

GRAU: Generic Reconfigurable Activation Unit Design for Neural Network Hardware Accelerators

Yuhao Liu^{1,2,3} , *Student Member, IEEE*, Salim Ullah¹ , Akash Kumar¹ , *Senior Member, IEEE*

¹Ruhr University Bochum, Germany ²Dresden University of Technology, Germany

³Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI Dresden/Leipzig), Germany

Email: {yuhao.liu, salim.ullah, akash.kumar}@rub.de

Abstract—With the continuous growth of neural network scales, low-precision quantization is widely used in edge accelerators. Classic multi-threshold activation hardware requires 2^n thresholds for n -bit outputs, causing a rapid increase in hardware cost as precision increases. We propose a reconfigurable activation hardware, GRAU, based on piecewise linear fitting, where the segment slopes are approximated by powers of two. Our design requires only basic comparators and 1-bit right shifters, supporting mixed-precision quantization and nonlinear functions such as *SiLU*. Compared with multi-threshold activators, GRAU reduces LUT consumption by over 90%, achieving higher hardware efficiency, flexibility, and scalability. The best trade-off is usually achieved with 6–8 segments, while complex nonlinearities under aggressive low-cost settings may suffer larger accuracy degradation.

I. INTRODUCTION

Continuously growing sizes of state-of-the-art neural network models encourage researchers to explore different schemes to accelerate the network inference and improve the power efficiency by reducing memory consumption and computing power requirements. Therefore, quantization becomes one of the most widely applied methods in related works, especially by training the weights and activations in the network model as low-precision integers. Considering that the outputs of each neuron or kernel in *Quantized Neural Networks* (QNNs) should be the quantized integers following the selected precision, one re-quantization unit should be implemented after the activation unit to convert the outputs of the activation function to integers in the design of a QNN hardware accelerator.

A. Motivation

Considering the nonlinear activation function and re-quantization computation are expensive on hardware implementation, previous works have extensively explored the designs of quantized activation hardware for QNN accelerators. One of the widely used design paradigms is the *Multi-Threshold* (MT) activation unit, adopted in well-known designs such as *FINN* [1] and *FINN-R* [2]. By folding *Batch Normalization*, nonlinear activation, and re-quantization into a single comparator-based block, MT units replace expensive arithmetic operations with a set of fixed thresholds. However, this design paradigm faces three major limitations as QNNs evolve:

TABLE I: Comparison between Unified-Precision and Mixed-Precision QNN on MNIST [3]

	MLP			CNN		
	Full 1-bit	Mixed (baseline)	Full 8-bit	Full 1-bit	Mixed (baseline)	Full 8-bit
Accuracy/%	92.29	95.91	97.36	96.26	98.79	99.14
Loss/%	-3.62	0.00	1.45	-2.53	0.00	0.35
Memory/Bytes	7,376	9,984	59,008	29,848	55,712	238,784
Baseline Ratio	0.74	1.00	5.91	0.54	1.00	4.29

1) *Exponential hardware scaling with precision*: MT unit takes the integer outputs from *Multiply-Accumulators* (MAC) and compares them with $2^n - 1$ thresholds to produce n -bit outputs, such as 15 thresholds for 4-bit and 255 for 8-bit. If the MAC result exceeds m thresholds, the quantized activation output is an integer, m . This design paradigm suggests exponentially increasing hardware resource consumption, since following an increase in output precision, the number of thresholds grows exponentially.

2) *Inefficiency in mixed-precision quantization*: Mixed-precision quantization has recently emerged as a key trend in the design of lightweight AI on the edge. For instance, Liu et al. [3] compared unified-precision and mixed-precision QNNs using a small 4-layer MLP and CNN on MNIST [4]. Their results in Table I indicate that 1/2/4/8-bit mixed precision offers a trade-off between accuracy and memory usage compared to BNN and QNN, tolerating a slight accuracy loss to significantly reduce memory cost compared to QNN. However, the MT unit must implement the maximum number of thresholds required by the highest precision. For instance, in 1/2/4/8-bit mixed-precision quantization, the MT unit must implement 255 thresholds for the 8-bit precision, while only a small subset is used for lower precisions (for example, only one threshold is used for 1-bit). Serial reuse of a comparator can reduce hardware costs but significantly increases latency at higher precisions.

3) *Inability to represent non-monotonic activations*: Since MT outputs always increase as more thresholds are exceeded by inputs, the design inherently supports only monotonically increasing functions. Emerging nonlinear activations, such as *SiLU* [5], violate this constraint, making the MT unit incompatible with many practical QNN settings. The left plot in Figure 1 instantiates a correct quantization processing of a *Sigmoid* function with three thresholds for 2-bit output. However, the right plot shows an incompatible case of MT

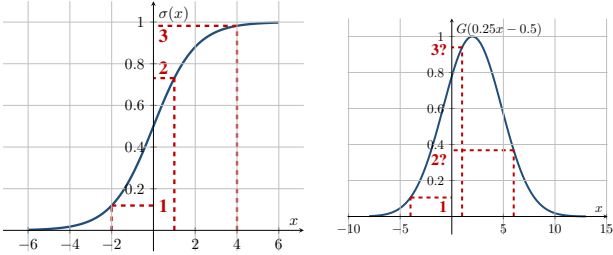


Fig. 1: Correct 2-bit quantization of *Multi-Threshold* unit (left) in *Sigmoid* and the mistake of *Multi-Threshold* unit in non-monotonically increasing function (right)

units where the expected output, 3, at the threshold of 0.9, is larger than the expected output, 2, at 6.0, violating the monotonically increasing condition. The quantized output at 6.0 should be 3 since it exceeds three thresholds, and the output at 0.9 should be 2 instead.

B. Related Works

Although the MT unit has been successfully adopted in various well-known works, considering the above-discussed shortcomings, we further surveyed other potential solutions introduced in prior works, such as *Piecewise Linearized Fitting* (PWL), *Second-Order Polynomial Fitting* (SOPF), and *Lookup Table* (LUT):

- PWLF and SOPF are more flexible than MT units to fit various nonlinear functions, such as: i) ML-PLAC [6] explored the Power-of-Two-based PWLF approximation for nonlinear functions, like *Tanh*, *Softsign*, 2^x , $\log_2(1+x)$, etc. ii) Zhang et al. [7], Li et al. [8], Tsmots et al. [9], and Nguyen et al. [10] explored the PWLF-based hardware design of *Sigmoid* activation functions. iii) Liu et al. [11] and Bouguezzi et al. [12] presented the PWLF-based implementation for *Tanh* and *TanhExp*. iv) Tsmots et al. and Bouguezzi et al. applied the SOPF-scheme for *Sigmoid* and *Tanh* approximation on hardware. However, these schemes are primarily focused on recovering the accuracy of the nonlinear function. For example, ML-PLAC [6] implements 24 and 42 segments for *Tanh(x)* and *Softsign(x)*, respectively, which consume more hardware resources. Furthermore, the *Batch Normalization* and *Output Re-quantization* before and after the nonlinear functions in the QNN accelerator design have not been folded with the activation function to reduce resource consumption. However, according to the theory of approximation computing, neural networks can tolerate computational errors, such as approximations in multiplication or in activation functions. Therefore, allocating more hardware resources to recover higher accuracy of nonlinear activations may not be the most hardware-efficient choice for resource-constrained QNN accelerators.
- Compared to the *Multi-Threshold* activation method, the Lookup-Table-based quantized activation unit is also

