An Efficient Hardware-Friendly Design of Vision Transformer With Integer-Only Computation and Optimized Tiling

Group: A501 Member : Chao-Wei, Chang Advisor: : Prof. Chih-Tsun, Huang

Abstract

In recent years, Vision Transformer(ViT) has emerged as a leading architecture for image classification, rivaling CNNs. However, compared to CNNs, Transformers are more complex and resource-intensive, prompting many studies focused on reducing their computational complexity. I-ViT is the first work to fully approximate all nonlinear functions in ViT using integer-only operations. It not only integrates previously proposed LayerNorm approximations but also simplifies the implementation of exponential functions using base-2 transformations and provides a new approximation methods for SoftMax and GeLU functions. [1]

Despite these advancements, I-ViT still contains certain operations that are not hardware-friendly, such as reciprocal computations and extensive use of multiplication and division. Building on the I-ViT framework, this study proposes a hardware-efficient FPGA design targeting the Multi-Head Self-Attention mechanism in ViT. We adopt the same 8-bit integer quantization format used in I-ViT and further simplify and optimize the operations such as exponentiation and division. For the exponential function, we propose a 3-stages pipelined exponential modules architecture to accelerate computation and employ seven parallel exponential modules, enabling SoftMax computation on 196 inputs in just 66 clock cycles. For matrix multiplication, we apply tiling techniques to partition matrices and proposed a dedicated timing schedule that allow the system to process Q×K and SoftMax(QK)×V operations concurrently, enhancing overall computational efficiency.

Overview: Vision Transformer[8-9]

Vision Transformer is one popular deep learning model that applies the transformer architecture to image classification tasks, leveraging self-attention mechanisms to capture global dependencies across image patches. Unlike CNNs, which rely on local receptive fields, ViT treats an image as a sequence of patches and processes them using Multi-Head Self-Attention (MSA) and a Feed-Forward Network (FFN) within the transformer encoder.

Multi-Head Self-Attention (MSA)

Consider a ViT model with h heads, and given an input sequence of patch embeddings $X \in \mathbb{R}^{N\times D}$, where N is the number of patches and D is the embedding dimension. The three input matrixs query(Q), key(K), and value(V) obtained by linear transformations are prepared as the input of the self-attention mechanism:

$$\begin{cases} Q = XW_Q \\ K = XW_K \\ V = XW_V \end{cases}$$

with weight matrices $W_Q, W_K, W_V \in \mathbb{R}^{N\times d_k}$ and d_k is the number of patches divided by the number of heads.

The self-attention mechanism starts with the inner products of Q and K matrix. Then, the Softmax operation is applied on the product result. Then, the result of previous operation product with V matrix is treated as the output of the self-attention mechanism. Hence, the self-attention mechanism can be represented as the following formula:

$$\mathrm{Attention}(\mathrm{Q},\mathrm{K},\mathrm{V}) = \mathrm{Softmax}\left(QK^T\Big/\sqrt{d_k}\right)\mathrm{V}$$

The division by $\sqrt{d_k}$ ensures numerical stability by scaling the dot product.

Finally, combine the result of multiple self-attention operations into the outpu of multi-head self-attention through a weight matrix $W_{concat} \in \mathbb{R}^{hd_k\times D}$:

$$MSA(X) = Concat(head_1, \ldots, head_h)W_{concat}$$

Feed-Forward Network (FFN)

Following the MSA module, ViT employs a Feed-Forward Network to further process the extracted features. The FFN consists of two fully connected layers with a non-linear GELU activation function in between:

