Implementing Gated Recurrent Units On Field-Programmable Gate Array For Brain-Computer Interface Applications

1st Aleksander Berezowski
Schulich School of Engineering
University of Calgary
Calgary, Canada
aleksander.berezowsk@ucalgary.ca

2nd Dr. Eli Kinney-Lang
Schulich School of Engineering
University of Calgary
Calgary, Canada
eli.kinneylang@ucalgary.ca

2nd Dr. Denis Onen
Schulich School of Engineering
University of Calgary
Calgary, Canada
donen@ucalgary.ca

***Abstract*—**

***Index Terms*—**

I. Introduction

Brain-Computer Interfaces (BCIs) enable communication between the human brain and external devices by processing neural signals through machine learning algorithms. These systems follow a structured pipeline encompassing signal acquisition, preprocessing, feature extraction, classification, and device control [1]. Due to the temporal and sequential nature of brain signals, machine learning algorithms capable of capturing dependencies over time are required. Gated Recurrent Units (GRUs) offer a computationally efficient solution for modeling these time-series signals, achieving performance comparable to commonly used Long Short-Term Memory (LSTM) networks while requiring significantly fewer computational resources due to their simplified architecture [7,9].

The deployment platform for BCI systems critically impacts their practical viability, particularly for real-time, low-power applications. Field-Programmable Gate Arrays (FPGAs) provide significant advantages over traditional Central Processing Unit (CPU) and Graphics Processing Unit (GPU) implementations due to their reconfigurable hardware, high parallelism, and superior power efficiency [12-14]. Despite these benefits, the first hardware implementation of a GRU was not presented until 2021 by Zaghloul et al. [11].

This research explores how fundamental design parameters affect the trade-offs between resource utilization, inference speed, power consumption, and accuracy in FPGA-based GRU implementations for BCI applications. Specifically, we investigate the following research question: ”How does input size, hidden state size, and word size affect resource usage, inference time, power consumption, and accuracy of brain-computer interface gated recurrent unit machine learning models deployed on field-programmable gate arrays?”

The primary objective of this research is to systematically characterize how different design parameters affect the performance metrics of hardware-implemented GRUs. By quantifying the trade-offs between competing design goals, this work aims to provide actionable guidance for designing optimal FPGA-based GRUs for BCI systems. Understanding the relationships between parameters is crucial for determining the best GRU design for specific application constraints.

II. Background and Related Work

A. Gated Recurrent Units for BCI Applications

EEG signals used in BCI are often lengthy, one-dimensional, complicated, and nonlinear time sequence signals. Due to these signal characteristics, Recurrent Neural Networks (RNNs) are commonly used to model this data. However, simple RNNs suffer from vanishing and exploding gradient problems, making them struggle to learn long-term dependencies [5]. To address this, Hochreiter et al. proposed Long Short-Term Memory (LSTM) units, which use gates to selectively retain, update, and output information [6].

GRUs, proposed by Cho et al., are a type of hidden unit similar to LSTM units but computationally simpler and more efficient. GRUs use only two gates, compared to LSTMs using four gates, to control information flow through the hidden state allowing them to capture dependencies over multiple time scales [7]. Chung et al. demonstrated that GRUs and LSTMs achieve comparable performance on multiple datasets [9]. Rivas et al. found GRU's and LSTM's performance comparable over several evaluation metrics including Root Mean Square Error (RMSE), Mean Square Error (MSE), and Mean Average Error (MAE) [25].

In modern implementations, GRUs are calculated using Equations (1)–(4) [30-31].