TABLE II: Comparison between SOPF, PWLF, MT, LUT, and GRAU for Quantized Activation Unit

Features	Hardware Friendly	Runtime Reconfigurable	Non-monotonically Increasing Function	Adaptive for Mixed-Precision	End-to-End MAC to Quant
SOPF	Low	No	Yes	Yes	No
PWLF	Low	No	Yes	Yes	No
MT	High	Yes	No	No	Yes
LUT	High	Limited	Yes	No	Yes
GRAU	High	Yes	Yes	Yes	Yes

hardware-friendly, as shown in the works of Piazza et al. [13], Pogiri et al. [14], and Kumar Meher et al. [15]. In principle, by rewriting the lookup contents stored in flip-flops or BRAM, the activation function can be reconfigured at runtime. However, such schemes are less suitable for a generic runtime-reconfigurable activation unit like the MT activation unit. For a direct lookup implementation, the storage cost grows exponentially with the input address width and linearly with the output bit-width. In our experiments, the integer MAC outputs of 8-bit quantized ResNet-18 [16] on ImageNet [17] can reach approximately $[-10^5, 10^5]$, which already corresponds to an address space close to 18 bits. Mapping such inputs directly to 8-bit outputs (e.g., $[-128, 127]$) would require a prohibitively large lookup table. Furthermore, in mixed-precision designs, different precision modes and activation functions require different lookup contents, implying either multiple tables or runtime table rewriting, which further increases memory overhead and control complexity.

C. Contribution

As summarized in Table II, comparing the four activation hardware design paradigms discussed above, we noticed that the unified, hardware-efficient, and reconfigurable design, which can simultaneously support multi-function, mixed-precision, and non-monotonic activations, for integer-aware QNN accelerators, is still not well supported by existing works. Therefore, we propose a novel **Generic Reconfigurable Activation Unit** (GRAU) for low-precision quantized, integer-based QNN hardware accelerator designs with flexible support of multi-activation function and mixed-precision quantization. Our work does not attempt to implement an approximate hardware generation framework for specific nonlinear functions, like the previous works, such as ML-PLAC [6]. Our design proposed one reusable end-to-end activation processing unit (from MAC output to quantized activated output). The main contributions of this work are as follows:

- *Hardware Friendly and Runtime Reconfiguration*: We propose a reusable generic activation unit based on PWLF with *Power-of-Two* (PoT) and *Additive Power-of-Two* (APoT) slope approximation. The design uses only comparators and a 1-bit shifter pipeline and can be reconfigured at runtime by updating a small set of break-point and shift-encoding registers. While PWLF, PoT, and APoT techniques have been individually explored

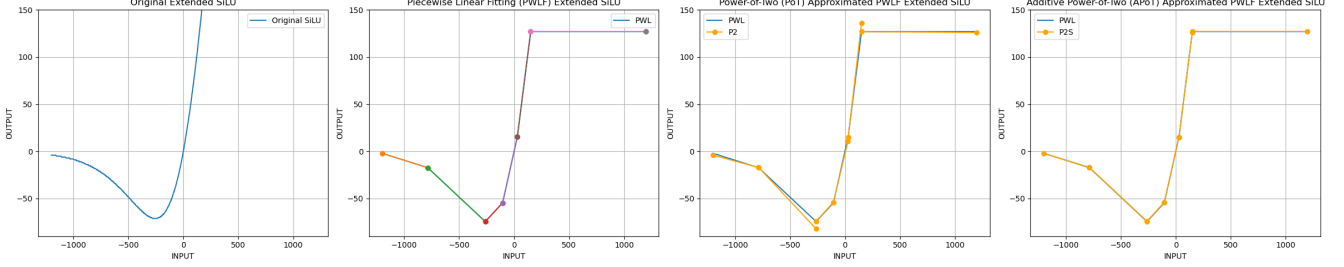


Fig. 2: Comparing the original nonlinear function, PWLF approximated function, PoT approximated PWLF function, and APoT approximated PWLF function

in prior work, such as [6], no existing design integrates these concepts into an approximate, reusable, unified activation processing unit that folds *Batch Normalization* and *Output ReQuantization*, with runtime configurability and mixed-precision integer support.

- **Flexible Function Support and Adaptation for Mixed-Precision Quantization:** GRAU supports multiple nonlinear and non-monotonic activations (e.g., *ReLU*, *Sigmoid*, *SiLU*) and easily adapts to different quantization precisions through lightweight reconfiguration of breakpoints and shifts.
- **Approximation Experiments:** We developed a fast greedy piecewise linear fitting algorithm to replace the nonlinear activation folded with batch normalization and output re-quantization in Quantized Neural Networks (QNN). We evaluate PoT/APoT approximations on CIFAR-10 and ImageNet with VGG16 and ResNet18 models under 4/8-bit unified-precision and 2/4/8-bit mixed-precision settings with various activation functions like ReLU, Sigmoid, and SiLU. The results suggest the feasibility of our approach to hardware: For many ReLU-dominant and selected APoT/PWLF settings, the accuracy drop is within $1 \sim 3\%$, while more complex cases, such as Sigmoid/SiLU under low segment counts or mixed precision, may incur larger degradation. The greedy piecewise linear fitting algorithm, integrated with PyTorch, has been open-sourced to facilitate easier reproduction of our work.¹
- **Hardware Implementation:** We implement both pipelined and serialized GRAU variants. The pipelined design achieves higher throughput, while the serialized version provides lower cost and greater configurability.
- **Resource Report:** Based on the synthesis and implementation in Vivado, the results show that our GRAU hardware reduces LUT usage by over 90% compared with *Multi-Threshold* (MT) units, achieving higher frequency, lower *Area-Delay-Product* (ADP), and lower *Power-Delay-Product* (PDP), which demonstrates promising hardware and power efficiency. Our implemented GRAU

TABLE III: Comparing the Accuracy of Original QNN and PWLF/PoT-PWLF/APoT-PWLF Approximated Models based on *pwlf* Library

Model Dataset Activation	SFC MNIST			CNV CIFAR-10		
	ReLU	Sigmoid	SiLU	ReLU	Sigmoid	SiLU
Original	97.73%	97.82%	97.55%	78.65%	76.81%	77.81%
PWLF	97.73%	97.82%	97.52%	78.24%	73.97%	78.21%
PoT-PWLF	97.76%	97.82%	88.14%	77.56%	73.69%	67.11%
APoT-PWLF	97.74%	97.82%	88.39%	77.52%	73.68%	65.22%

variants achieve up to 250 MHz in our Vivado post-implementation results, compared with 200 MHz for the pipelined MT baseline.

Although GRAU can be integrated into any QNN accelerator on FPGA/ASIC, the goal of this work is not to design a full accelerator architecture. Instead, we aim to establish a unified activation hardware design that enables runtime reconfiguration, multi-function support, and mixed-precision activation with significantly reduced overhead. Since quantized activation functions are present in every layer of modern QNNs, improving each activation unit directly scales across the entire accelerator.

