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], and current FPGA implementations have not fully explored available hardware-specific optimizations such as shift-and-add multiplication, bit-serial structures, and improved activation function implementations.

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, word size, and the usage of shift-and-add multiplication 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. Different parameter combinations are hypothesized to yield varying trade-offs, and understanding these relationships is crucial for determining the best GRU design for specific application constraints.

II. Background and Related Work

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].

Gated Recurrent Units (GRUs), proposed in 2014 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—a reset gate and an update gate—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], while Rivas et al. found their 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 the following functions [30-31]:

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

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

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

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

Where r_t is the reset gate, z_t is the update gate, n_t is the candidate hidden state, h_t is the final hidden state, and $\odot$ denotes the Hadamard product.

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].

Despite GRUs having a smaller design architecture and fewer computations compared to LSTMs, the first hardware implementation of a GRU was not presented until 2021 by Zaghoul et al. [11]. As shown in their work, the GRU hardware implementation used 50% fewer Buffers, 42% fewer Digital Signal Processors (DSPs), 39% less Block RAM, and 35% fewer Slice Look-Up Tables (LUTs) compared to an LSTM implementation. 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

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 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 accuracy measurements. The main orchestration script (main.py) coordinates all phases, collects results, calculates Mean Absolute Error (MAE) metrics, and compiles data into a structured format for analysis.

Trial Execution Flow

Each experimental trial follows a structured sequence to ensure consistency and data integrity:

Environment Initialization: The system clears previous results and initializes tracking files (status.txt) to monitor progress through the design space.

Parameter Configuration: For each combination of INT_WIDTH, FRAC_WIDTH, d, and h, the system sets these values as constants in the code generation scripts.

RTL Generation: The script creates the GRU module implementing the mathematical operations (reset gate, update gate, candidate hidden state, and final hidden state computation) with saturating arithmetic and fixed-point multiplication. The produces a wrapper module that instantiates the GRU with synthesis preservation directives to prevent optimization from removing logic. The script generates a testbench with 100 test vectors using deterministic mathematical functions (sine waves) for reproducible accuracy testing.

Vivado Execution: The TCL script performs the following operations: creates a Vivado project targeting the Artix-7 FPGA (xc7a100tcsg324-1); adds all generated SystemVerilog source files with appropriate file type settings; applies timing constraints from constraints.xdc (100 MHz clock target); runs behavioral simulation for 500ns to generate output predictions; executes synthesis with resource and timing optimizations; performs placement and routing; and generates detailed reports on resource utilization, timing analysis, and power consumption.

Data Capture: The script parses multiple report formats using regular expressions to extract hardware metrics including LUTs, registers, BRAMs, DSPs, WNS, and power consumption. Simulation outputs are written to text files containing fixed-point binary representations of GRU predictions. The module handles conversion between floating-point and fixed-point representations using two's complement encoding.

Accuracy Calculation: For each group sharing the same d, h, and INT_WIDTH, the implementation with maximum FRAC_WIDTH serves as the ground truth. MAE is computed between all other configurations and this reference using the function, defined as:

where y_i represents the ground truth output and ŷ_i represents the predicted output for test vector i.

Error Handling: The system implements comprehensive exception handling to capture failures including resource overflow, routing failures, and file access errors, recording the failure mode for later analysis.

This structured flow ensures that each trial is executed identically, carryover effects are eliminated through complete regeneration of all files, and data integrity is maintained through automated validation checks.

Design Space and Variables

Independent Variables

INT_WIDTH: Number of integer bits in fixed-point representation (excluding sign bit), 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 multiplier complexity but increase quantization error. The tested range is $FRAC\_WIDTH \in \{2, 4, 6, 8\}$, representing a spectrum from coarse to moderate precision. The upper bound of 8 bits was selected to balance accuracy requirements with reasonable hardware costs for the initial design space exploration.

d (Input Dimension): Input feature dimension corresponding to the number of EEG channels in BCI applications. The tested range is $d \in \{4, 5, 6, 7, 8\}$. This range was selected based on computational constraints while capturing the relationship between input dimension and hardware utilization. Real-world BCI systems typically use 6-64 channels [47], so this range represents smaller-scale implementations suitable for resource-constrained deployments.

h (Hidden State Size): Hidden state dimension, a hyperparameter typically determined during GRU training. The tested range is $h \in \{4, 5, 6, 7, 8\}$. These values represent architecturally small GRU configurations appropriate for the target FPGA platform and enable comprehensive design space exploration within reasonable computational time.