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

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{t-1} + b_{hn})) \tag{2}$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \tag{3}$$

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{t-1} \tag{4}$$

r_t (defined by Equation (1)) is the reset gate, n_t (defined by Equation (2)) is the candidate hidden state, z_t (defined by Equation (3)) is the update gate, and h_t (defined by

Equation (4)) is the final hidden state. The variables are defined as follows: d is the input feature dimension, h is the number of hidden units, $x_t \in \mathbb{R}^d$ is the input vector at timestep t, $h_{t-1} \in \mathbb{R}^h$ is the previous hidden state, $W_{ir}, W_{iz}, W_{in} \in \mathbb{R}^{h \times d}$ are input-to-hidden weight matrices, $W_{hr}, W_{hz}, W_{hn} \in \mathbb{R}^{h \times h}$ are hidden-to-hidden weight matrices, $b_{ir}, b_{iz}, b_{in}, b_{hr}, b_{hz}, b_{hn} \in \mathbb{R}^h$ are bias vectors, σ denotes the sigmoid activation function, $\tanh$ denotes the hyperbolic tangent function, and $\odot$ denotes the Hadamard (element-wise) product.

B. FPGA-Based Neural Network Implementations

FPGAs are well-suited for BCI systems because their reconfigurability allows for flexible deployment and updating of different BCI machine learning models, while their high performance and low power consumption make them ideal for edge deployment close to users [12-14]. In 2021, Cai et al. implemented an epilepsy detection algorithm on an FPGA, demonstrating that FPGA-based BCI systems tend to have higher accuracy than software-based implementations while achieving improved classification performance and reduced power consumption [18].

As shown by Zaghoul et al., the GRU hardware implementation used 50% fewer Buffers, 42% fewer Digital Signal Processors (DSPs), 39% less Block RAM (BRAM), and 35% fewer Slice Look-Up Tables (LUTs) compared to an LSTM implementation [11]. Additionally, the GRU had higher inference performance per Watt and lower execution time [11]. This implementation was later refined by Rizwan et al. in 2025 [33].

The hardware implementation architecture consists of two primary modules: the gate module and the output module. The gate module implements the reset gate, update gate, and activation functions using two Multiply-and-Accumulate (MAC) units that are summed and passed through sigmoid or tanh activation functions. The output module performs Hadamard products and summation to generate the final hidden state [11].

III. Methodology

A. Experimental Design

The experimental system comprises several interconnected Python modules and a Tcl script that automate the entire design, implementation, and analysis pipeline. This automated framework enables systematic exploration of the GRU design space by programmatically generating hardware descriptions, orchestrating FPGA toolchain execution, and extracting performance metrics.

Python scripts automatically generate SystemVerilog code for the GRU module, top-level wrapper, and a testbench based on specified parameters. A Tcl script orchestrates the Xilinx Vivado tool to perform simulation, synthesis, optimization, placement, and routing while generating detailed reports. Additional Python scripts parse Vivado-generated reports and simulation outputs to extract hardware metrics and calculate the accuracy measurements Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

B. Design Space and Variables

Independent Variables

- INT_WIDTH: Number of integer bits in fixed-point representation, determining the range of representable values. The test data ranges from -3.28 to 2.96, requiring a minimum INT_WIDTH of 3 bits based on two's complement representation range of $-2^{\text{INT\_WIDTH}-1}$ to $2^{\text{INT\_WIDTH}-1} - 1$. This study uses INT_WIDTH = 6 to provide sufficient headroom and prevent overflow during intermediate calculations.
- FRAC_WIDTH: Number of fractional bits in fixed-point representation, determining numerical precision. Lower values reduce hardware size but increase quantization error. The tested range is $FRAC\_WIDTH \in \{4, 9, 14, 19, 24\}$, representing a spectrum from coarse precision to floating-point precision [X].
- d (Input Dimension): Input feature dimension corresponding to the number of EEG channels in BCI applications. The dataset used contains 64 channels, thus 64 will be the upper range for d [46]. Özkahraman et al. demonstrated an 83% accuracy using Motor Imagery with just 6 channels, thus 6 will be the lower range for d [47]. The tested range is $d \in \{4, 8, 16, 32, 64\}$.
- h (Hidden State Size): Hidden state dimension, a hyperparameter typically determined during GRU training. During GRU training it was found that the lower range for h was 4, and the upper range was 16. The tested range is $h \in \{4, 6, 8, 12, 16\}$.