D. Organization

This manuscript is structured in the following way: **Section II** discusses how to convert the original QNN models to the PoT and APoT approximated models for GRAU hardware and the hardware designs of GRAU activation units in this work. **Section III** shows the hardware evaluation results of the above-mentioned GRAU designs. **Section IV** discusses the further potential improvement and optimization of GRAU and concludes the contents of this paper.

II. IMPLEMENTATION

A. PoT and APoT Approximated Activation Functions for GRAU Hardware

Following the objectives outlined above, we transform the original nonlinear activation functions folded with BN and output re-quantization in QNNs into PWLF, PoT-PWLF, and APoT-PWLF representations compatible with the proposed GRAU architecture. To ensure efficient hardware realization, we additionally constrain the number of segments and limit

¹<https://github.com/liuyh-Horizon/GRAU-Approx-Toolkit>

the allowable range of power-of-two slopes when generating the PoT- and APoT-based approximations.

In Figure 2, we introduce the instances of PWLF approximated *Sigmoid* and *SiLU* functions and their PoT and APoT variants with six segments for 8-bit quantization. The first column in Figure 2 plots the original *SiLU* folded with BN and output re-quantization. The second and third columns are its PoT and APoT approximated functions. As shown in the plot of the original *SiLU*, its output is out of the allowed range of signed 8-bit integers, causing the clamp shown in the PWLF, PoT-PWLF, and APoT-PWLF plots of *SiLU*.

From PWLF to PoT- and APoT-PWLF, we adopt a three-step approximation:

- Considering that the inputs to our quantized activation unit in the QNN accelerator are the integer outputs from MACs, in PoT-PWLF and APoT-PWLF approximation, we adjust the breakpoints of segments to their nearest integers.
- We approximate the slope of each segment in PWLF functions to the nearest PoT and APoT value. For instance, if we define the allowable power range for PoT and APoT approximation as $[-10, -6]$, it means PoT approximated slopes can be $2^{-10}, 2^{-9}, \dots, 2^{-7}, 2^{-6}$, and APoT slopes can be the sum of one combination with any of these allowable PoT values, where one PoT value can only be used once in combination.
- We chose the left rounded breaking point of each segment to create a new linear function with approximated PoT and APoT slopes. Therefore, as shown in the third column in Figure 2, the PoT approximation has a small gap in the right end of each segment, since the approximated breaking points and slopes have a small bias for each segment compared with the original PWLF functions. APoT approximation also has this gap. However, it's more accurate than PoT. Therefore, the fourth column cannot clearly show the tiny gap in APoT.

Therefore, based on *Brevitas* [18] and the piecewise linear fitting algorithm, we created the PWLF, PoT-PWLF, and APoT-PWLF approximated QNN models on CIFAR-10 and ImageNet after the *Quantization-Aware Training* (QAT) as the following steps:

- Training the QNN models and recording the output ranges of each quantized *Fully Connected* (FC) layer, *QuantLinear*, and quantized *Convolution* (CONV) layers, *QuantConv2d* while training, which are the MAC outputs on hardware.
- For each layer, doubling the recorded MAC output range and averagely generating 1000 samples from the extended range as dummy input. Then, extracting the corresponding BN and quantized activation layers, such as *QuantReLU* and *QuantSigmoid* in *Brevitas*, from the trained QNN model and packaging them as black-boxes. Computing the output of these black boxes with the dummy input. Then, adopt these outputs to fit a piecewise linear function

based on the piecewise linear fitting algorithm, which will be the same form as the second column in Figure 2.

- Extracting the breaking points, slopes from the fitted piecewise linear function. Rounding the breaking points and approximating the slopes as PoT and APoT form with selected power ranges. Then, we created the PoT and APoT PWLF functions and replaced the BN and quantized activation layers in the original model to generate the PWLF, PoT-PWLF, and APoT-PWLF approximated models.
- Evaluating the accuracy of the original accurate model and PWLF, PoT-PWLF, and APoT-PWLF approximated models

As shown in Table III, in the early stage of this work, we evaluated the above-mentioned four different models with a *Small Fully Connected Network* (SFC) and *Simplified VGG-like Convolution Neural Network* (CNV) from FINN [1] based on the MNIST [4] and CIFAR-10 [19] datasets, utilizing the open-source *pwlf* [20] library to construct PWLF models and obtaining their PoT and APoT approximations. SFC has four FC layers, containing 256/256/256/10 neurons, respectively. We evaluated it with 4/8-bit QNN models and one 2/4/8-bit mixed-precision QNN model. CNV has three CONV blocks followed by three FC layers. Each block consists of two 3x3 CONV layers and one 2x2 max-pooling layer. The channel number of CONV layers in each CONV block is 64/128/256. The original FC layer in CNV models has 512/512/10 neurons. To reduce the time consumption of fitting the PWLF functions, we reduced it to 256/256/10. Since we use one bit to represent the usage of one power of two in the allowable range of $[-10, 6)$ and $[-24, 8)$, GRAU hardware would require 16- and 32-bit data for PoT and APoT setting encoding. As a result, the models that support these two ranges are named 16/32-bit PoT-PWLF and APoT-PWLF models.

As the results listed in III, PWLF, PoT-PWLF, and APoT-PWLF approximations generally introduce less than 1% accuracy loss across most experiments. More significant losses occur primarily in SiLU-based models: PoT-PWLF and APoT-PWLF approximations degrade accuracy by approximately 3% ~ 10% on 4-bit and mixed-precision SFC models, and similarly by around 10% on mixed-precision CNV models. In contrast, in Sigmoid-based CNV models, most degradation originates from PWLF itself, with PoT/APoT approximations introducing negligible additional loss.

However, for the *pwlf* library we used in the early stage of this work, there are some shortcomings:

- Based on the fitted functions in the above-mentioned two models, we found that: since *pwlf* is a continuous, floating-point-oriented library, it does not naturally adapt to the discrete integer-domain characteristics of MAC outputs in QNN. When two fitted breakpoints are close (e.g., 1.2 and 1.3), both round to the same integer, collapsing the corresponding linear segment and reducing expressive capacity.

Algorithm 1: Greedy Integer-Aware PWLF Breakpoint Selection

Input: sampled points (x_i, y_i) , target segments S , min gap g , min improvement ϵ

Output: breakpoint set $\mathcal{B}$

```
1 sort  $(x_i, y_i)$  by  $x$ ;  
2  $\mathcal{B} \leftarrow \emptyset$ ,  $\mathcal{S} \leftarrow \{(x_1, x_N)\}$ ;  
3 while  $|\mathcal{B}| < S - 1$  do  
4    $\mathcal{P} \leftarrow \emptyset$ ;  
5   foreach  $(a, b) \in \mathcal{S}$  do  
6     find  $x^* = \arg \max$  distance to chord  $(a, b)$ ;  
7      $\hat{x} \leftarrow \text{round}(x^*)$ ;  
8     if  $a < \hat{x} < b$  and  $\text{dist}(x^*) > \epsilon$  then  
9       add  $(\text{dist}, \hat{x}, (a, b))$  to  $\mathcal{P}$ ;  
10  if  $\mathcal{P} = \emptyset$  then  
11    break;  
12  select valid  $(\hat{x}, (a, b))$  from  $\mathcal{P}$  with largest distance;  
13   $\mathcal{B} \leftarrow \mathcal{B} \cup \{\hat{x}\}$ ;  
14  replace  $(a, b)$  with  $(a, \hat{x})$  and  $(\hat{x}, b)$  in  $\mathcal{S}$ ;  
15 return  $\mathcal{B}$ ;
```

- the *pwlf* library can only run on a CPU optimized with multi-core acceleration. We measured the time cost of piecewise linear fitting on our server (AMD EPYC 7513 32-Core Processor with 1TB memory). Fitting a single nonlinear function with 1,000 samples takes approximately four minutes. A model like ResNet-26 contains around 4,904 activation kernels, which would require approximately 13.6 days for PWLF fitting. In our experiment setup, each model structure is explored with nine variants; therefore, a full evaluation would take around four months, excluding training.