Dependent Variables

Resource Utilization Metrics: LUTs (Lookup Tables) quantify combinational logic resources; registers (flip-flops) measure sequential logic requirements; BRAMs (Block RAMs) indicate memory usage; and DSPs (Digital Signal Processor slices) count hardware multipliers. These metrics are extracted from Vivado's using regex pattern matching for "CLB LUTs," "CLB Registers," "Block RAM Tile," and "DSPs."

Timing Metrics: WNS (Worst Negative Slack) measures timing margin relative to the 100 MHz clock constraint (10ns period), extracted. 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; and static power (W) quantifies leakage current. All power metrics are extracted from generated by Vivado's power analysis.

Accuracy Metrics: MAE (Mean Absolute Error) quantifies prediction accuracy relative to the highest-precision ground truth implementation within each parameter group. MAE provides a robust error metric less sensitive to outliers than squared error measures [27].

Controlled Variables

Target FPGA: Fixed to Xilinx Artix-7 xc7a100tcsg324-1, ensuring consistent LUT architecture, slice organization, and resource availability across all trials [48].

Clock Frequency: Fixed to 100 MHz via timing constraints in constraints.xdc, 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: Deterministic sine-based functions in testbench provide 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.

Data Collection and Analysis

The experimental data is exported to CSV format using pandas during the main.py execution phase. Each row represents one design configuration with its associated independent variables (INT_WIDTH, FRAC_WIDTH, d, h) and measured dependent variables (resource utilization, timing, power, accuracy).

Post-processing and visualization are performed in a Jupyter notebook environment. Initial data preparation includes removal of constant columns (controlled variables) to focus on varying parameters, and filtering of failed implementations (identified by LUTs = 0) which indicate designs exceeding resource constraints or failing placement and routing.

Correlation analysis examines relationships between independent and dependent variables using Pearson correlation coefficients. A reduced correlation matrix focuses on primary variables of interest: d, h, FRAC_WIDTH versus LUTs, Registers, DSPs, WNS, Time Utilization, MAE, Total Power, and Dynamic Power. This visualization clarifies which parameters most strongly influence each hardware metric and supports identification of dominant factors in accuracy-resource trade-offs.

Descriptive statistics (mean, standard deviation, minimum, maximum, quartiles) characterize the distribution of all variables, providing context for interpreting individual results and identifying outliers or unexpected patterns.

Pairwise scatter plots enable visual inspection of relationships between all variable pairs, revealing non-linear relationships, threshold effects, and correlation structures that may not be captured by linear correlation coefficients alone.

This methodology provides a systematic, reproducible framework for exploring FPGA-based GRU implementations. The automated pipeline ensures consistency across thousands of potential configurations, while the comprehensive metrics collection enables multi-objective analysis of the complex trade-offs inherent in hardware accelerator design for BCI applications.

IV. Results and Analysis

GRU implementations were evaluated across a design space with the following parameters: INT_WIDTH = 6, $FRAC\_WIDTH \in \{2, 4, 6, 8\}$, $d \in \{4, 5, 6, 7, 8\}$, and $h \in \{4, 5, 6, 7, 8\}$, targeting the PUT FPGA PART HERE. This yielded XXX unique configurations, each processed through automated SystemVerilog generation, Vivado synthesis and implementation, and metrics extraction.

Design Space Exploration Results Parameter Correlation Analysis Accuracy vs. Resource Trade-offs Performance Comparisons

V. Discussion

Interpretation of findings Design guidelines for FPGA-based GRU implementations Practical implications for BCI systems

VI. Conclusion and Future Work

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

VII. References