Dependent Variables

- Resource Utilization Metrics: Quantity of Lookup Tables (LUTs), flip-flops (registers), Block RAMs (BRAMs), and DSPs (Digital Signal Processors) used.These metrics are extracted from detailed reports output by Vivado.
- Timing Metrics: Worst Negative Slack (WNS) measures timing margin relative to the 100 MHz clock constraint (10ns period). Time utilization is calculated as (10ns - WNS) / 10ns, representing the fraction of the clock period utilized.
- Power Metrics: Total power (W) represents complete FPGA power consumption. Dynamic power (W) measures power from switching activity. Static power (W) quantifies leakage current. All power metrics are extracted from Vivado's power analysis reports.
- Accuracy Metrics: MAE and RMSE quantifies prediction accuracy relative to the ground truth. MAE provides a robust error metric less sensitive to outliers than RMSE, however RMSE does a better job quantifying sensitivity to outliers [27].

Controlled Variables

- Target FPGA: Fixed to a Xilinx Artix-7, ensuring consistent LUT architecture, slice organization, and resource availability across all trials, as this can affect the outputted design [48].
- Clock Frequency: Fixed to 100 MHz, providing a consistent performance target.

- Vivado Version: Fixed to 2024.1, as CAD tool versions can affect synthesis and implementation results [48].
- Synthesis Settings: Identical optimization flags and strategies applied across all designs.
- Test Vectors: The same data is used for each testbench, providing 100 reproducible test cases without random variation.
- Weight Values: Identical pre-initialized weight matrices used across all configurations, ensuring that performance differences result from architectural parameters rather than weight variation.
- Generation Scripts: SystemVerilog generation logic remains constant across all parameter combinations, varying only the parametric inputs.

C. *Trial Execution Flow*

Each experimental trial follows a structured sequence to ensure consistency:

1) RTL Generation: An untested combination of INT_WIDTH, FRAC_WIDTH, d, and h is selected from the design space and used as input parameters for RTL generation. Three SystemVerilog modules are automatically generated: the GRU module implementing Equations (1)–(4), a wrapper module that instantiates the GRU with synthesis preservation directives to prevent logic optimization, and a testbench with 100 deterministic test vectors using a BCI motor imagery dataset for reproducible accuracy evaluation.
2) Vivado Execution: A Vivado project is created and all generated SystemVerilog files are added to it. Synthesis is executed with resource and timing optimizations, placement and routing is performed, and detailed reports on resource utilization, timing analysis, and power consumption are generated. Finally, a simulation is run using the generated testbench to produce a simulated output.
3) Data Capture: The detailed reports are parsed to extract hardware metrics including LUTs, registers, BRAMs, DSPs, WNS, and power consumption.
4) Accuracy Calculation: Each simulated output is compared to a ground truth calculated using floating point numbers using MAE and MSE to determine GRU implementation accuracy.

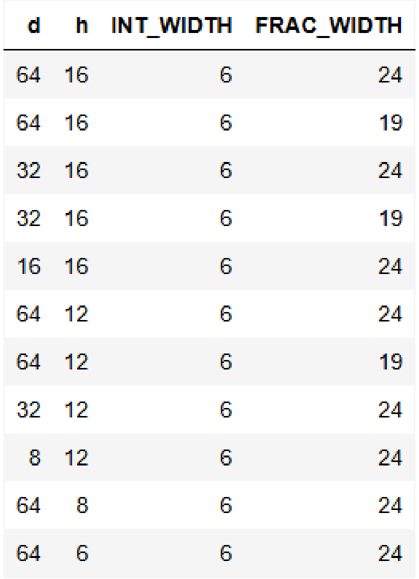
This structured flow ensures that each trial is executed identically and carryover effects are eliminated through complete regeneration of all files.

IV. Results and Analysis

GRU implementations were synthesized for the XCU250-FIGD2104-2L-E FPGA. This particular FPGA was selected solely to ensure sufficient resources for all design variations under investigation, thereby preventing resource constraints from limiting the scope of experimental results. The choice of this specific device does not constrain the applicability of findings; the architectural optimizations and performance characteristics demonstrated herein are device-agnostic and can be applied to any FPGA platform. In practical deployments, the target FPGA should be selected based on the resource requirements of the optimized design rather than constraining the design to fit a predetermined device.

