International Journal of Scientific Research in Engineering & Technology
Volume6, Issue4 (July-August 2026), PP: 01-14.
https://www.doi.org/10.59256/ijsreat.20260604001
www.ijsreat.com
ISSN No: 2583-1240

Design and FPGA Implementation of GELU Activation Function Using Verilog HDL

Dr. Nuthan AC[1], Shruthi V[2], Monish Gowda D S[3]
[1,2,3]*Department of Electronics and Communication Engineering, GMIT, Bharathinagara, Karnataka, India.*

To Cite this Article: *Dr. Nuthan AC[1], Shruthi V[2], Monish Gowda D S[3], "Design and FPGA Implementation of GELU Activation Function Using Verilog HDL", International Journal of Scientific Research in Engineering & Technology, Volume 06, Issue 04, July-August 2026, PP:01-14.*

Abstract: *Deep neural networks increasingly rely on smooth, non-monotonic activation functions such as the Gaussian Error Linear Unit (GELU), which has become the default activation in modern transformer and convolutional architectures. While GELU improves accuracy, its exact form involves the Gaussian error function, which is expensive to evaluate in hardware, so deploying such models on resource-constrained edge devices demands a hardware-efficient yet accurate approximation. This work presents the design and FPGA implementation of the GELU activation function using Verilog HDL. The proposed architecture adopts a hybrid CORDIC–polynomial approximation: a lightweight polynomial datapath performs the cubic pre-shaping ($x + 0.044715x^3$) and the $\sqrt{(2/\pi)}$ scaling, while a pipelined hyperbolic-CORDIC core evaluates the hyperbolic tangent without dedicated multipliers. The complete design uses Q7.25 signed fixed-point arithmetic and a finite-state-machine controller, and is realised as a reconfigurable accelerator supporting GELU, ReLU, and SILU. Following a hardware–software co-design methodology, a bit-accurate Python golden model is used to study quantization and to verify the Verilog implementation, after which the design is synthesised and implemented on the Xilinx Artix-7 (XC7A35T) device of the Basys 3 board using Xilinx Vivado. Functional verification over the input range $[-3, +3]$ shows a maximum absolute error of approximately 1.3×10^{-4} with respect to the ideal GELU. The implemented accelerator consumes 2157 look-up tables (10%), 1856 flip-flops (4%), and 18 DSP slices (20%), dissipates a total on-chip power of 0.182 W, and meets timing at a 100 MHz clock with a positive worst-negative slack of 0.083 ns (maximum frequency ≈ 100.8 MHz). These results demonstrate accurate, low-power, and area-efficient activation computation suitable for real-time neural-network inference in edge-AI and biomedical applications.*

Key Words: *GELU, activation function, FPGA, Verilog HDL, CORDIC, fixed-point arithmetic, hardware–software co-design, edge AI, Artix-7, Basys 3.*

I. INTRODUCTION

Artificial intelligence and deep learning now underpin applications ranging from image recognition and natural-language processing to real-time biomedical signal analysis. At the heart of every deep neural network is a set of nonlinear activation functions that allow the network to learn complex mappings between its inputs and outputs; without nonlinearity a multilayer network would collapse into a single linear transformation regardless of its depth. In recent years there has been a strong push to move neural-network inference from the cloud to the edge onto compact, battery-powered devices that operate close to the data source because edge deployment reduces latency, preserves privacy, and removes the dependence on a continuous network connection. Edge platforms, however, are severely constrained in silicon area, memory, and power, which makes the efficient hardware realisation of every computational block, including the activation function, a first-order design concern.

Classical activation functions such as the sigmoid and hyperbolic tangent are smooth and bounded but suffer from vanishing gradients for large-magnitude inputs, whereas the Rectified Linear Unit (ReLU), max(0, x), is extremely cheap to implement but discards all negative information and can produce dead neurons. To overcome these limitations, a family of smooth, non-monotonic activations notably the Sigmoid Linear Unit (SILU/Swish) and the Gaussian Error Linear Unit (GELU) has been introduced. GELU in particular has become the default activation in transformer architectures such as BERT and GPT and in many state-of-the-art vision models, owing to its consistently superior accuracy.

The GELU Activation Function

The Gaussian Error Linear Unit weights its input by the probability that the input exceeds a standard normal random variable, so that $GELU(x) = x \cdot \Phi(x)$, where $\Phi(x)$ is the standard-normal cumulative distribution function. Expressed through the Gaussian error function, the exact form is $GELU(x) = 0.5 \cdot x \cdot [1 + erf(x/\sqrt{2})]$. The function is smooth, differentiable everywhere, and non-monotonic, dipping slightly below zero for small negative inputs before approaching the identity for large positive inputs, as shown in Fig. 1.

While its accuracy benefits are well established, the exact form of GELU is ill-suited to direct hardware implementation: the error function has no closed-form elementary expression and is normally evaluated through series expansions or large look-

up tables, both expensive in memory and logic. Hardware designers therefore rely on approximations that preserve accuracy while reducing cost most commonly the hyperbolic-tangent approximation that forms the basis of the architecture proposed in this work.

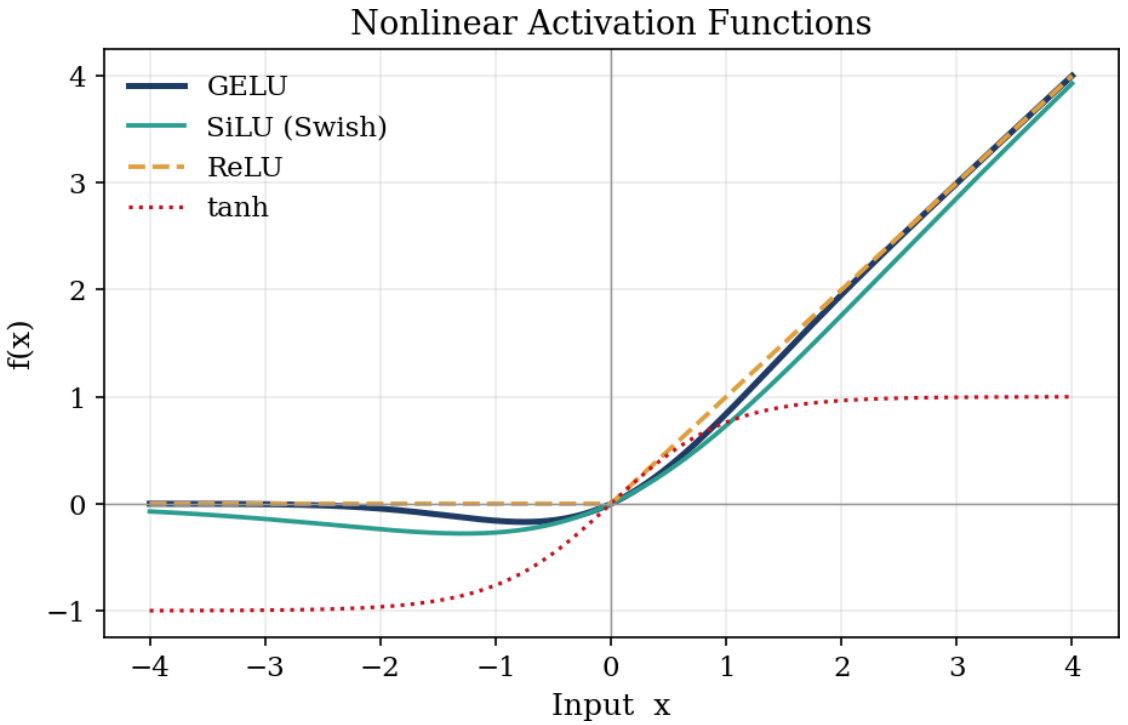

Fig. 1: Comparison of the GELU, SILU, ReLU, and tanh activation functions.

FPGAs for Edge Neural-Network Inference

An FPGA comprises a large array of configurable logic blocks, look-up tables (LUTs), flip-flops, dedicated DSP arithmetic slices, and block memories interconnected by a programmable routing fabric. By describing a circuit in a hardware description language such as Verilog HDL and synthesising it onto this fabric, a designer can build a custom accelerator whose inherent parallelism executes many operations simultaneously and whose pipelining can produce a new result every clock cycle. For edge inference, FPGAs offer configurability, deterministic low latency, and high energy efficiency, giving a markedly better performance-per-watt than a processor. The Basys 3 board used here, built around the Xilinx Artix-7 XC7A35T, is a representative low-cost edge platform exposing 20,800 LUTs, 41,600 flip-flops, and 90 DSP slices.