Consequently, in the early stage of this work, *pwlf* library limits our experiments to smaller networks while maintaining coverage across activation types and quantization precisions.

Therefore, to adopt our PoT and APoT PWLF approximation algorithm to larger networks and datasets, like ResNet-18 and ImageNet, we developed a fast greedy piecewise linear fitting algorithm as shown in algorithm 1. This algorithm employs a greedy integer-aware breakpoint selection strategy for PWLF. It starts from a single segment spanning the entire input range and iteratively refines the approximation. For each current segment, the point with the maximum vertical distance between the target nonlinear function and the chord connecting the segment endpoints is identified. This point is then rounded to the nearest integer to satisfy the integer breakpoint constraint required by hardware implementation. A candidate breakpoint is accepted only if it lies inside the segment, provides an improvement larger than the minimum threshold ϵ , and satisfies the minimum gap constraint g with respect to neighboring breakpoints. Among all valid candidates, the one with the largest distance is greedily selected to split the corresponding segment into two subsegments. The process continues until the

target number of segments is reached or no valid breakpoint can provide sufficient improvement.

Based on this algorithm, we evaluated it on the CIFAR-10 and ImageNet datasets using the VGG16 and ResNet18 models. For VGG16, we evaluated our PWLF approximation under 4/8-bit unified-precision quantization and 2/4/8-bit mixed-precision quantization. For ResNet18, we evaluated its 8-bit unified-precision quantization and 2/4/8-bit mixed-precision quantization models. For the mixed-precision quantization model, we adopted the same precision across all layers in one stage of VGG16 and ResNet18, with 8/4/2/4/8-bit precision for each stage and the fully connected layer. For VGG16 models, we replace all activations with ReLU/Sigmoid/SiLU. And for ResNet18 models, we evaluated the models with ReLU activation and ReLU/SiLU-mixed models (The layers in the fourth stage in the ResNet18 model adopted SiLU activation functions). Table IV and Table V listed our approximate results on CIFAR-10 and ImageNet dataset. We thoroughly explored our approximate schemes based on our PWLF function with 4/6/8 segments and continuous 4/8/16 exponents of 2^n .

Table IV and Table V show that the proposed folded-nonlinearity PWLF approximation is generally effective, and increasing the segment number from 4 to 6 or 8 usually improves accuracy. ReLU-based cases are the easiest to approximate and often remain close to the original QNN even with only 4 segments, e.g., the 8-bit ReLU model in Table IV drops only from 92.10% to 92.04%, while the 8-bit ReLU model in Table V still achieves 65.76%/86.31% Top-1/Top-5 versus the original 67.07%/87.23%. In contrast, SiLU and especially Sigmoid are more sensitive to approximation errors and benefit more from larger segment numbers, e.g., the 8-bit SiLU case in Table IV improves from 88.00% with 4 segments to 92.32% with 8 segments. Moreover, APoT-PWLF consistently outperforms PoT-PWLF, showing a better accuracy–hardware trade-off; for example, for mixed-precision SiLU with 8 segments in Table IV, APoT-PWLF reaches 89.26% while PoT-PWLF only achieves 86.89%. Overall, a 4-segment approximation is still feasible in some cases if a small accuracy loss is acceptable, 6 segments often provide a good balance between accuracy and complexity, and restricting the slope search to negative exponents is sufficient in practice since the folded function mainly compresses a large MAC output range into a low-bit quantized output range.

B. Hardware Implementation of GRAU

As shown in Figure 4, we implemented different 1-bit right shifter units pipeline for PoT-PWLF and APoT-PWLF functions:

For PoT-PWLF, the shifter unit in Figure 4 (a) loads the input data and, according to the 1-bit setting input, decides if it needs to pass the 1-bit right-shifted data or the original input data to the next shifter unit. For APoT-PWLF, the shifter unit in Figure 4 (b) loads the input data and a sum output from the prior shifter unit. After applying the 1-bit right shift to

TABLE IV: Comparing the Accuracy of Original QNN and PWLF/PoT-PWLF/APoT-PWLF Approximated Models based on Greedy-PWLF Algorithm on CIFAR-10 with VGG16 [21] Network