A. *Design Space Exploration Results*

Of the 125 design parameter sets, 11 could not be generated due to insufficient resource availability on the FPGA. This limitation could not be addressed by selecting a larger FPGA, as the largest device available in Xilinx Vivado had already been employed. The failed design parameter sets are presented in Table 1.



d	h	INT_WIDTH	FRAC_WIDTH
64	16	6	24
64	16	6	19
32	16	6	24
32	16	6	19
16	16	6	24
64	12	6	24
64	12	6	19
32	12	6	24
8	12	6	24
64	8	6	24
64	6	6	24

Fig. 1. Table illustrating failed design parameter sets.

B. *Parameter Correlation Analysis*

The correlation matrix presented in Figure 2 reveals several significant relationships between design parameters and performance metrics, providing insights into the fundamental trade-offs inherent in FPGA-based GRU implementations.

1) Resource Utilization Patterns: The hidden state size (h) exhibits the strongest positive correlation with hardware resource consumption, demonstrating correlations of 0.41 with LUTs, 0.50 with registers, and 0.27 with DSPs. This relationship reflects the quadratic scaling of hidden-to-hidden weight matrices ($W_{hr}, W_{hz}, W_{hn} \in \mathbb{R}^{h \times h}$) with respect to h, as defined in Equations (1)–(4). The input dimension (d) demonstrates a moderate positive correlation with LUT utilization (0.36) and DSP usage (0.18), attributable to the linear scaling of input-to-hidden weight matrices ($W_{ir}, W_{iz}, W_{in} \in \mathbb{R}^{h \times d}$). Conversely, FRAC_WIDTH shows a weak negative correlation with LUT consumption (-0.29), suggesting that increased precision does not proportionally increase combinational logic requirements, likely due to the fixed-width arithmetic units in the implementation.

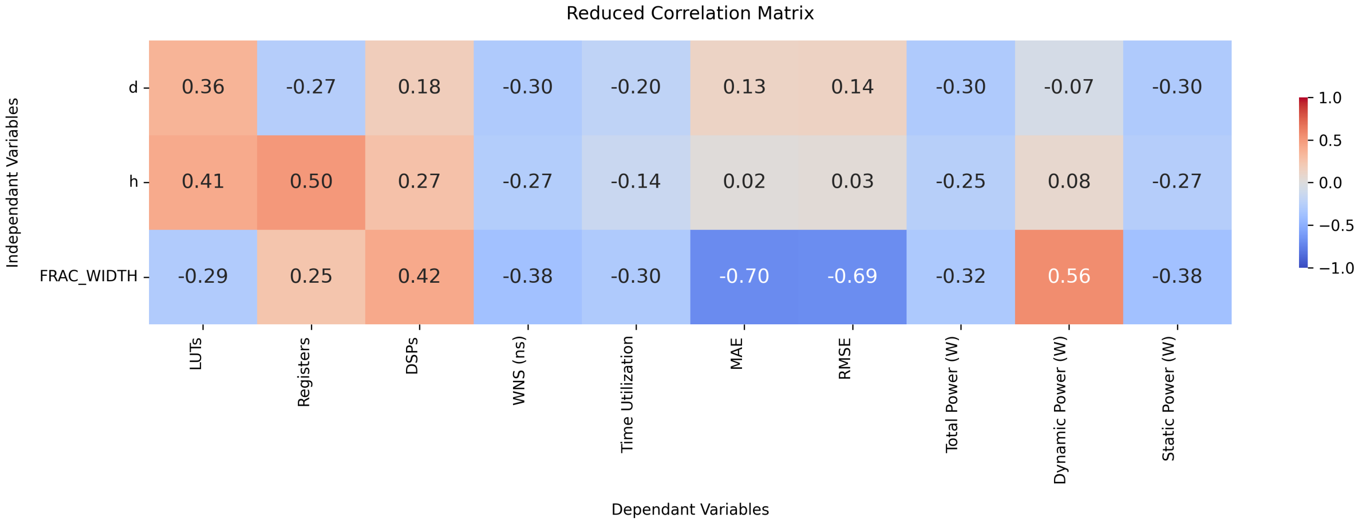


Fig. 2. Correlation matrix illustrating the relationships between independent variables and dependent variables.