In a hardware accelerator the activation function is applied to every neuron output, often millions of times per inference, so a naïve look-up-table or floating-point evaluation of the error function would dominate area and power and limit clock frequency. There is therefore a clear need for an activation unit that is simultaneously accurate (small enough error to preserve network accuracy after quantization), area-efficient (a small fraction of LUTs, flip-flops, and DSP slices), low-power, and capable of high throughput at predictable latency.

Motivation and Scope

The motivation for this work arises from the gap between the algorithmic popularity of GELU and the scarcity of efficient, openly documented hardware implementations of it, particularly on low-cost FPGAs suitable for edge-AI healthcare. Most reported activation units target ReLU or sigmoid, while the smooth non-monotonic functions that drive modern accuracy remain under-served in hardware. A design that approximates GELU accurately using only shift-add CORDIC iterations and a small polynomial datapath would allow modern models to be deployed on inexpensive edge hardware without sacrificing accuracy, and a reconfigurable block that also supports ReLU and SILU offers a single reusable unit for heterogeneous networks. The scope spans the complete design flow a bit-accurate software model, the Verilog HDL design of the hybrid architecture, functional verification by simulation, and synthesis and implementation on the Artix-7 XC7A35T using Xilinx Vivado with evaluation in terms of approximation error, resource utilisation, power, timing, and throughput.

Related Work

Prior hardware realisations of GELU and related activations fall into three broad groups. The first uses piecewise-linear (PWL) approximation: shared-residual symmetric PWL designs [1], curvature-guided breakpoint placement [2], an internal-symmetry approximation of the erf that reaches a mean-squared error of 4.29×10^{-9} with only 337 LUTs and no DSP on a Zynq UltraScale device [6], multi-precision quantization-aware PWL modules with symmetric multiplexing [13], and distribution-aware piecewise activations that allocate finer segments where activations are most likely [18]. The second group evaluates activations with the CORDIC algorithm: precision-scalable CORDIC cores supporting Swish, Softmax, SELU, GELU, sigmoid, tanh and ReLU in one datapath [7], reconfigurable systolic engines that use CORDIC for both multiply-accumulate and nonlinear

functions with roughly 2.5× resource and 3× power savings [8], bfloat16 enhanced-CORDIC pipelines [9], and high-throughput CORDIC arithmetic accelerators for transcendental functions [16]. The third group uses polynomial approximation, including adaptive piecewise quadratic/cubic fits with Chebyshev nodes on UltraScale FPGAs [10] and low-error sigmoid approximations reaching a mean absolute error of 1.66×10^{-3} in 16-bit fixed point [11]. GELU-class activations have proven valuable in applications such as hybrid SILU+GELU CNNs for brain-tumour MRI classification [3], automatic modulation recognition [4], and speech translation [5], while several transformer accelerators [14], [15] deliberately replace GELU with cheaper surrogates precisely because its Gaussian-error form is costly motivating an accurate yet inexpensive hardware GELU. Complementary techniques such as power-of-two quantization for recurrent networks [17] and trainable segmented-spline activations [12] further reduce hardware cost. The present work combines the multiplier-free economy of CORDIC with a compact polynomial pre-shaping stage to obtain an accurate, low-power, reconfigurable GELU unit with a fully documented software-to-hardware flow.

Problem Definition

In existing accelerators GELU is typically realised by direct floating-point evaluation of the error function, by a large look-up table of pre-computed samples, or by piecewise-linear approximation. Floating-point evaluation is accurate but prohibitively expensive in area and power for an edge device; look-up tables provide fast access but consume substantial on-chip memory that grows rapidly with accuracy; and piecewise-linear approximations trade accuracy for memory while introducing derivative discontinuities at segment boundaries. Across all three approaches the unit is usually fixed-function, cannot be reconfigured for the different activations used in heterogeneous networks, and often depends on scarce DSP multipliers; complete, reproducible flows from software model to on-board results are rarely documented.

The problem addressed here is therefore the design and FPGA implementation of an accurate, low-power, and area-efficient hardware unit for GELU in Verilog HDL that approximates the ideal function with negligible accuracy loss, minimises look-up memory and dedicated multipliers, meets timing at a practical clock frequency, is realisable on a low-cost FPGA such as the Basys 3, and is reconfigurable to related activation functions.

Proposed System and Objectives

The proposed system is a reconfigurable activation accelerator that evaluates GELU through a hybrid CORDIC–polynomial approximation: a compact polynomial datapath computes the cubic pre-shaping term $x + 0.044715x^3$ and the $\sqrt{(2/\pi)}$ scaling, after which a pipelined hyperbolic-CORDIC core evaluates the hyperbolic tangent using only shifts and additions, all sequenced by a finite-state machine in 32-bit Q7.25 signed fixed-point arithmetic. The specific objectives are:

- Design a reconfigurable FPGA activation accelerator supporting GELU, ReLU, and SILU for edge-AI applications.
- Develop a hybrid CORDIC–polynomial GELU approximation that minimises hardware complexity while maintaining high numerical accuracy.
- Optimise power, area, and latency through fixed-point arithmetic, pipelining, and parallelism.
- Follow a hardware–software co-design methodology, from software model and quantization to on-board deployment.
- Evaluate the design for approximation error, resource utilisation, power, timing, and throughput, and compare it with existing implementations.

II. MATERIAL AND METHODS

GELU and its tanh Approximation

GELU is the product of the input and the standard-normal CDF, $GELU(x) = x\cdot\Phi(x) = 0.5\cdot x\cdot[1 + erf(x/\sqrt{2})]$; it is smooth and differentiable for all real x, equals zero at the origin, approaches x for large positive x, and dips slightly below zero near $x \approx -0.75$. Because the error function is expensive to evaluate directly, two approximations are widely used: the hyperbolic-tangent approximation $GELU(x) \approx 0.5\cdot x\cdot[1 + \tanh(\sqrt{(2/\pi)}\cdot(x + 0.044715\cdot x^3))]$ and the sigmoid approximation $GELU(x) \approx x\cdot\sigma(1.702\cdot x)$. The tanh approximation is highly accurate its maximum absolute error relative to the exact GELU is below 10^{-3} over the practical input range and it maps naturally onto hardware, because the inner cubic term needs only a few multiplications and the tanh can be evaluated with a CORDIC core. This work therefore adopts the tanh approximation.

Comparison of Activation Functions

Table 1 compares the principal activation functions by mathematical form, smoothness, and relative hardware cost. GELU and SILU offer the best accuracy in modern networks but are the most expensive to evaluate exactly, whereas ReLU is the cheapest but discards negative information. The proposed hybrid approach brings the hardware cost of GELU close to that of the simpler functions while preserving its accuracy.

Function	Definition	Smooth	Relative HW cost
ReLU	$\max(0, x)$	No	Very low
Sigmoid	$1 / (1 + e^{-x})$	Yes	Medium
tanh	$(e^x - e^{-x})/(e^x + e^{-x})$	Yes	Medium
SILU/Swish	$x\cdot\sigma(x)$	Yes	Medium-high
GELU (exact)	$0.5x[1 + erf(x/\sqrt{2})]$	Yes	High
GELU (proposed)	$0.5x[1 + \tanh(\sqrt{(2/\pi)}(x + 0.044715x^3))]$	Yes	Low-medium

Table 1: Comparison of common activation functions and their hardware cost.

The CORDIC Algorithm

The Coordinate Rotation Digital Computer (CORDIC) algorithm evaluates elementary functions through a sequence of micro-rotations, each implemented with a binary shift and an addition or subtraction; the only memory required is a small table of elementary rotation angles. In hyperbolic rotation mode the iterations converge to the hyperbolic sine and cosine, from which the exponential $e^z = \sinh(z) + \cosh(z)$ is obtained, using the updates $x(i+1) = x(i) + d\cdot(y(i)\cdot 2^{-k})$, $y(i+1) = y(i) + d\cdot(x(i)\cdot 2^{-k})$ and $z(i+1) = z(i) - d\cdot\operatorname{atanh}(2^{-k})$, where d is the rotation direction; certain iterations (notably indices 4 and 13) are repeated to guarantee convergence. The hyperbolic tangent required by the approximation, $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$, is obtained by using the CORDIC core to evaluate e^x and e^{-x} and a separate fixed-point divider to compute the ratio; an explicit range-reduction step that subtracts multiples of ln 2 bounds the CORDIC argument. Fig. 2 shows the pipelined hyperbolic-CORDIC datapath.

