization: From classic feedforward and feedback through analog and digital predistortion," *IEEE Microwave Magazine*, vol. 17, no. 2, pp. 32–40, Feb. 2016.

[3] F. M. Ghannouchi and O. Hammi, "Behavioral modeling and predistortion," *IEEE Microwave Magazine*, vol. 10, no. 7, pp. 52–64, Dec. 2009.

[4] L. Guan and A. Zhu, "Low-cost FPGA implementation of Volterra series-based digital predistorter for RF power amplifiers," *IEEE Transactions on Microwave Theory and Techniques*, vol. 58, no. 4, pp. 866–872, Apr. 2010.

[5] W. Cao, S. Wang, P. N. Landin, C. Fager, and T. Eriksson, "Complexity optimized digital predistortion model of RF power amplifiers," *IEEE Transactions on Microwave Theory and Techniques*, vol. 70, no. 3, pp. 1490–1499, Mar. 2022.

[6] C. Eun and E. J. Powers, "A new Volterra predistorter based on the indirect learning architecture," *IEEE Transactions on Signal Processing*, vol. 45, no. 1, pp. 223–227, Jan. 1997.

[7] J. Kim and K. Konstantinou, "Digital predistortion of wideband signals based on power amplifier model with memory," *Electronics Letters*, vol. 37, no. 23, pp. 1417–1418, 2001.

[8] D. R. Morgan, Z. Ma, J. Kim, M. G. Zierdt, and J. Pastalan, "A generalized memory polynomial model for digital predistortion of RF power amplifiers," *IEEE Transactions on Signal Processing*, vol. 54, no. 10, pp. 3852–3860, Oct. 2006.

[9] J. Reina-Tosina, M. Allegue-Martínez, C. Crespo-Cadenas, C. Yu, and S. Cruces, "Behavioral modeling and predistortion of power amplifiers under sparsity hypothesis," *IEEE Transactions on Microwave Theory and Techniques*, vol. 63, no. 2, pp. 745–753, Feb. 2015.

[10] J. A. Becerra, M. J. Madero-Ayora, J. Reina-Tosina, C. Crespo-Cadenas, J. García-Frías, and G. R. Arce, "A doubly orthogonal matching pursuit algorithm for sparse predistortion of power amplifiers," *IEEE Microwave and Wireless Components Letters*, vol. 28, no. 8, pp. 726–728, Aug. 2018.

[11] D. Wisell, J. Jalden, and P. Handel, "Behavioral power amplifier modeling using the LASSO," in *Proc. IEEE Instrumentation and Measurement Technology Conference (IMTC)*, 2008, pp. 1091–1095.

[12] P. L. Gilabert, G. Montoro, D. López, N. Bartzoudis, E. Bertran, M. Payaró, and A. Hourtane, "Order reduction of wideband digital predistorters using principal component analysis," in *2013 IEEE MTT-S International Microwave Symposium Digest (MTT)*, Seattle, WA, USA, Jun. 2013.

[13] N. Dwivedi, V. A. Bohara, M. A. Hussein, and O. Venard, "Fixed point digital predistortion system based on indirect learning architecture," in *Proc. International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sep. 2014, pp. 1376–1380.

[14] S. A. Juárez-Cázares, A. Meléndez-Cano, J. R. Cárdenas-Valdez, J. A. Galaviz-Aguilar, C. E. Vazquez-Lopez, P. Roblin, and J. C. Núñez-Pérez, "FPGA-based modeling and design methodology of a digital pre-distortion system for power amplifier linearization," in *Proc. International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)*, Nov. 2016, pp. 113–118.

[15] Mythile C, Jaisiva S, Arunadevi A, and Sudhapriya K, "Examining floating point precision in contemporary FPGAs," in *2024 Third International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*, 2024, pp. 1–7.

[16] Z. Yu, T. S. Mubanda, and P. Larsson-Edefors, "Exploration of short floating-point numbers for hardware-friendly digital predistortion," in *IEEE Nordic Circuits and Systems Conference (NorCAS)*, 2025.

[17] P. N. Landin, S. Gustafsson, C. Fager, and T. Eriksson, "WebLab: A web-based setup for PA digital predistortion and characterization," *IEEE Microwave Magazine*, vol. 16, no. 1, pp. 138–140, Feb. 2015.

[18] S. C. Cripps, *RF Power Amplifiers for Wireless Communications*, 2nd ed. Norwood, MA, USA: Artech House, 2006.

[19] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std. 754-2019, 2019.

[20] M. Henriksson, T. Lindberg, and O. Gustafsson, "APyTypes: Algorithmic data types in Python for efficient simulation of finite word-length effects," in *Proc. IEEE 31st Symposium on Computer Arithmetic (ARITH)*, Jun. 2024, pp. 72–75.

[21] Cadence Design Systems, *Genus Synthesis Solution with ChipWare Component Library*, Cadence Design Systems, Inc., 2023.

[22] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2011.

[23] V. Vashishtha, M. Vangala, and L. T. Clark, "Asap7 predictive design kit development and cell design technology co-optimization: Invited paper," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 992–998.

[24] P. Larsson-Edefors and E. Börjeson, “Efficacy of pipelining to reduce energy of floating-point adders and multipliers,” in *IEEE Symp. Computer Arithmetic (ARITH)*, 2026.