Experiment Result									
Precision Activation	ReLU	4-bit Sigmoid	SiLU	ReLU	8bit Sigmoid	SiLU	ReLU	Mixed-Precision Sigmoid	SiLU
Original	91.18%	87.81%	89.55%	92.10%	88.71%	92.35%	92.11%	88.39%	91.10%
Max. Segment	PWLF								
4	90.83%	86.37%	77.30%	92.04%	23.50%	88.00%	91.58%	64.30%	54.50%
6	91.00%	86.72%	84.88%	92.08%	76.79%	91.92%	91.78%	80.75%	88.20%
8	91.08%	85.99%	85.58%	92.10%	85.87%	92.32%	91.86%	74.87%	88.89%
Max. Segment	PoT-PWLF (Accuracy/Exponent Range)								
4	89.67%	54.28%	77.95%	90.93%	-	84.42%	90.54%	34.65%	56.87%
	($2^{-18} \sim 2^{-3}$)	($2^{-18} \sim 2^{-3}$)	($2^{-18} \sim 2^{-3}$)	($2^{-22} \sim 2^{-7}$)	-	($2^{-22} \sim 2^{-7}$)	($2^{-18} \sim 2^{-3}$)	($2^{-20} \sim 2^{-5}$)	($2^{-16} \sim 2^{-1}$)
	89.67%	48.21%	76.13%	90.93%	-	75.09%	88.24%	-	42.15%
	($2^{-10} \sim 2^{-3}$)	($2^{-10} \sim 2^{-3}$)	($2^{-12} \sim 2^{-5}$)	($2^{-14} \sim 2^{-7}$)	-	($2^{-16} \sim 2^{-9}$)	($2^{-14} \sim 2^{-7}$)	-	($2^{-14} \sim 2^{-7}$)
6	72.09%	28.56%	25.51%	77.42%	-	14.45%	12.42%	-	18.87%
	($2^{-8} \sim 2^{-5}$)	($2^{-10} \sim 2^{-7}$)	($2^{-8} \sim 2^{-5}$)	($2^{-12} \sim 2^{-9}$)	-	($2^{-16} \sim 2^{-13}$)	($2^{-12} \sim 2^{-9}$)	-	($2^{-8} \sim 2^{-5}$)
	90.24%	67.89%	80.29%	91.28%	30.98%	90.81%	91.15%	51.72%	83.12%
	($2^{-18} \sim 2^{-3}$)	($2^{-16} \sim 2^{-1}$)	($2^{-16} \sim 2^{-1}$)	($2^{-22} \sim 2^{-7}$)	($2^{-22} \sim 2^{-7}$)	($2^{-18} \sim 2^{-3}$)	($2^{-20} \sim 2^{-5}$)	($2^{-16} \sim 2^{-1}$)	($2^{-20} \sim 2^{-5}$)
8	90.24%	64.18%	82.50%	91.28%	27.76%	90.40%	89.89%	17.37%	82.04%
	($2^{-10} \sim 2^{-3}$)	($2^{-10} \sim 2^{-3}$)	($2^{-12} \sim 2^{-5}$)	($2^{-14} \sim 2^{-7}$)	($2^{-14} \sim 2^{-7}$)	($2^{-16} \sim 2^{-9}$)	($2^{-14} \sim 2^{-7}$)	($2^{-16} \sim 2^{-9}$)	($2^{-14} \sim 2^{-7}$)
	84.30%	32.75%	48.63%	84.73%	-	56.67%	27.47%	15.48%	15.10%
	($2^{-8} \sim 2^{-5}$)	($2^{-10} \sim 2^{-7}$)	($2^{-8} \sim 2^{-5}$)	($2^{-12} \sim 2^{-9}$)	-	($2^{-14} \sim 2^{-11}$)	($2^{-12} \sim 2^{-9}$)	($2^{-20} \sim 2^{-17}$)	($2^{-10} \sim 2^{-7}$)
8	90.49%	59.55%	82.14%	91.47%	47.72%	92.01%	91.45%	45.97%	86.89%
	($2^{-18} \sim 2^{-3}$)	($2^{-16} \sim 2^{-1}$)	($2^{-16} \sim 2^{-3}$)	($2^{-22} \sim 2^{-7}$)	($2^{-22} \sim 2^{-7}$)	($2^{-16} \sim 2^{-1}$)	($2^{-20} \sim 2^{-5}$)	($2^{-20} \sim 2^{-5}$)	($2^{-20} \sim 2^{-5}$)
	90.46%	56.76%	83.12%	91.47%	40.02%	91.96%	90.63%	21.94%	68.56%
	($2^{-10} \sim 2^{-3}$)	($2^{-10} \sim 2^{-3}$)	($2^{-12} \sim 2^{-5}$)	($2^{-14} \sim 2^{-7}$)	($2^{-16} \sim 2^{-9}$)	($2^{-14} \sim 2^{-7}$)	($2^{-14} \sim 2^{-7}$)	($2^{-16} \sim 2^{-9}$)	($2^{-14} \sim 2^{-7}$)
8	87.96%	31.54%	75.08%	87.03%	18.95%	85.33%	51.60%	16.00%	24.49%
	($2^{-8} \sim 2^{-5}$)	($2^{-8} \sim 2^{-5}$)	($2^{-8} \sim 2^{-5}$)	($2^{-12} \sim 2^{-9}$)	($2^{-16} \sim 2^{-13}$)	($2^{-12} \sim 2^{-9}$)	($2^{-12} \sim 2^{-9}$)	($2^{-16} \sim 2^{-13}$)	($2^{-8} \sim 2^{-5}$)
Max. Segment	APoT-PWLF (Accuracy/Exponent Range)								
4	90.98%	86.43%	77.28%	92.09%	23.84%	87.98%	91.66%	64.45%	72.89%
	($2^{-18} \sim 2^{-3}$)	($2^{-16} \sim 2^{-3}$)	($2^{-16} \sim 2^{-1}$)	($2^{-16} \sim 2^{-1}$)	($2^{-16} \sim 2^{-1}$)	($2^{-24} \sim 2^{-9}$)	($2^{-22} \sim 2^{-7}$)	($2^{-22} \sim 2^{-5}$)	($2^{-18} \sim 2^{-3}$)
	90.85%	81.58%	76.44%	92.02%	23.93%	73.69%	91.35%	22.51%	60.05%
	($2^{-10} \sim 2^{-3}$)	($2^{-10} \sim 2^{-3}$)	($2^{-12} \sim 2^{-5}$)	($2^{-14} \sim 2^{-7}$)	($2^{-16} \sim 2^{-9}$)	($2^{-16} \sim 2^{-9}$)	($2^{-14} \sim 2^{-7}$)	($2^{-16} \sim 2^{-9}$)	($2^{-14} \sim 2^{-7}$)
6	85.07%	30.40%	43.15%	86.68%	-	14.65%	33.41%	-	20.85%
	($2^{-8} \sim 2^{-5}$)	($2^{-8} \sim 2^{-5}$)	($2^{-10} \sim 2^{-7}$)	($2^{-12} \sim 2^{-9}$)	-	($2^{-18} \sim 2^{-15}$)	($2^{-12} \sim 2^{-9}$)	-	($2^{-8} \sim 2^{-5}$)
	91.04%	86.89%	85.16%	92.12%	76.92%	91.96%	91.79%	80.14%	88.45%
	($2^{-18} \sim 2^{-3}$)	($2^{-16} \sim 2^{-1}$)	($2^{-20} \sim 2^{-5}$)	($2^{-18} \sim 2^{-3}$)	($2^{-22} \sim 2^{-7}$)	($2^{-20} \sim 2^{-5}$)	($2^{-20} \sim 2^{-5}$)	($2^{-18} \sim 2^{-3}$)	($2^{-22} \sim 2^{-7}$)
8	90.93%	79.28%	86.04%	92.09%	47.46%	89.57%	91.71%	40.81%	87.64%
	($2^{-10} \sim 2^{-3}$)	($2^{-10} \sim 2^{-3}$)	($2^{-12} \sim 2^{-5}$)	($2^{-14} \sim 2^{-7}$)	($2^{-16} \sim 2^{-9}$)	($2^{-18} \sim 2^{-11}$)	($2^{-14} \sim 2^{-7}$)	($2^{-16} \sim 2^{-9}$)	($2^{-14} \sim 2^{-7}$)
	88.59%	33.68%	54.43%	89.22%	-	85.70%	68.90%	15.06%	-
	($2^{-8} \sim 2^{-5}$)	($2^{-10} \sim 2^{-7}$)	($2^{-8} \sim 2^{-5}$)	($2^{-12} \sim 2^{-9}$)	-	($2^{-14} \sim 2^{-11}$)	($2^{-12} \sim 2^{-9}$)	($2^{-22} \sim 2^{-19}$)	-
8	91.14%	86.16%	85.78%	92.09%	85.92%	92.42%	91.94%	75.49%	89.26%
	($2^{-18} \sim 2^{-3}$)	($2^{-16} \sim 2^{-1}$)	($2^{-18} \sim 2^{-3}$)	($2^{-18} \sim 2^{-3}$)	($2^{-22} \sim 2^{-7}$)	($2^{-20} \sim 2^{-5}$)	($2^{-22} \sim 2^{-7}$)	($2^{-20} \sim 2^{-5}$)	($2^{-22} \sim 2^{-7}$)
	90.94%	77.83%	85.82%	92.04%	75.46%	92.38%	91.83%	23.97%	85.14%
	($2^{-10} \sim 2^{-3}$)	($2^{-10} \sim 2^{-3}$)	($2^{-12} \sim 2^{-5}$)	($2^{-14} \sim 2^{-7}$)	($2^{-16} \sim 2^{-9}$)	($2^{-16} \sim 2^{-9}$)	($2^{-14} \sim 2^{-7}$)	($2^{-18} \sim 2^{-11}$)	($2^{-16} \sim 2^{-9}$)
8	89.70%	37.46%	71.24%	90.20%	21.23%	89.22%	79.10%	15.98%	40.71%
	($2^{-8} \sim 2^{-5}$)	($2^{-10} \sim 2^{-7}$)	($2^{-10} \sim 2^{-7}$)	($2^{-12} \sim 2^{-9}$)	($2^{-14} \sim 2^{-11}$)	($2^{-14} \sim 2^{-11}$)	($2^{-12} \sim 2^{-9}$)	($2^{-20} \sim 2^{-17}$)	($2^{-14} \sim 2^{-11}$)