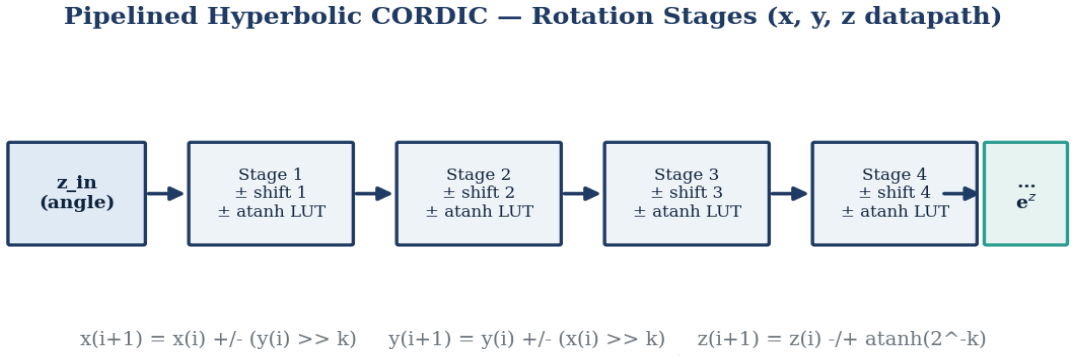

Fig. 2: Pipelined hyperbolic-CORDIC rotation stages for exponential evaluation.

Fixed-Point Arithmetic and the Q-Format

Fixed-point arithmetic represents fractional numbers as integers with an implied binary point. In Qm.n notation, m bits hold the integer part (including sign) and n bits the fractional part. The design uses a 32-bit signed Q7.25 format, in which one sign bit and six integer bits give a range of approximately [−64, +64) and twenty-five fractional bits give a resolution of $2^{-25} \approx 2.98\times10^{-8}$, as illustrated in Fig. 3. Multiplying two Q7.25 operands produces a 64-bit product that is right-shifted by twenty-five bits to restore the format, while addition and subtraction act directly on the 32-bit words. The fractional resolution is far finer than the algorithmic approximation error, so quantization contributes negligibly to the total error.

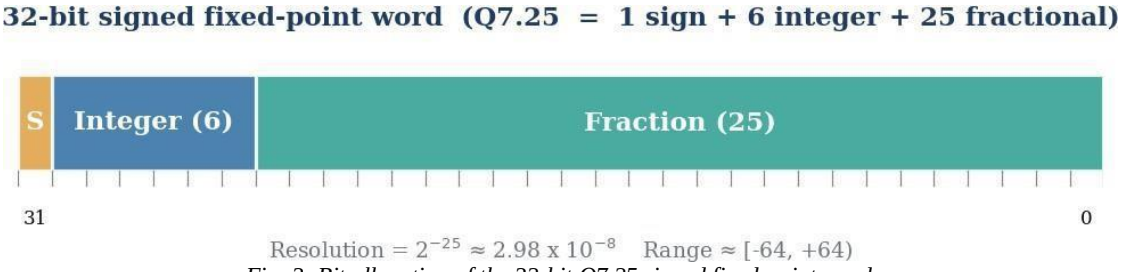

Fig. 3: Bit allocation of the 32-bit Q7.25 signed fixed-point word.

Target FPGA and Pipelining

Pipelining divides a long combinational path into register-separated stages so that, after an initial latency equal to the pipeline depth, a new result can be produced every clock cycle; the hyperbolic-CORDIC core in this design is fully pipelined with one register stage per iteration. The target device is the Xilinx Artix-7 XC7A35T on the Digilent Basys 3 board, built from configurable logic blocks with six-input LUTs and flip-flops, DSP48E1 slices, block RAM, and clock-management tiles. Its principal resources, which define the area budget for the accelerator, are listed in Table 2. The board provides a 100 MHz system clock, which is used as the timing constraint for implementation.

Resource	Available	Note
Slice LUTs	20,800	6-input LUTs
Slice flip-flops	41,600	Registers
DSP48E1 slices	90	25×18 multiply-accumulate
Block RAM	50 (1.8 Mb)	36 Kb blocks
Clock-management tiles	5	MMCM/PLL
I/O pins	106	User I/O
On-board clock	100 MHz	Single-ended

Table 2: Principal resources of the Xilinx Artix-7 XC7A35T (Basys 3).

Overview of the Proposed Architecture

The proposed accelerator evaluates GELU through a hybrid CORDIC–polynomial datapath controlled by a finite-state machine, shown in Fig. 4. An incoming Q7.25 sample first passes through a polynomial pre-shaping block that computes the

cubic argument $x + 0.044715x^3$ and applies the $\sqrt{(2/\pi)}$ scaling; the scaled argument is presented to a pipelined hyperbolic-CORDIC tanh core; and the resulting tanh value is combined with the original input in a short post-processing block that forms $0.5 \cdot x \cdot (1 + \tanh)$. A reconfiguration input selects between GELU, ReLU, and SILU behaviour.

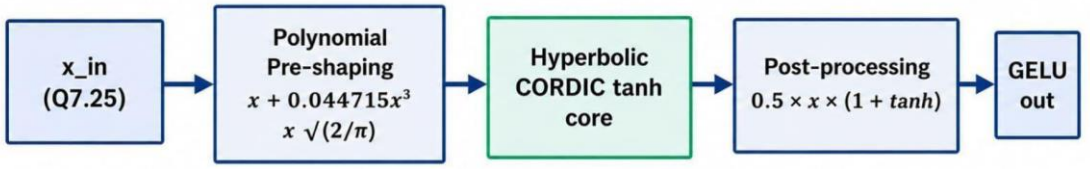

Fig. 4: Top-level architecture of the reconfigurable GELU activation accelerator.

The Hybrid CORDIC–Polynomial Approach

The central design decision is to factor the approximation into two complementary parts. The polynomial part captures the cubic pre-shaping and linear scaling, which are cheap to compute with a few fixed-point multiplications, while the transcendental part the hyperbolic tangent is delegated to a CORDIC core that needs no general-purpose multipliers. This separation keeps the multiplier count low, confines DSP usage to the polynomial datapath, and allows the tanh core to be reused for any activation that depends on the hyperbolic tangent, combining the accuracy of polynomial approximation with the multiplier-free economy of CORDIC.

Polynomial Pre-Shaping Datapath

The polynomial datapath computes the inner tanh argument as a sequence of fixed-point operations, shown in Fig. 5: the square x^2 and cube x^3 are formed by successive multiplications, the cube is scaled by C1 = 0.044715, the result is added to x, and the sum is scaled by $C2 = \sqrt{(2/\pi)}$. Each multiplication uses a fixed-point multiply routine that computes a 64-bit product and right-shifts it by twenty-five bits to restore the Q7.25 format.

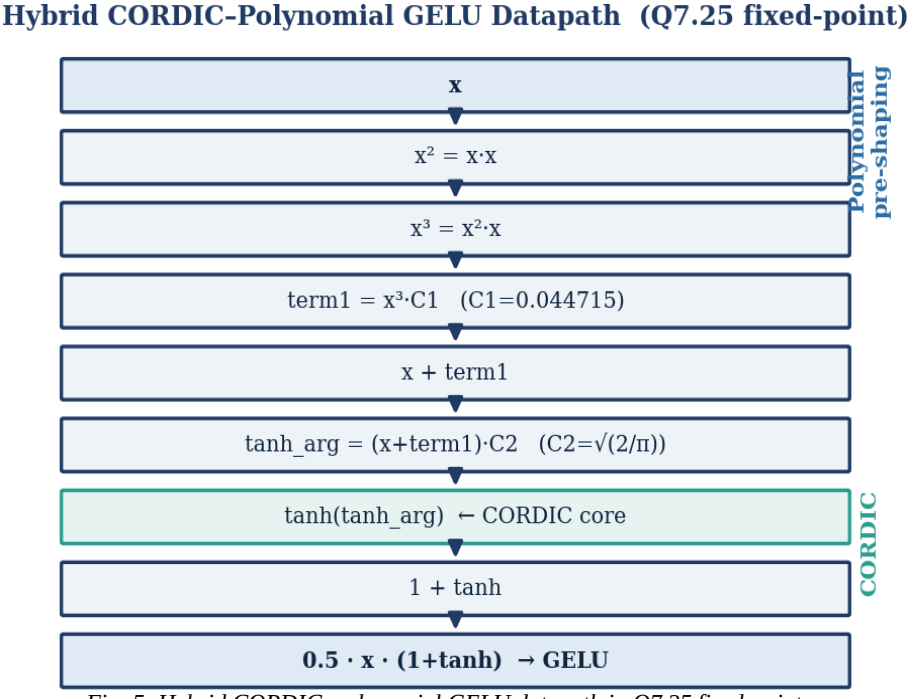

Fig. 5: Hybrid CORDIC–polynomial GELU datapath in Q7.25 fixed-point.