2) Timing Performance: Both WNS and time utilization demonstrate inverse relationships with architectural complexity. The hidden state size exhibits negative correlations with WNS (-0.27) and time utilization (-0.14), indicating that larger hidden states impose greater timing constraints due to increased computational depth and interconnect complexity. Similarly, the input dimension shows negative correlations with WNS (-0.30) and time utilization (-0.20), reflecting the additional logic delays introduced by wider input vectors. FRAC_WIDTH presents a more substantial negative correlation with both WNS (-0.38) and time utilization (-0.30), demonstrating that higher precision arithmetic operations introduce longer critical paths through the combinational logic.

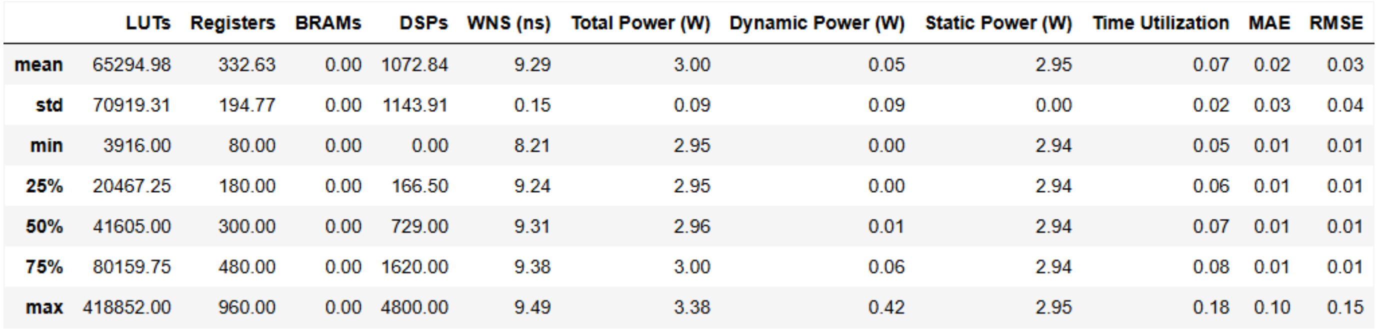
3) Accuracy Characteristics: FRAC_WIDTH exhibits the strongest influence on implementation accuracy, with correlation coefficients of -0.70 for MAE and -0.69 for RMSE. These strong negative correlations confirm that increased fractional precision directly reduces quantization error, as expected from fixed-point number theory. The architectural parameters h and d show negligible correlations with accuracy metrics (MAE: 0.02 and 0.13; RMSE: 0.03 and 0.14, respectively), indicating that network topology does not significantly affect numerical precision in hardware implementations when precision is held constant.

4) Power Consumption: Total power consumption demonstrates modest negative correlations with d (-0.30), h (-0.25), and FRAC_WIDTH (-0.32), contrary to the expected positive relationship between circuit complexity and power dissipation. This counterintuitive finding warrants further investigation and may result from Vivado's power estimation methodology or interactions between resource utilization and routing efficiency. Dynamic power exhibits a strong positive correlation with FRAC_WIDTH (0.56), suggesting that higher precision arithmetic operations generate increased switching activity, consistent with the larger number of bits transitioning during computations. The architectural parameters d and h show weak negative correlations with dynamic power (-0.07 and 0.08, respectively), indicating minimal impact of network topology on switching activity. Static power demonstrates negative correlations with all design parameters (d: -0.30, h: -0.27, FRAC_WIDTH: -0.38), mirroring the pattern observed in total power consumption. This relationship suggests that static power may be influenced by factors beyond simple resource count, potentially including variations in routing congestion, placement density, or device-level power optimization strategies employed by the synthesis toolchain. The correlation patterns between static and total power are nearly identical, indicating that static power constitutes a significant fraction of total power dissipation in these implementations.