the input data, according to the 1-bit setting input, this shifter unit will determine whether to add the right-shifted data to the loaded sum output. Then, it will transfer the 1-bit right-shifted input and the sum result to the next shifter unit.

As a result, if we can implement a 4/8/16 right shifter unit and connect them as a pipeline, we can compute the results of the inputs times PoT or APoT slopes. To control these shifter units, as shown in Figure 3, we defined the shifter control encoding format, which is the *Setting In* signals in each shifter unit of Figure 4. Figure 3 shows the 17-bit shifter control encoding for 16 shifter units in PoT and APoT approximation with the power range of $[-10, 6]$. Its first bit is the sign bit.

For the PoT-PWLF shifter computation in Figure 3 (down), because it only supports the format of single 2^n , the setting should consist of multiple consecutive "1" and consecutive "0". For instance, if the slope is $\frac{1}{8}$, the digits from 32 to $\frac{1}{8}$ should be "1". Then, the 6-bit pre-left-shifted input will be

right-shifted 9 times in the shifter units, which received the "1" from *Setting In*. The result from the last shifter unit is equal to the original input divided by 8 (right shifting 3 bits). For APoT-PWLF shifter computation in Figure 3 (up), if the slope is $1 + \frac{1}{2} + \frac{1}{16} + \frac{1}{1024}$, the *Setting In* digits for 1, $\frac{1}{2}$, $\frac{1}{16}$, and $\frac{1}{1024}$ should be "1". Then, the 6-bit pre-left-shifted input will be executed 1-bit right shifting in every shifter unit, and the shifted data in 1, $\frac{1}{2}$, $\frac{1}{16}$, and $\frac{1}{1024}$ units will be added into the sum.

If all shifter encoding bits are 0, it means the slope is 0.

Based on evaluation results in Table IV and Table V, considering we limit the exponent range of slopes in PoT-PWLF and APoT-PWLF functions to 4/8/16 continuous 2^n , and the n is always negative, we give a pre-right-shifting, which means dividing by 2^{-m} to every input. Then, the shifter pipeline will only need to process the continuous right shifting in $[2^{-4}, 2^{-1}]$, $[2^{-8}, 2^{-1}]$, and $[2^{-16}, 2^{-1}]$. Therefore,

TABLE V: Comparing the Accuracy of Original QNN and PWLF/PoT-PWLF/APoT-PWLF Approximated Models based on Greedy-PWLF Algorithm on ImageNet [17] with ResNet-18 [16] Network

Experiment Result								
Precision Activation	8-bit				Mixed-Precision			
	ReLU Top-1	ReLU Top-5	ReLU+SiLU Top-1	ReLU+SiLU Top-5	ReLU Top-1	ReLU Top-5	ReLU+SiLU Top-1	ReLU+SiLU Top-5
Original	67.07%	87.23%	67.44%	87.23%	65.41%	86.29%	65.26%	86.10%
Max. Segment	PWLF							
4	65.76%	86.31%	60.41%	82.75%	53.59%	77.39%	55.34%	79.16%
6	66.71%	86.96%	66.50%	86.66%	62.63%	84.13%	56.78%	79.82%
8	66.69%	87.01%	67.17%	87.06%	64.23%	85.47%	62.20%	84.29%
Max. Segment	APoT-PWLF (Accuracy/Exponent Range)							
4	65.76%	86.25%	60.42%	82.76%	53.67%	77.45%	55.47%	79.34%
	($2^{-20} \sim 2^{-5}$)		($2^{-22} \sim 2^{-7}$)		($2^{-20} \sim 2^{-5}$)		($2^{-20} \sim 2^{-5}$)	
	64.89%	85.81%	44.38%	70.56%	42.53%	67.94%	45.78%	71.08%
	($2^{-14} \sim 2^{-7}$)		($2^{-16} \sim 2^{-9}$)		($2^{-12} \sim 2^{-5}$)		($2^{-12} \sim 2^{-5}$)	
6	55.42%	78.94%	-	-	-	-	-	-
	($2^{-12} \sim 2^{-9}$)		-	-	-	-	-	-
	66.67%	86.97%	66.47%	86.64%	62.48%	84.11%	56.84%	79.75%
	($2^{-22} \sim 2^{-7}$)		($2^{-22} \sim 2^{-7}$)		($2^{-20} \sim 2^{-5}$)		($2^{-18} \sim 2^{-3}$)	
8	66.65%	86.92%	64.35%	85.46%	59.55%	82.09%	46.65%	71.43%
	($2^{-16} \sim 2^{-9}$)		($2^{-16} \sim 2^{-9}$)		($2^{-14} \sim 2^{-7}$)		($2^{-12} \sim 2^{-5}$)	
	60.03%	82.46%	-	-	-	-	-	-
	($2^{-12} \sim 2^{-9}$)		-	-	-	-	-	-
8	66.75%	87.00%	67.22%	87.04%	64.28%	85.48%	62.56%	84.39%
	($2^{-22} \sim 2^{-7}$)		($2^{-18} \sim 2^{-3}$)		($2^{-18} \sim 2^{-3}$)		($2^{-22} \sim 2^{-7}$)	
	66.71%	86.92%	65.86%	86.35%	63.41%	84.95%	61.45%	83.54%
	($2^{-16} \sim 2^{-9}$)		($2^{-14} \sim 2^{-7}$)		($2^{-14} \sim 2^{-7}$)		($2^{-14} \sim 2^{-7}$)	
8	60.30%	82.65%	48.95%	73.23%	-	-	-	-
	($2^{-12} \sim 2^{-9}$)		($2^{-12} \sim 2^{-9}$)		-	-	-	-

unlike the encoding setting in Figure 3, in the real hardware implementation, all shifting encodings are negative. We used some positive powers of 2^n in Figure 3 because we allowed a wider range of slopes for piecewise linearized functions in our early-stage experiment based on the *pwlf* library.

Therefore, based on the above-mentioned principle, we implemented two different GRAU architectures for PoT-PWLF and APoT-PWLF shifter units. Figure 5 shows the serialized architecture of GRAU. It only implements one shifter unit and reuses it for different slope approximations. Figure 6 shows the pipelined architecture of GRAU, which consists of a shifter unit pipeline.