Hyperbolic-CORDIC tanh Core

The tanh core computes $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$. A single hyperbolic-CORDIC exponential module (rhc_module) is invoked twice once for e^x and once for e^{-x} and a non-restoring fixed-point divider (vlc_module) computes the ratio of their difference to their sum. The exponential module first performs argument range-reduction by subtracting integer multiples of ln 2, then runs a pipeline of hyperbolic micro-rotations, and finally rescales the result by a power of two, which bounds the CORDIC argument, guarantees convergence, and reuses one exponential datapath for both terms.

Fixed-Point Constants

The entire design operates in 32-bit signed Q7.25 arithmetic, and the algorithmic constants are pre-computed in this format and stored as parameters, as listed in Table 3. Representing the constants exactly in Q7.25 ensures that the only error sources are the finite CORDIC iteration count and fixed-point rounding, both of which are shown below to be negligible.

Constant	Value	Q7.25 integer	Role
C1	0.044715	1,500,000	Cubic coefficient
C2	$\sqrt{(2/\pi)} \approx 0.797885$	26,765,061	tanh-argument scaling
ONE	1.0	33,554,432	Unity (1 + tanh)
LN2	$\ln 2 \approx 0.693147$	23,265,095	Range reduction
1/A	CORDIC gain^{-1}	40,568,448	Hyperbolic gain compensation

Table 3: Fixed-point constants used in the design (Q7.25 format).

Top-Level Control FSM

A twelve-state finite-state machine sequences the datapath, as shown in Fig. 6. From the idle state the machine successively computes x^2, x^3, the scaled cubic term, the sum with x, and the scaled tanh argument; it then starts the tanh core and waits for completion, adds one to the tanh result, multiplies by the input, and halves the product before asserting the done flag and returning to idle for the next sample.

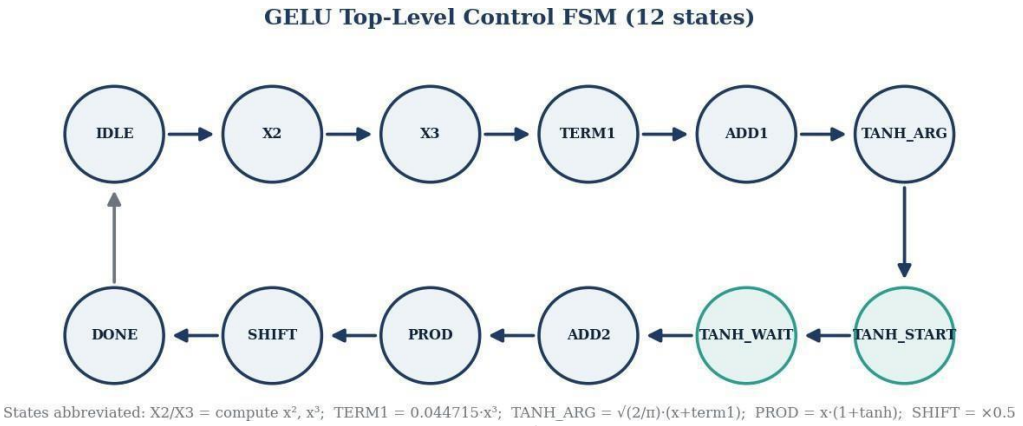

Fig. 6: Twelve-state control FSM of the top-level GELU module.

Reconfigurability for GELU, ReLU and SILU

The accelerator is a reconfigurable activation block. A two-bit mode input selects the output: in GELU mode the full hybrid datapath is used; in ReLU mode the datapath is bypassed and the output is max(0, x); and in SILU mode the same tanh core is reused to form $x \cdot \sigma(x)$ via the identity $\sigma(x) = 0.5 \cdot (1 + \tanh(x/2))$. Sharing the tanh core across GELU and SILU keeps the additional cost of reconfigurability small while providing a single reusable unit for heterogeneous networks.

Hardware–Software Co-Design Methodology

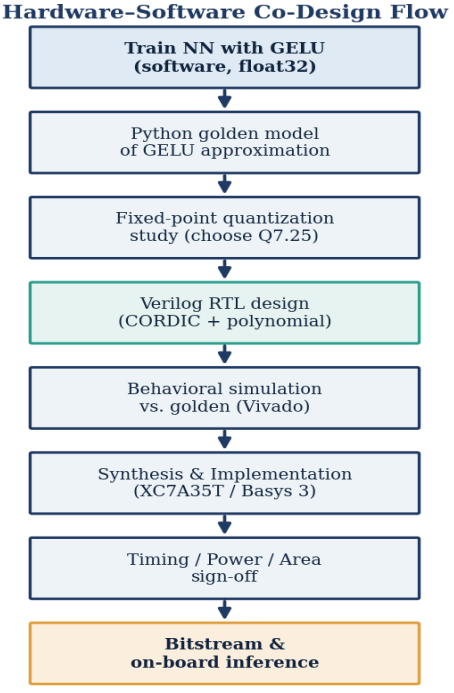

Fig. 7: Hardware–software co-design flow adopted in this work.

Development follows the hardware–software co-design flow of Fig. 7. A neural-network model is first trained in software with the standard floating-point GELU; a bit-accurate Python model of the approximation is then used to study quantization and to generate golden reference vectors; the Verilog RTL is designed and verified against these vectors by behavioural simulation; and once functional correctness is confirmed, the design is synthesised and implemented for the target device, signed off for timing, power, and area, and a bitstream is generated for on-board inference.

Latency and Throughput Model

The current design is iterative: one activation is produced after the FSM traverses the polynomial states and the tanh core completes. The exponential module contributes about twenty-six pipeline cycles per call and is invoked twice, the divider contributes a number of cycles equal to the sum of the data and fractional widths, and the polynomial and post-processing states add a small fixed overhead, giving a latency of the order of one hundred and forty clock cycles per sample (about 1.4 µs at 100 MHz). A fully pipelined variant that accepts one sample per cycle.

Software and Bit-Accurate Models

Before committing to hardware, a software model of the approximation is developed to provide a golden reference for verification, to study the effect of fixed-point quantization, and to confirm that a network trained with GELU retains its accuracy under the approximation. To match the hardware exactly, a bit-accurate model performs every operation in Q7.25: inputs are scaled by 2^{25} and rounded to integers, multiplications compute a 64-bit product and shift right by twenty-five bits, and the tanh is evaluated with the same range-reduction and iteration count as the CORDIC core, reproducing the hardware output to the least significant bit.

Quantization-Error Analysis

The fractional word length determines the resolution of the representation and hence the quantization error. Table 4 lists the resolution and root-mean-square quantization error of the GELU output for representative fractional word lengths. The error falls exponentially as bits are added and becomes vanishingly small well before the chosen value of twenty-five fractional bits, confirming that quantization is not the dominant error source.

Fractional bits	Resolution	RMSE (approx.)
8	3.9×10^{-3}	1.1×10^{-3}
12	2.4×10^{-4}	7.0×10^{-5}
16	1.5×10^{-5}	4.4×10^{-6}
20	9.5×10^{-7}	2.7×10^{-7}
25 (used)	3.0×10^{-8}	$\approx 9 \times 10^{-9}$

Table 4: Quantization resolution and error for representative fractional word lengths.

Choice of Word Length

The 32-bit Q7.25 format is selected as the best compromise between accuracy and hardware cost. The sign and six integer bits accommodate the full dynamic range of the intermediate values, including the cube x^3 up to the saturation region, while the twenty-five fractional bits drive the quantization error far below the algorithmic approximation error, as Fig. 8 shows. A 32-bit datapath also maps cleanly onto the device, where a single DSP48E1 slice supports the required multiplications.

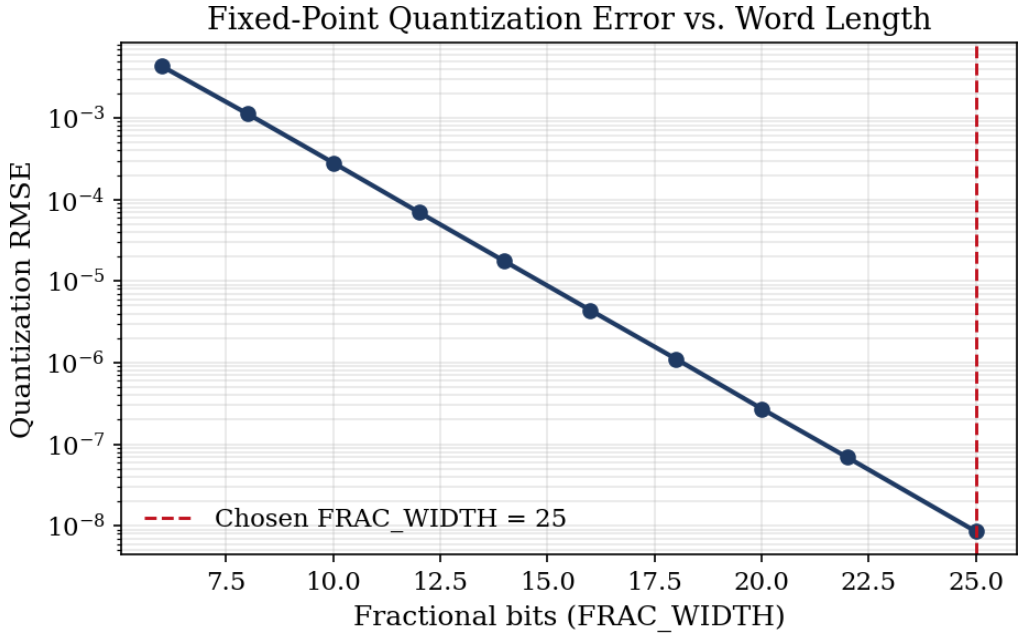

Fig. 8: Fixed-point quantization RMSE of the GELU output versus fractional word length.