5) Design Trade-off Implications: The correlation analysis reveals a fundamental design trade-off between accuracy and hardware efficiency. FRAC_WIDTH simultaneously improves accuracy (strong negative correlations with MAE and RMSE) while degrading timing performance (negative correlation with WNS) and increasing dynamic power consumption (positive correlation of 0.56). The hidden state size primarily affects resource utilization with minimal impact on accuracy, suggesting that h can be optimized for computational requirements without significantly compromising numerical precision. The relatively weak correlations between structural parameters (d, h) and timing metrics indicate that the implemented architecture maintains acceptable timing margins across the explored design space, with FRAC_WIDTH serving as the primary determinant of critical path delay.

V. Discussion

Figure 3 shows a description of the output variables, which is used in this discussion.



	LUTs	Registers	BRAMs	DSPs	WNS (ns)	Total Power (W)	Dynamic Power (W)	Static Power (W)	Time Utilization	MAE	RMSE
mean	65294.98	332.63	0.00	1072.84	9.29	3.00	0.05	2.95	0.07	0.02	0.03
std	70919.31	194.77	0.00	1143.91	0.15	0.09	0.09	0.00	0.02	0.03	0.04
min	3916.00	80.00	0.00	0.00	8.21	2.95	0.00	2.94	0.05	0.01	0.01
25%	20467.25	180.00	0.00	166.50	9.24	2.95	0.00	2.94	0.06	0.01	0.01
50%	41605.00	300.00	0.00	729.00	9.31	2.96	0.01	2.94	0.07	0.01	0.01
75%	80159.75	480.00	0.00	1620.00	9.38	3.00	0.06	2.94	0.08	0.01	0.01
max	418852.00	960.00	0.00	4800.00	9.49	3.38	0.42	2.95	0.18	0.10	0.15

Fig. 3. Description of the dependent variables.

A. *Static Power Dominance and Idle Operation*

The descriptive statistics reveal that static power consumption (mean: 2.95 W, standard deviation: 0.00 W) constitutes the dominant component of total power dissipation (mean: 3.00 W, standard deviation: 0.09 W), while dynamic power contributes minimally (mean: 0.05 W, standard deviation: 0.09 W). This distribution directly correlates with the observed timing characteristics, where the mean WNS of 9.29 ns indicates substantial timing margin relative to the 10 ns clock period constraint, resulting in a mean time utilization of only 7%. Consequently, the FPGA remains idle for approximately 93% of each clock cycle, with logic transitions occurring only during the brief computation window. This idle time explains the negligible dynamic power consumption, as switching activity, the primary driver of dynamic power dissipation, occurs infrequently. The near-constant static power across all design configurations (range: 2.94-2.95 W) reflects continuous leakage current independent of computational activity, confirming that power optimization in these implementations must primarily address static rather than dynamic power consumption.

B. *Temporal Multiplexing and Bit-Serial Optimization Opportunities*

The vast disparity between FPGA clock frequencies (100 MHz in this study) and typical BCI sampling rates (on the order of Hz) [39] presents a substantial optimization opportunity through temporal multiplexing across multiple clock cycles. The FPGA executes approximately one million clock cycles between successive BCI data samples, yet the current implementations complete their computations within a single clock cycle and remain idle while awaiting new input data [40]. The exceptionally low time utilization (mean: 7%, maximum: 18%) quantifies the underutilization within each clock cycle, but the more significant inefficiency lies in the extended idle periods between BCI samples, during which the hardware remains powered but performs no useful computation. By distributing GRU computations across clock cycles through time-multiplexed architectures, hardware resource requirements could be dramatically reduced without compromising real-time processing requirements. One particularly effective implementation of temporal multiplexing is through bit-serial architectures, which compute results sequentially one bit position at a time rather than processing all bits in parallel, trading temporal resources for spatial efficiency [38]. While parallel addition of two 32-bit integers requires 32 adders operating in a single clock cycle, an equivalent bit-serial implementation utilizes only one adder over 32 cycles. This architectural approach could significantly reduce the LUT, register, and DSP resource requirements observed in the current implementations. The observed timing margins (mean WNS: 9.29 ns) indicate that extending individual arithmetic operations across multiple clock cycles would not violate timing constraints. Given the million-fold difference between FPGA clock rates and BCI sampling rates, even fully bit-serial implementations of all arithmetic operations would complete well within the inter-sample interval, making this aggressive hardware minimization strategy viable for BCI applications while fundamentally shifting the design paradigm from spatial parallelism to temporal efficiency. This and other temporal reuse strategies could achieve order-of-magnitude reductions in hardware consumption while maintaining sufficient throughput to process BCI signals.