The integer outputs of the MAC will be loaded into the thresholds first to determine which segment they belong to. As shown in Table IV or Table V, for a PWLF function with 6 segments, if we consider the inputs, which are out of the range of the function approximation, belong to the first and last segments, we only need to implement 5 thresholds to classify the inputs. After obtaining the index of the segment by thresholds, GRAU loads the shifter unit setting from the setting buffer, which is a look-up table, and passes it to the setting loader to regenerate the setting encoded in the correct format for both pipelined and serialized GRAU designs. The MAC output will be transmitted to the initial module simultaneously to apply the pre-right shift. After both input and shifter settings are ready for processing, they will be loaded into the serialized shifter unit or shifter unit pipeline to compute the products between the input and the slope, approximated in the PoT or APoT form. The final result will apply the sign bit and bias to complete the piecewise linear approximation for the activation unit. Therefore, based on the above-mentioned designs, if we reload the value of thresholds and shifter settings, GRAU can

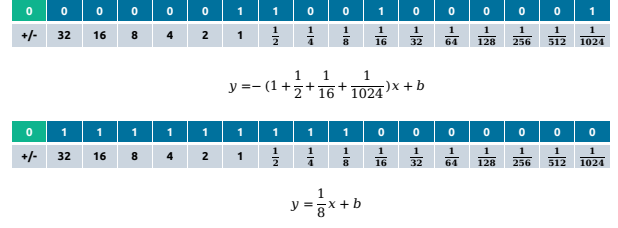


Fig. 3: Encoding of segment slopes for PoT-PWLF (down) and APoT-PWLF (up) approximation

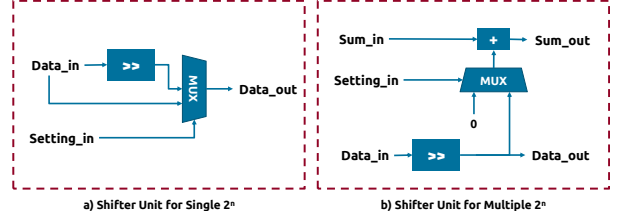


Fig. 4: Shifter unit design in hardware for PoT-PWLF (a) and APoT-PWLF (b) approximation

be reconfigured to approximate different nonlinear functions for the QNN acceleration on hardware.

III. EVALUATION

To evaluate this work, we implemented 16 different activation unit instances on hardware based on *Multi-Threshold* (MT) architecture and PoT/APoT-PWLF-based GRAU architectures. Three of these 16 instances are implemented as a serialized structure, and the others are pipelined structures. Considering that the *Multi-Threshold* activation unit can reconfigure thresholds for different activation functions, which can also be considered a kind of generic reconfigurable activation unit, we include it in this evaluation as a baseline. This baseline follows the official FINN-R implementation[2], where each output precision level requires $2^n - 1$ thresholds after BN folding. This behavior is inherent to the MT paradigm rather than a design choice, and is well documented in prior FINN literature. We optimized it as a pipelined structure consisting of 255 threshold units connected in a pipeline, and a serialized structure that implements only one reusable threshold with 255 threshold registers.

Therefore, as shown in Table VI, we synthesized and implemented these instances using *Vivado* on the *Ultra96-V2* FPGA platform. Based on the post-implementation timing simulation, we evaluated their power, critical path delay, *Area-Delay-Product* (ADP), and *Power-Delay-Product* (PDP).

1) *Hardware Resource Consumption*: Table VI shows that both PoT-PWLF and APoT-PWLF activation units are much more LUT-efficient than the MT unit. Compared with the pipelined MT design requiring 10206 LUTs, the proposed pipelined PoT/APoT-PWLF units only use 324 ~ 755 and 376 ~ 895 LUTs, respectively, i.e., only 3.2% ~ 7.4% and 3.7% ~ 8.8% of the MT cost. A similar trend also holds for the serialized designs (270/283 vs. 2796 LUTs). This advantage

TABLE VI: Hardware Results of Multi-Threshold, PoT-PWLF, and APoT-PWLF Activation Units

Activation Unit	Design	Segment Number	Exponent Number	LUT	FF	Frequency	Total Delay (ns)	Dynamic Power (W)	Power-Delay Products	Area-Delay Products	1-bit	Pipeline Depth 2-bit	4-bit	8-bit
Multi-Threshold	Serialized	-	-	10206	18568	200MHz	2.848	0.129	0.3674	29066.688	1	3	15	255
		-	-	2796	8264	100MHz	5.777	0.032	0.1849	16152.492	-	-	-	-
PoT-PWLF	Serialized	4	8	324	500	250MHz	1.705	0.010	0.0171	552.420	1	3	14	14
		4	16	560	816		1.662	0.013	0.0216	930.720	1	3	22	22
		6	8	408	675		1.658	0.013	0.0216	676.464	1	3	16	16
		6	16	647	1007		1.570	0.015	0.0236	1015.790	1	3	24	24
		8	8	507	854		1.811	0.013	0.0235	918.177	1	3	18	18
		8	16	755	1202		1.655	0.015	0.0248	1249.525	1	3	26	26
	Serialized	-	-	270	456		2.338	0.012	0.0281	631.260	-	-	-	-
		4	8	376	534	250MHz	1.675	0.011	0.0184	629.800	1	3	14	14
		4	16	699	906		1.684	0.014	0.0236	1177.116	1	3	22	22
		6	8	458	709		1.858	0.013	0.0242	850.964	1	3	16	16
APoT-PWLF	Serialized	6	16	786	1097		1.946	0.016	0.0311	1529.556	1	3	24	24
		8	8	558	888		1.608	0.013	0.0209	897.264	1	3	18	18
		8	16	895	1292		1.775	0.017	0.0302	1588.625	1	3	26	26
		-	-	283	463		2.352	0.011	0.0259	665.616	-	-	-	-
	Serialized	-	-	283	463		2.352	0.011	0.0259	665.616	-	-	-	-
		4	8	376	534		1.675	0.011	0.0184	629.800	1	3	14	14
		4	16	699	906		1.684	0.014	0.0236	1177.116	1	3	22	22
		6	8	458	709		1.858	0.013	0.0242	850.964	1	3	16	16
	Serialized	6	16	786	1097		1.946	0.016	0.0311	1529.556	1	3	24	24
		8	8	558	888		1.608	0.013	0.0209	897.264	1	3	18	18
		8	16	895	1292		1.775	0.017	0.0302	1588.625	1	3	26	26
		-	-	283	463		2.352	0.011	0.0259	665.616	-	-	-	-

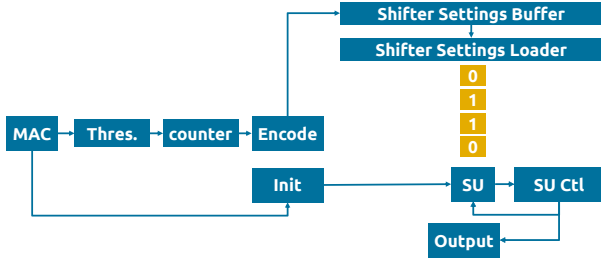


Fig. 5: Serialized hardware implementation of Generic Activation Unit PoT-PWLF and APoT-PWLF shifter unit