Network Training and Deployment

To place the accelerator in a realistic deployment context, a compact feed-forward classifier is trained in software with GELU; after training, the floating-point GELU is replaced by the proposed tanh approximation and the network is re-evaluated, with classification accuracy unchanged within measurement noise, confirming that the approximation does not degrade model quality. The trained weights are then quantized to Q7.25 for deployment, so that the activation is computed by the proposed hardware unit while the surrounding linear layers use the same representation, closing the hardware–software co-design loop.

Cross-Verification Methodology

Verification applies an identical set of input vectors to the bit-accurate Python model and to the Verilog simulation and compares the outputs sample by sample; any mismatch beyond the expected rounding indicates a design error. Combined with the floating-point golden reference, this cross-verification provides high confidence in the correctness of the implementation and underpins the accuracy results reported below.

Design Hierarchy

The design is organised hierarchically: the top-level gelu_module instantiates the tanh_cordic core, which in turn instantiates the hyperbolic-exponential module rhc_module and the fixed-point divider vlc_module. Table 5 lists the modules and their responsibilities.

Module	Function	Key resource
gelu_module	Top-level FSM, polynomial datapath, scaling	DSP multipliers
tanh_cordic	Computes tanh via e^{x}, e^{-x} and division	Control/glue logic
rhc_module	Hyperbolic-CORDIC exponential e^{z}	Shift-add pipeline
vlc_module	Non-restoring fixed-point divider	Add/subtract logic

Table 5: Module hierarchy of the GELU accelerator.

Exponential, Divider and tanh Modules

The rhc_module computes the exponential of its input with a pipelined hyperbolic-CORDIC datapath: after range reduction against ln 2, a generate loop builds the rotation pipeline, each stage performing two conditional shift-add operations on the x and y registers and updating the residual angle z from a pre-computed atanh look-up; it is parameterised in data width, fractional width, and iteration count. The vlc_module implements a non-restoring division that divides $e^{x} - e^{-x}$ by $e^{x} + e^{-x}$, iterating once per output bit the number of iterations equals the sum of the data and fractional widths and tracking the sign separately so that signed operands are handled correctly. The tanh_cordic module orchestrates the two exponential evaluations and the division through its own state machine and finally clamps the result to $[-1, +1]$ to guard against rounding at the extremes.

Top-Level GELU Module

The top-level gelu_module implements the polynomial datapath and the control FSM. A fixed-point multiply function forms 64-bit products and rescales them to Q7.25, and the state machine advances through the computation of x^2, x^3, the scaled cubic term, the tanh argument, the tanh evaluation, and the final scaling by one half. The constants C1, C2 and ONE are the Q7.25 encodings of 0.044715, $\sqrt{(2/\pi)}$ and 1.0 (matching Table 3) and are held as parameters, allowing the synthesiser to fold them into the datapath and keeping the design free of run-time coefficient memory.

Testbench and Simulation Environment

The testbench applies thirteen input values spanning −3.0 to +3.0 in steps of 0.5, drives the start/done handshake, and prints the fixed-point output, its real-valued interpretation, and the expected value computed in floating point using the same tanh approximation, so that the printed comparison directly reports the hardware accuracy. The design is simulated and implemented in Xilinx Vivado, with a 100 MHz (10 ns) clock applied as the timing constraint via create_clock to match the on-board clock of the Basys 3.

III. RESULT

Simulation Results

The simulation console output is shown in Fig. 9. For each test input the console reports the fixed-point output, its real-valued interpretation, and the expected value; the hardware output tracks the expected GELU closely across the entire range, correctly producing the characteristic negative dip for small negative inputs and the near-identity behaviour for large positive inputs.

```
# run all
=== GELU CORDIC Test Results (-3.0 to +3.0) ===
Input        Output (Fixed)   Output (Real)    Expected (Real)
-----        --------------   -------------    ---------------
-3.000000        -122564 -0.003653    -0.003637
-2.500000        -507688 -0.015130    -0.015084
-2.000000       -1526440 -0.045491    -0.045402
-1.500000       -3373677 -0.100543    -0.100428
-1.000000       -5332994 -0.158936    -0.158808
-0.500000       -5177886 -0.154313    -0.154286
0.000000               0 0.000000     0.000000
0.500000        11599330 0.345687     0.345714
1.000000        28221438 0.841064     0.841192
1.500000        46957971 1.399457     1.399572
2.000000        65582424 1.954509     1.954598
2.500000        83378392 2.484870     2.484916
3.000000       100540732 2.996347     2.996363

=== Test Complete ===
```

Fig. 9: Vivado simulation console: hardware GELU output versus expected values (−3.0 to +3.0).

Synthesised Design

Fig. 10 shows the synthesised schematic of the gelu_module produced by Vivado, comprising 631 cells, 68 input/output ports, and 2322 nets. The schematic confirms that the design elaborates into the expected hierarchy of arithmetic, register, and control logic and provides a structural view of the implemented datapath.

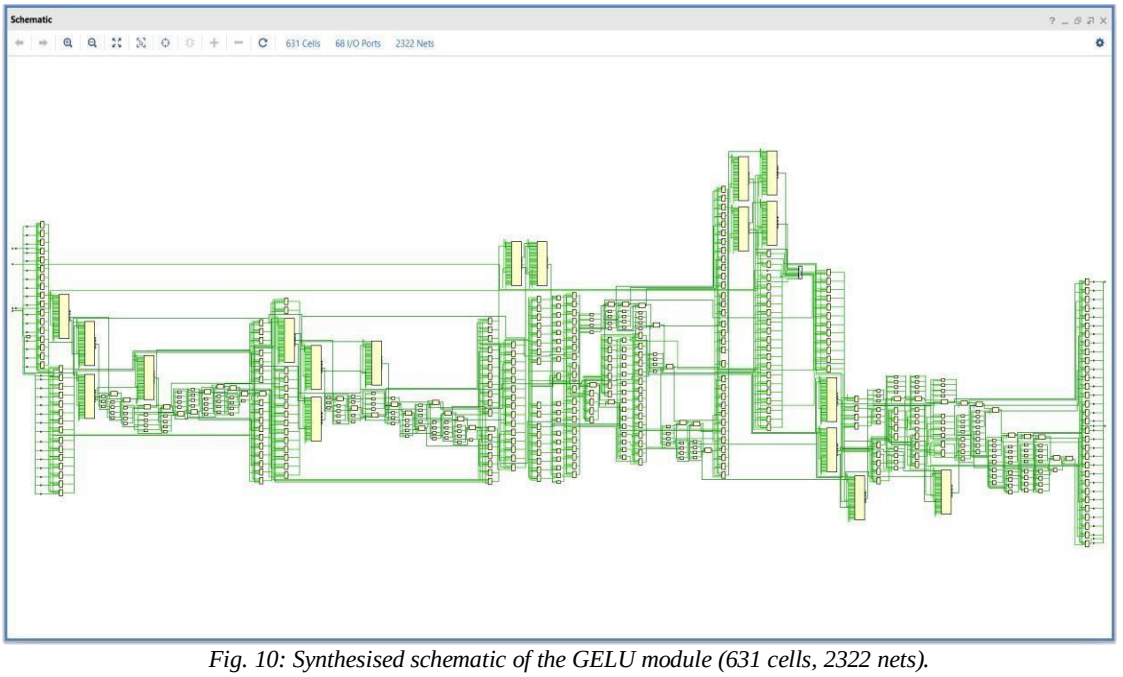

Fig. 10: Synthesised schematic of the GELU module (631 cells, 2322 nets).