C. *Diminishing Returns of Fractional Precision*

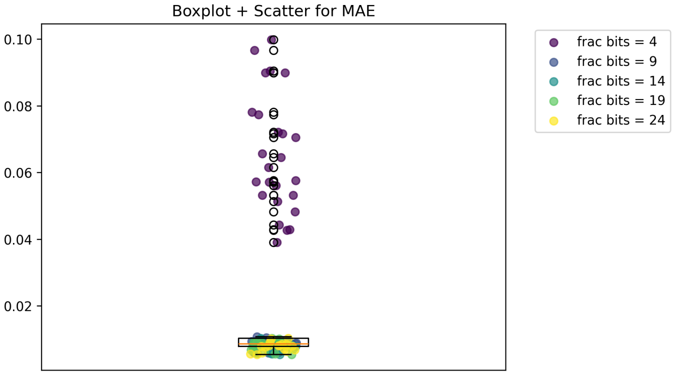


Fig. 4. Boxplot and scatter visualization for MAE.

The boxplot and scatter visualizations for MAE (Figure 4) and RMSE (Figure 5) reveal a critical threshold effect in the relationship between fractional precision and accuracy. While FRAC_WIDTH = 4 exhibits substantially degraded

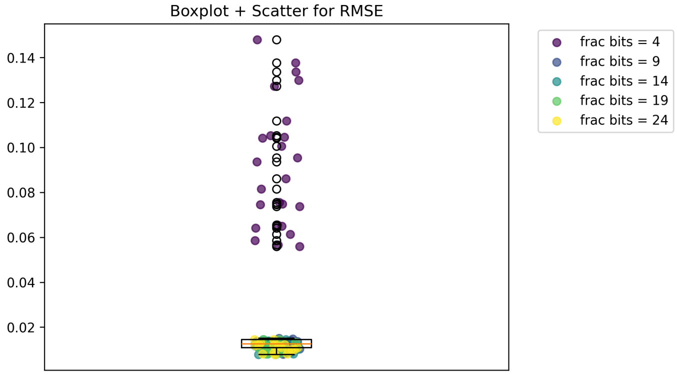


Fig. 5. Boxplot and scatter visualization for RMSE.

accuracy with high error dispersion (MAE range: approximately 0.04-0.10, RMSE range: approximately 0.06-0.15), the accuracy improvements diminish rapidly beyond moderate precision levels. The implementations with FRAC_WIDTH $\in \{9, 14, 19, 24\}$ form tightly clustered distributions in the lower portion of both plots, with both boxplots indicating statistically similar performance. The marginal reduction in error between FRAC_WIDTH = 14 and FRAC_WIDTH = 24 is negligible relative to the substantial hardware and timing costs associated with higher precision arithmetic, as evidenced by the correlation analysis showing FRAC_WIDTH's strong negative impact on WNS (-0.38) and positive impact on dynamic power (0.56). This plateau effect suggests that FRAC_WIDTH values beyond approximately 9-14 bits provide minimal accuracy benefits while incurring disproportionate implementation costs, establishing an optimal precision threshold for resource-constrained FPGA deployments. The clustering pattern indicates that practitioners can confidently select moderate precision configurations (FRAC_WIDTH $\approx$ 9-14) to achieve near-optimal accuracy while minimizing hardware overhead, critical path delays, and power consumption.

VI. Conclusion and Future Work

Summary of contributions Shift-and-add multiplication investigation Real-world validation plans

VII. References