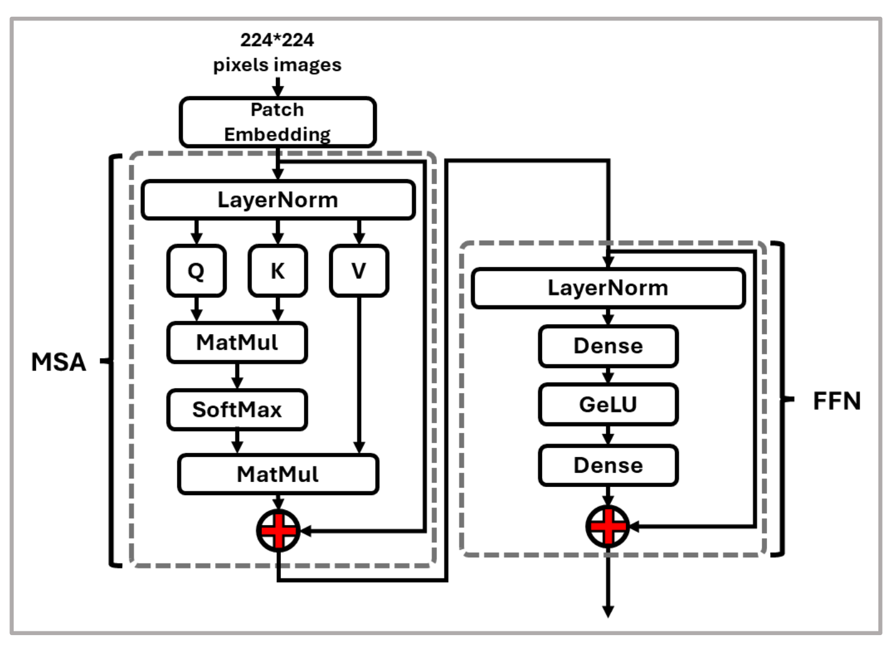
$$\mathrm{FFN}(X) = GELU(XW_1 + bias_1)W_2 + bias_2$$

where X is the output of the MSA after the LayerNorm process with residual connection, $W_1 \in \mathbb{R}^{D\times D_{hidden}}$ and $W_2 \in \mathbb{R}^{D_{hidden}\times D}$ are weight matrices, $bias_1$ and $bias_2$ are bias terms. The FFN enhances the model's expressiveness by applying non-linear transformations to the feature representations.

Both the MSA and FFN components are wrapped within residual connections and layer normalization:

$$X' = LayerNorm(X + MSA(X))$$
$$X'' = LayerNorm(X' + MSA(X'))$$

where layer normalization stabilizes training by normalizing activations across feature dimensions.

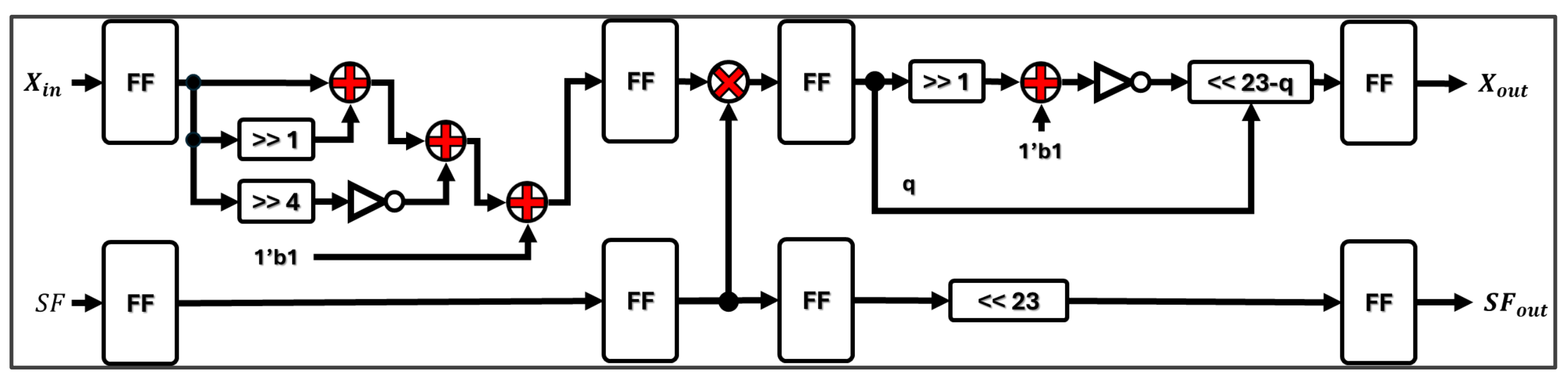


The fundamental Architecture of Vision Transformer.