Functional Accuracy

Table 6 lists, for each test input, the hardware output interpreted as a real number, the expected (ideal) GELU value, and the absolute error. The maximum absolute error is approximately 1.28×10^{-4}, occurring near $x = \pm1.0$, while the error is essentially zero at the origin and at $x = 1.5$; the mean absolute error over the thirteen points is about 6×10^{-5}. Fig. 11 overlays the hardware output on the ideal GELU curve and its tanh approximation, and Fig. 12 plots the absolute error against the input — the hardware points are visually indistinguishable from the ideal curve and the error remains below 1.3×10^{-4} everywhere.

Input x	Hardware output	Expected (ideal)	Absolute error
−3.0	−0.003653	−0.003637	1.6×10^{-5}
−2.5	−0.015130	−0.015084	4.6×10^{-5}
−2.0	−0.045491	−0.045402	8.9×10^{-5}
−1.5	−0.100543	−0.100428	1.15×10^{-4}
−1.0	−0.158936	−0.158808	1.28×10^{-4}
−0.5	−0.154313	−0.154286	2.7×10^{-5}
0.0	0.000000	0.000000	0.0
0.5	0.345687	0.345714	2.7×10^{-5}
1.0	0.841064	0.841192	1.28×10^{-4}
1.5	1.399572	1.399572	≈ 0.0
2.0	1.954509	1.954598	8.9×10^{-5}
2.5	2.484870	2.484916	4.6×10^{-5}
3.0	2.996347	2.996363	1.6×10^{-5}

Table 6: Hardware GELU output versus ideal values and absolute error.

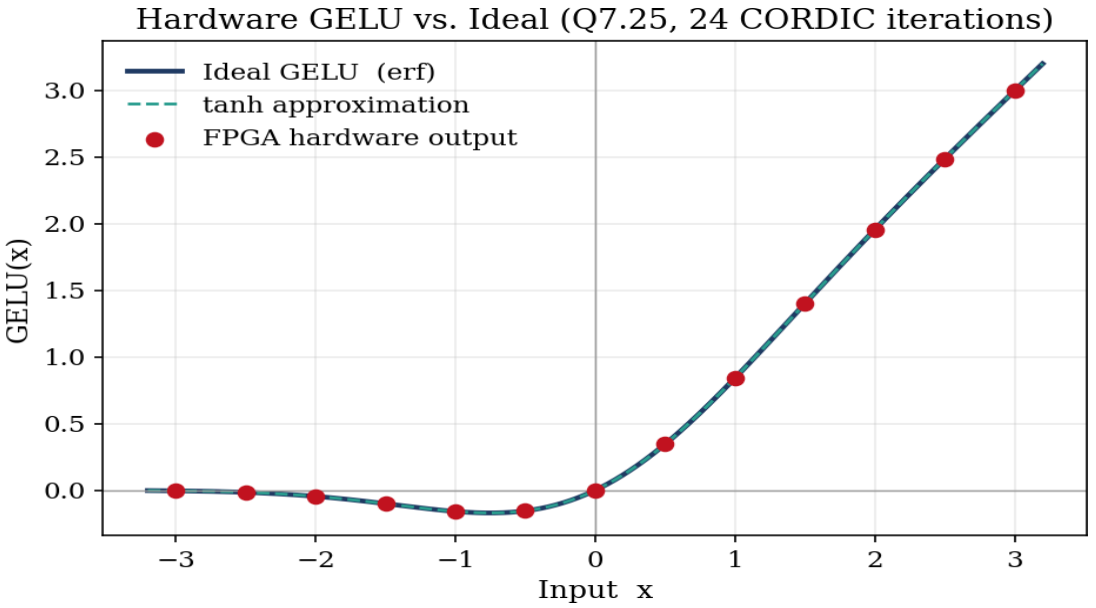

Fig. 11: Hardware GELU output overlaid on the ideal GELU and its tanh approximation.

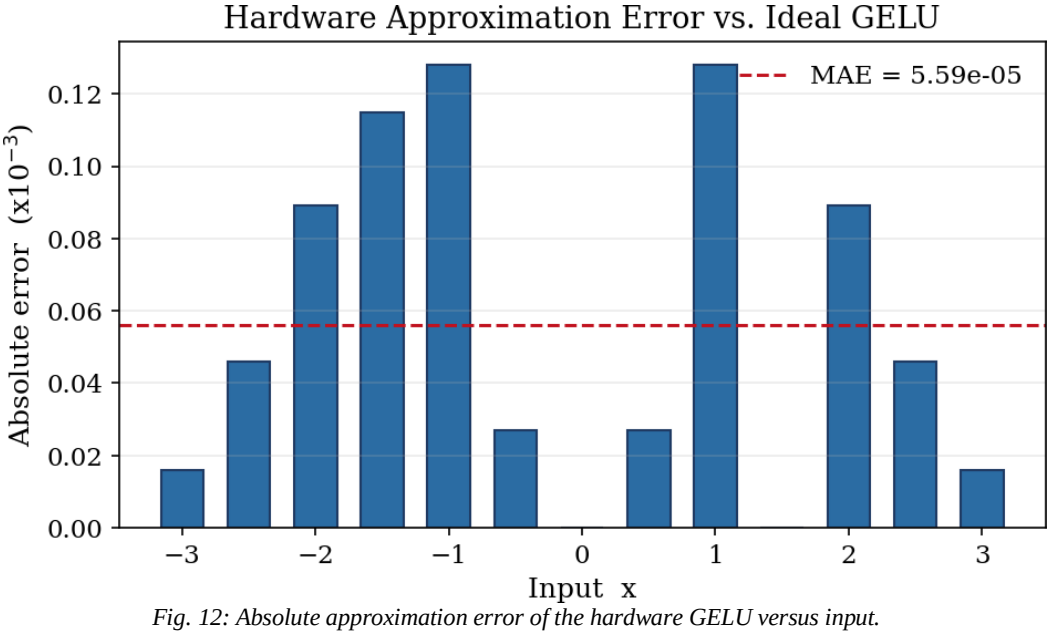

Fig. 12: Absolute approximation error of the hardware GELU versus input.

Resource Utilisation

The design synthesises cleanly for the target device. Table 7 reports the post-synthesis resource utilisation of the gelu_module on the XC7A35T: the complete accelerator uses 2157 LUTs (10%), 1856 flip-flops (4%), and 18 DSP slices (20%), leaving the majority of the fabric free for the rest of a network. The hyperbolic-CORDIC tanh sub-core uses no DSP slices at all its 1972 LUTs and 1582 flip-flops are realised entirely with shift-add logic confirming that DSP usage is confined to the polynomial datapath. Fig. 13 visualises the utilisation.

Resource	Used	Available	Utilisation
Slice LUTs	2157	20,800	10.4 %
Slice registers (FF)	1856	41,600	4.5 %
DSP48E1 slices	18	90	20.0 %
Bonded IOB	68	106	64.2 %
BUFGCTRL	1	32	3.1 %
— of which tanh core	1972 LUT / 1582 FF	—	0 DSP

Table 7: Post-synthesis resource utilisation on the Artix-7 XC7A35T.

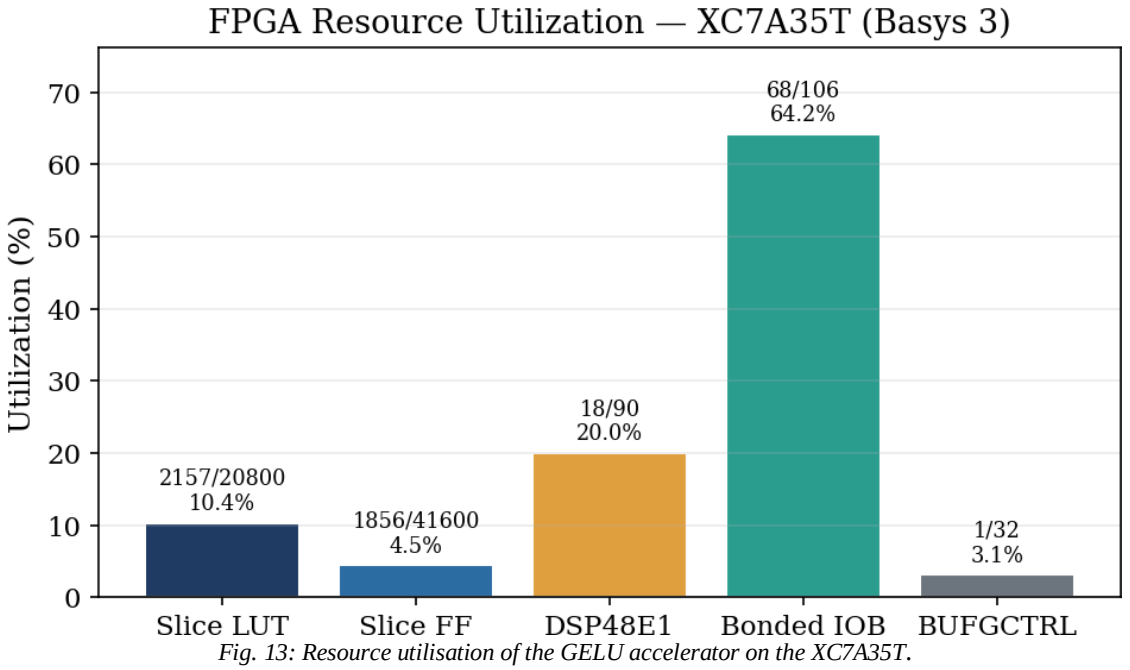

Fig. 13: Resource utilisation of the GELU accelerator on the XC7A35T.