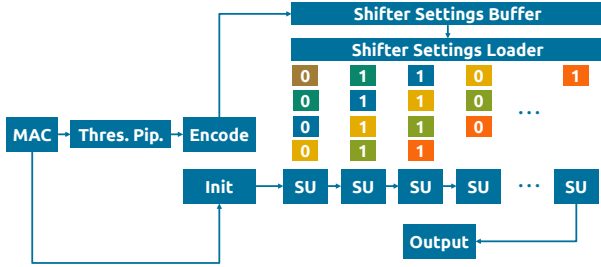


Fig. 6: Pipelined hardware implementation of Generic Activation Unit PoT-PWLF and APoT-PWLF shifter unit

mainly comes from the fact that the PWLF-based design scales with the number of segments and exponent candidates, whereas the MT unit scales with maximum output precision, as also reflected by its much deeper 255-threshold pipeline. Moreover, within the proposed design, increasing the segment number is generally more LUT-efficient than increasing the exponent number. For example, in PoT-PWLF, increasing the exponent number from 8 to 16 raises the LUT count from 324 to 560 for the 4-segment case, while increasing the segment number from 4 to 8 with 8 exponents only raises the LUT count from 324 to 507. Similar trends can also be observed for APoT-PWLF. Combined with the accuracy results, this suggests that allocating hardware budget to more segments is often more cost-effective than allocating it to more 2^n candidates, while larger exponent sets are mainly beneficial for the most difficult cases.

2) *Pipeline Depth*: As shown in the pipeline depth column in Table VI, pipelined *Multi-Threshold* activation unit takes 1/3/15/255 cycles to processing one input. Because our pipeline GRAU instances have one pre-right-shifting unit, 3/5/7 thresholds, 8/16 right shifters, one sign bit processing unit, and one bias adder, it will take 24 cycles to complete one approximate nonlinear processing, which is slower than the 1/2/4-bit quantization of the *Multi-Threshold* activation unit. However, considering that GRAU also has thresholds, it can support the 1/2-bit *Multi-Threshold* quantization with 1/3 thresholds. We implemented a bypass for our GRAU instances for 1/2-bit. Therefore, as shown in the latency column in Table VI, they take the same cycles as MT units in 1/2-bit columns.

3) *Total Delay, Power, ADP, and PDP*: The post-implementation timing simulation on Vivado reported the power, critical path delay, *Area-Delay-Product* (ADP), and *Power-Delay-Product* (PDP) of our 16 instances. The frequency shows the highest frequency these instances can support. Therefore, we can infer that our GRAU implementations can support higher clock frequencies due to their low critical total path delay. Moreover, the lower ADP and PDP of our GRAU also show that our work has better power and design efficiency than the *Multi-Threshold* activation units.

IV. CONCLUSION AND FURTHER WORKS

To support the multi-function, mixed-precision QNN and optimize the hardware resource consumption, we explored the *Generic Reconfigurable Activation Unit* (GRAU) based on our fast greedy *Piecewise Linear Fitting* approximated algorithm to generate *Power-of-Two approximated PWLF* function and *Additive Power-of-Two approximated PWLF* function. The experiment results for the VGG16 and ResNet18 models, using the CIFAR-10 and ImageNet datasets, show that the PoT-PWLF and APoT-PWLF approximations result in a small accuracy loss compared to the original, accurate QNN models. Therefore, we implemented the serialized and pipelined PoT-PWLF and APoT-PWLF approximated *Generic Reconfigurable Activation Unit* (GRAU) in this work. The implementation results show that our work can reduce more than 90% of the

LUT consumption of *Multi-Threshold*-based generic activation functions, supporting higher clock frequencies with better power and hardware efficiency.

In our current design, we set a unified approximation range of 2^n for all layers in network models. However, based on our analysis of the MAC output range in each layer suggest that the input range of our GRAU unit can be between $[-10^6, 10^6]$ or $[-10^4, 10^4]$. Therefore, if we flexibly apply the different continuous exponent ranges in different layers, the shifter pipeline depth perhaps can be reduced to save more hardware resources. We will deeply explore the fine-grained approximation for the QNN models and the design space exploration of hardware based on our PWLF algorithm.

REFERENCES

- [1] Yaman Umuroglu et al. "FINN: A framework for fast, scalable binarized neural network inference". In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2017, pp. 65–74.
- [2] Michaela Blott et al. "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks". In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11.3 (2018), pp. 1–23.
- [3] Yuhao Liu, Salim Ullah, and Akash Kumar. "Bitwise Systolic Array Architecture for Runtime-Reconfigurable Multi-Precision Quantized Multiplication on Hardware Accelerators". In: *2025 26th International Symposium on Quality Electronic Design (ISQED)*. 2025, pp. 1–9.
- [4] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.
- [5] Stefan Elfving, Eiji Uchibe, and Kenji Doya. "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning". In: *Neural networks* 107 (2018), pp. 3–11.
- [6] Fei Lyu et al. "ML-PLAC: Multiplierless Piecewise Linear Approximation for Nonlinear Function Evaluation". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 69.4 (2022), pp. 1546–1559.
- [7] Ming Zhang, Stamatis Vassiliadis, and Jose G. Delgado-Frias. "Sigmoid generators for neural computing using piecewise approximations". In: *IEEE transactions on Computers* 45.9 (1996), pp. 1045–1049.
- [8] Zerun Li et al. "FPGA Implementation for the Sigmoid with Piecewise Linear Fitting Method Based on Curvature Analysis". In: *Electronics* 11.9 (2022).
- [9] Ivan Tsmots, Oleksa Skorokhoda, and Vasyl Rabyk. "Hardware Implementation of Sigmoid Activation Functions using FPGA". In: *2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*. 2019, pp. 34–38.
- [10] Vantruong Nguyen, Jueping Cai, and Linyu Wei. "Low Complexity Sigmoid Function Implementation Using Probability-Based Piecewise Linear Function". In: *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence. ACAI '19*. New York, NY, USA: Association for Computing Machinery, 2020, 236–241.
- [11] Kezhu Liu et al. "Cost effective Tanh activation function circuits based on fast piecewise linear logic". In: *Microelectronics Journal* 138 (2023), p. 105821.
- [12] Safa Bouguezzi, Hassene Faiedh, and Chokri Souani. "Hardware Implementation of Tanh Exponential Activation Function using FPGA". In: *2021 18th International Multi-Conference on Systems, Signals, and Devices (SSD)*. 2021, pp. 1020–1025.
- [13] F. Piazza, A. Uncini, and M. Zenobi. "Neural networks with digital LUT activation functions". In: *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*. Vol. 2. 1993, 1401–1404 vol.2.
- [14] Revathi Pogiri, Samit Ari, and K K Mahapatra. "Design and FPGA Implementation of the LUT based Sigmoid Function for DNN Applications". In: *2022 IEEE International Symposium on Smart Electronic Systems (iSES)*. 2022, pp. 410–413.
- [15] Pramod Kumar Meher. "An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks". In: *2010 18th IEEE/IFIP International Conference on VLSI and System-on-Chip*. 2010, pp. 91–95.
- [16] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].
- [17] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255.
- [18] Alessandro Pappalardo. *Xilinx/brevitas*. 2023.
- [19] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).
- [20] Charles F. Jekel and Gerhard Venter. *pwlf: A Python Library for Fitting 1D Continuous Piecewise Linear Functions*. 2019.
- [21] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).