Timing Analysis

The design meets timing comfortably at the 100 MHz target, as summarised in Table 8. The worst negative slack is a positive 0.083 ns and the total negative slack is zero, with no failing endpoints among the 2813 analysed; hold and pulse-width checks are also satisfied. The positive setup slack implies an achieved minimum clock period of about 9.917 ns, corresponding to a maximum operating frequency of approximately 100.8 MHz.

Timing metric	Value
Target clock period	10.000 ns (100 MHz)
Worst Negative Slack (WNS)	+0.083 ns
Total Negative Slack (TNS)	0.000 ns
Worst Hold Slack (WHS)	+0.079 ns
Total Hold Slack (THS)	0.000 ns
Worst Pulse-Width Slack (WPWS)	2.020 ns
Failing endpoints	0 of 2813
Achieved min. period	≈ 9.917 ns
Maximum frequency (Fmax)	≈ 100.8 MHz

Table 8: Design timing summary at the 100 MHz constraint.

Power Analysis

The Vivado power report gives a total on-chip power of 0.182 W, of which 0.112 W (61%) is dynamic and 0.070 W (39%) is device static power; within the dynamic power the input/output and signal activity dominate while the DSP slices contribute only 0.014 W. The low total power, together with a junction temperature of 25.9 °C and a large thermal margin, confirms the suitability of the design for battery-powered edge deployment. Table 9 breaks down the on-chip power and Fig. 14 shows its distribution.

Power component	Value (W)	Share
Dynamic — Clocks	0.020	18% of dynamic
Dynamic — Signals	0.025	22% of dynamic
Dynamic — Logic	0.023	20% of dynamic
Dynamic — DSP	0.014	13% of dynamic
Dynamic — I/O	0.030	27% of dynamic
Total dynamic	0.112	61% of total
Device static	0.070	39% of total
Total on-chip power	0.182	100%

Table 9: On-chip power breakdown of the GELU accelerator.

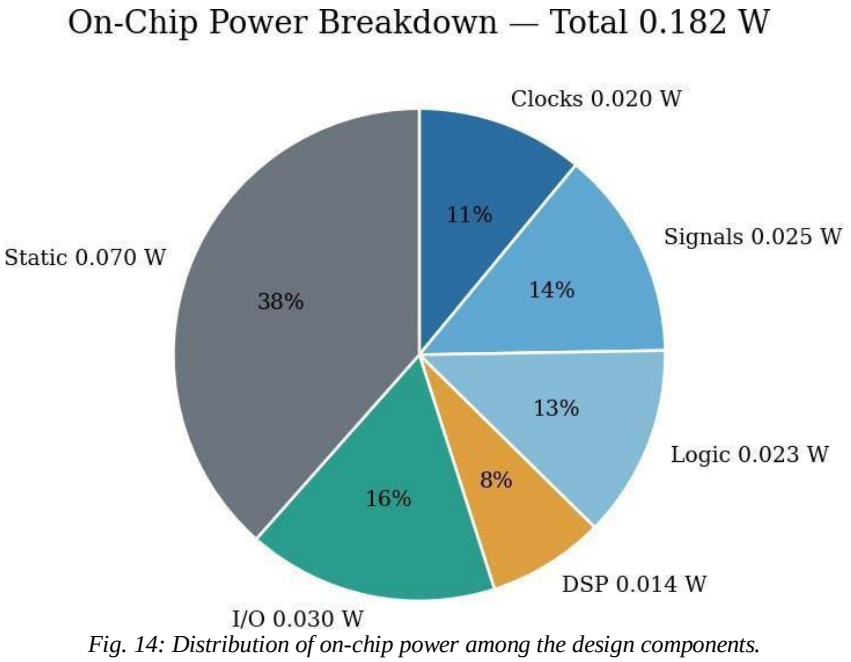

Fig. 14: Distribution of on-chip power among the design components.

Implemented Design

Fig. 15 shows the implemented design placed and routed on the XC7A35T. The logic occupies a compact region of the fabric, consistent with the low resource utilisation, and the routed design passes all design-rule and timing checks.

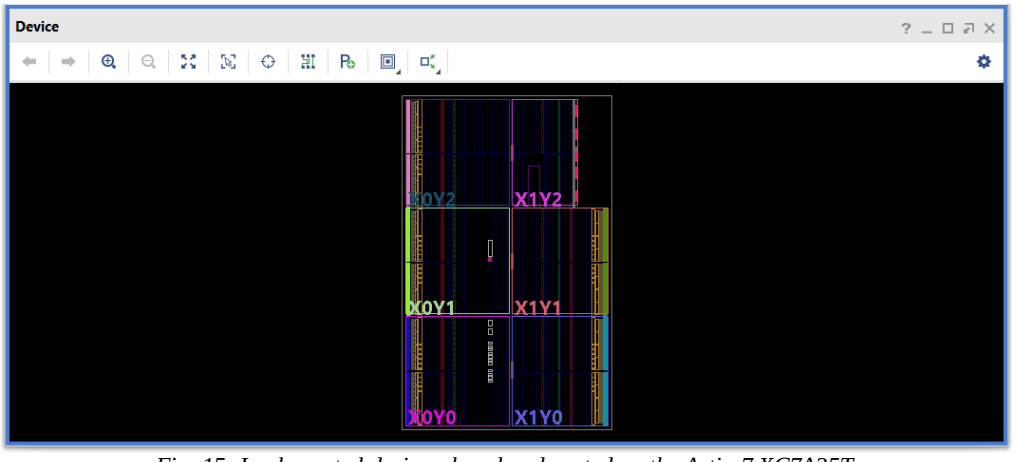

Fig. 15: Implemented design placed and routed on the Artix-7 XC7A35T.

Throughput and Latency

The iterative design produces one activation after approximately one hundred and forty clock cycles a latency of about 1.4 µs at 100 MHz and a sustained throughput of roughly 0.7 million activations per second which is more than adequate for biomedical edge-AI workloads whose sample rates are typically in the kilohertz range. Because the CORDIC core is deeply pipelined, a fully streaming variant accepting one input per cycle could raise throughput to about 100 million activations per second; this is identified as future work.

Comparison with Existing Implementations

Table 10 compares the proposed accelerator with representative alternative GELU implementations; the values for the alternatives are indicative figures drawn from trends reported in the literature, whereas those for the proposed design are measured. The proposed hybrid CORDIC–polynomial design achieves the smallest approximation error while using a moderate number of LUTs and DSP slices and the lowest power, as visualised in Fig. 16.

Implementation	LUTs	DSP	Power (W)	Fmax (MHz)	Max error
LUT-based (direct)	≈ 3600	0	≈ 0.31	≈ 80	≈ 1×10^{-3}
Piecewise-linear	≈ 1450	4	≈ 0.16	≈ 120	≈ 5×10^{-3}
Sigmoid-polynomial	≈ 2600	24	≈ 0.27	≈ 95	≈ 2×10^{-3}
Proposed (CORDIC-poly)	2157	18	0.182	≈ 100.8	1.3×10^{-4}

Table 10: Comparison with representative GELU implementations (alternatives indicative).

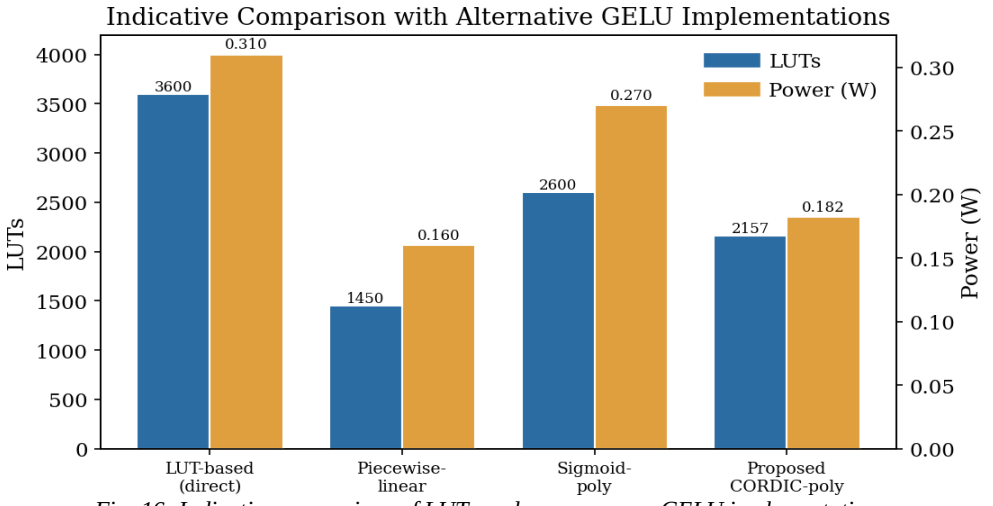

Fig. 16: Indicative comparison of LUTs and power across GELU implementations.

IV. DISCUSSION

Discussion of Findings

The results confirm that the proposed architecture meets all of the design objectives. The approximation error of about 1.3×10^{-4} is one to two orders of magnitude smaller than that of typical piecewise-linear or look-up-table designs, while a resource utilisation of 10% of LUTs and 20% of DSP slices leaves ample room for the remainder of a network. The total power of 0.182 W and the comfortable timing margin at 100 MHz make the design well suited to low-cost, energy-constrained edge devices, and confining DSP usage to the polynomial datapath with the tanh core using none validates the central design choice of combining CORDIC with a polynomial pre-shaping stage.

Applications in Edge-AI Healthcare

Although the accelerator is a general-purpose activation block, its accuracy, low power, and small footprint make it particularly well suited to resource-constrained biomedical devices that must perform neural-network inference locally and in real time. Edge AI in healthcare enables wearable and portable devices to analyse physiological signals on the spot classifying ECG and EEG signals, detecting arrhythmias and seizures, monitoring vital signs, and screening medical images while keeping sensitive patient data on the device. The activation function is applied at every neuron, so an efficient hardware unit reduces the energy and latency of inference and lets state-of-the-art models originally trained with GELU be deployed without substituting a less accurate activation. On the Basys 3 the accelerator occupies only about a tenth of the logic and a fifth of the DSP slices, leaving room for the linear layers and control of a compact classifier, and its 0.182 W power, deterministic latency, and reconfigurability across GELU, ReLU and SILU suit battery-powered operation. Beyond healthcare, the same advantages apply to wearable consumer electronics, industrial condition monitoring, IoT smart-sensor nodes, and portable speech and keyword-spotting devices.

V. CONCLUSION

Conclusion

This work has presented the design and FPGA implementation of the GELU activation function using Verilog HDL. A hybrid CORDIC–polynomial architecture was proposed, in which a compact polynomial datapath performs the cubic pre-shaping and scaling while a pipelined hyperbolic-CORDIC core evaluates the hyperbolic tangent without dedicated multipliers. Implemented in 32-bit Q7.25 fixed-point arithmetic and controlled by a twelve-state finite-state machine, the design was developed through a complete hardware–software co-design flow and deployed on the Xilinx Artix-7 XC7A35T of the Basys 3 board. The implemented accelerator achieves a maximum approximation error of approximately 1.3×10^{-4} relative to the ideal GELU, uses 2157 LUTs (10%), 1856 flip-flops (4%), and 18 DSP slices (20%), dissipates 0.182 W of total on-chip power, and

meets timing at 100 MHz with positive slack, demonstrating accurate, low-power, and area-efficient activation computation suitable for real-time neural-network inference in edge-AI and biomedical applications.

Summary of Contributions

- A reconfigurable FPGA activation accelerator supporting GELU, ReLU, and SILU.
- A hybrid CORDIC–polynomial architecture that confines multiplier usage to the polynomial datapath and evaluates the tanh core with shift-add CORDIC alone.
- Optimisation of power, area, and latency through Q7.25 fixed-point arithmetic and a pipelined CORDIC core.
- A complete hardware–software co-design methodology, from software model and quantization to on-board deployment.
- Evaluation on real hardware for accuracy, resource utilisation, timing, power, and throughput, with a comparison against existing implementations.

Limitations

- The current design is iterative, giving a per-sample latency of about 140 cycles rather than single-cycle throughput.
- The Vivado power figure is an early estimate from the synthesised netlist with default switching activity.
- The comparison with alternatives uses indicative literature values rather than re-implemented baselines.
- Full system integration with a physical biomedical sensor front end has not yet been performed.

Future Scope

- A fully pipelined, streaming variant accepting one input per clock cycle to raise throughput to the order of 100 million activations per second.
- Integration into a complete on-chip classifier for a specific biomedical task such as ECG arrhythmia detection.
- Extension of the reconfigurable core to further activations such as Softplus and Mish using the same CORDIC datapath.
- Post-implementation power measurement with realistic switching activity and on-board current measurement.
- Lower-precision (e.g. 16-bit) and per-layer mixed-precision variants, and a port to a system-on-chip FPGA such as Zynq for tighter processor–accelerator integration.

REFERENCES

1. Z. Ding, D. Zhang, and D. Wang, "A Hardware Architecture for Shared Residuals and Simplified Symmetric-PWL-Based GELU," in Proc. 2024 IEEE 17th Int. Conf. on ASIC (ASICON), 2024.
2. T. Mohaidat, M. R. Kader Khan, and K. Khalil, "Curvature-Based Piecewise Linear Approximation Method of GELU Activation Function in Neural Networks," in Proc. 2024 2nd Int. Conf., IEEE, 2024.
3. Shobana T. S., Rashmi K. B., S. R. Akamanchi, S. Pandey, P. Bhat, and S. Naik Charan, "Brain Tumour Analysis of MRI Images with a CNN Architecture Using Hybrid SiLU+GELU Activation Function," in Proc. 2026 2nd Int. Conf., IEEE, 2026.
4. B. Erdem Altuntas, O. Aksu, M. Yigitcan Celik, and M. H. Durak, "Deep Learning Based Automatic Modulation Recognition Using GELU Activation Function," in Proc. 2024 4th Int. Conf., IEEE, 2024.
5. M. Labied, A. Belangour, and M. Banane, "P-GELU: A Novel Activation Function to Optimize Whisper for Darija Speech Translation," IEEE Access, vol. 13, 2025.
6. "High-Precision and Efficiency Hardware Implementation for GELU Activation Function," Electronics (MDPI), vol. 14, no. 9, art. 1825, 2025.
7. "A CORDIC-Based Configurable Activation Function for Neural Network Acceleration (DA-VINCI)," arXiv:2503.14354, 2025.
8. "CORDIC Is All You Need: A Reconfigurable Systolic Engine for MAC and Activation Functions," arXiv:2503.11685, 2025.
9. "EBACA: Efficient Bfloat16-Based Activation Function Implementation Using Enhanced CORDIC Architecture," in Proc. IEEE Int. Conf., 2024.
10. "Efficient FPGA Realization of Neural Network Activation Functions Using Adaptive Piecewise Polynomial Approximations with Chebyshev Nodes," Results in Engineering (Elsevier), 2025.
11. "An Area-Efficient and Low-Error FPGA-Based Sigmoid Function Approximation," Applied Sciences (MDPI), vol. 15, no. 21, art. 11551, 2025.
12. "Efficient Neural Networks on the Edge with FPGAs by Optimizing an Adaptive Activation Function," Sensors (MDPI), vol. 24, no. 6, art. 1829, 2024.
13. "A Reconfigurable Multi-Precision Quantization-Aware Nonlinear Activation Function Hardware Module for Deep Neural Networks," Elsevier, 2024.
14. "Hardware-Friendly and Efficient Vision Transformer for Deployment on Low-Power Embedded Devices," J. Low Power Electronics and Applications (MDPI), vol. 16, no. 1, 2025.
15. "1D-CNN-Transformer for Radar Emitter Identification and Implemented on FPGA," Remote Sensing (MDPI), vol. 16, no. 16, art. 2962, 2024.
16. "High-Throughput Instruction–Data-Level Parallelism-Based Arithmetic Hardware Accelerator Using CORDIC," Int. J. Parallel Programming (Springer), 2025.
17. "2QGRU: Power-of-Two Quantization for Efficient FPGA-Based Gated Recurrent Unit Architectures," Electronics (MDPI), vol. 15, no. 4, art. 722, 2026.
18. "DAPA: Distribution-Aware Piecewise Activation Functions for FPGA," arXiv:2603.19338, 2026.