Hardware Implementation of Exponential Function

Traditional methods for implementing exponential functions in digital circuits typically rely on lookup tables or polynomial approximation methods[5]. While LUT-based approaches enable direct function value lookups, they require substantial hardware resources, making them inefficient for resource-constrained designs. Alternatively, polynomial approximation methods utilize polynomial expansions to approximate exponentiation but necessitate multiple multipliers, leading to high computational complexity and increased hardware utilization.

To address these challenges, various low-precision approximation techniques have been developed based on the logarithmic-exponential transformation.[1] Considering the hardware resource limitations on FPGA, an efficient hardware-aware implementation is required. The low-precision approximation method is applied in this study leveraging exponential-logarithmic base conversion to achieve a hardware-friendly and resource-efficient design.



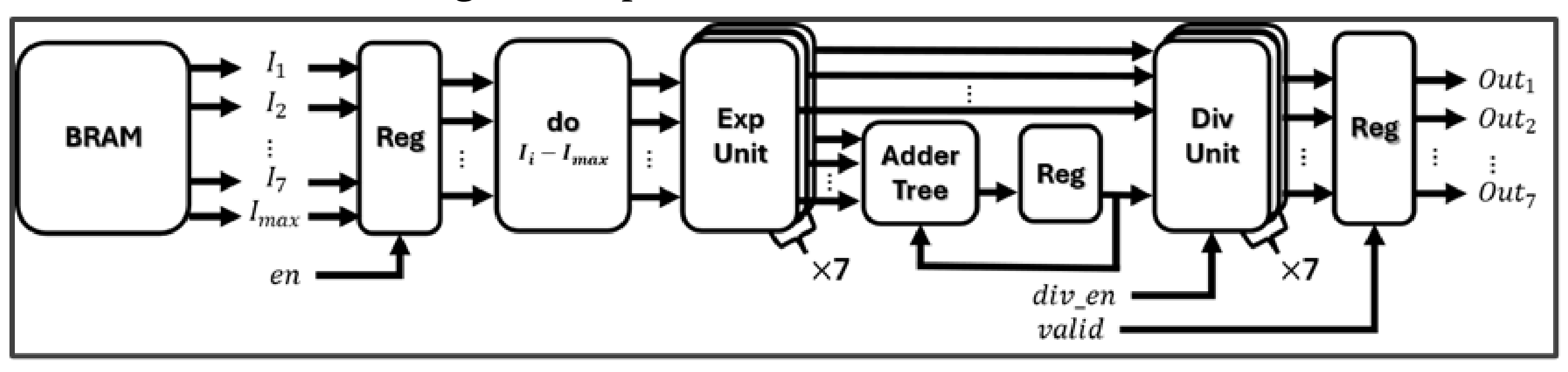
Proposed Exponential Module with three-stage pipeline structure

Hardware Implementation of SoftMax

In ViT, the row-wise SoftMax is applied after the matrix multiplication of the Q and K matrices. Since the input dimension of SoftMax in the DeiT-Tiny model—used as the evaluation benchmark in this study — is 196 × 196, the SoftMax function processes 196 elements per row. However, directly loading all 196 8-bit numbers at once is not feasible for hardware implementation due to resource constraints. To address this, the 196 tokens are divided into 28 groups, with each group containing 7 elements. To smooth the data distribution and prevent overflow. The modified SoftMax function is applied and can be represented as

$$\mathrm{SoftMax}(x_i) = \frac{e^{S_{x_i}(I_{x_i} - I_{max})}}{\sum_j e^{S_{x_j}(I_{x_j} - I_{max})}}$$

The design is divided into two states: the accumulation state(ACC) and the division state (DIV). In the ACC state, the inputs are first processed by the exponential modules after subtracting I_{max}, and the resulting exponentiated values are accumulated using an adder tree. Due to the limitations of memory bandwidth and hardware reuse, the exponential of the DIV state as the numerator is not saved in the ACC state, but directly calculated again in the DIV state through the exponential module.

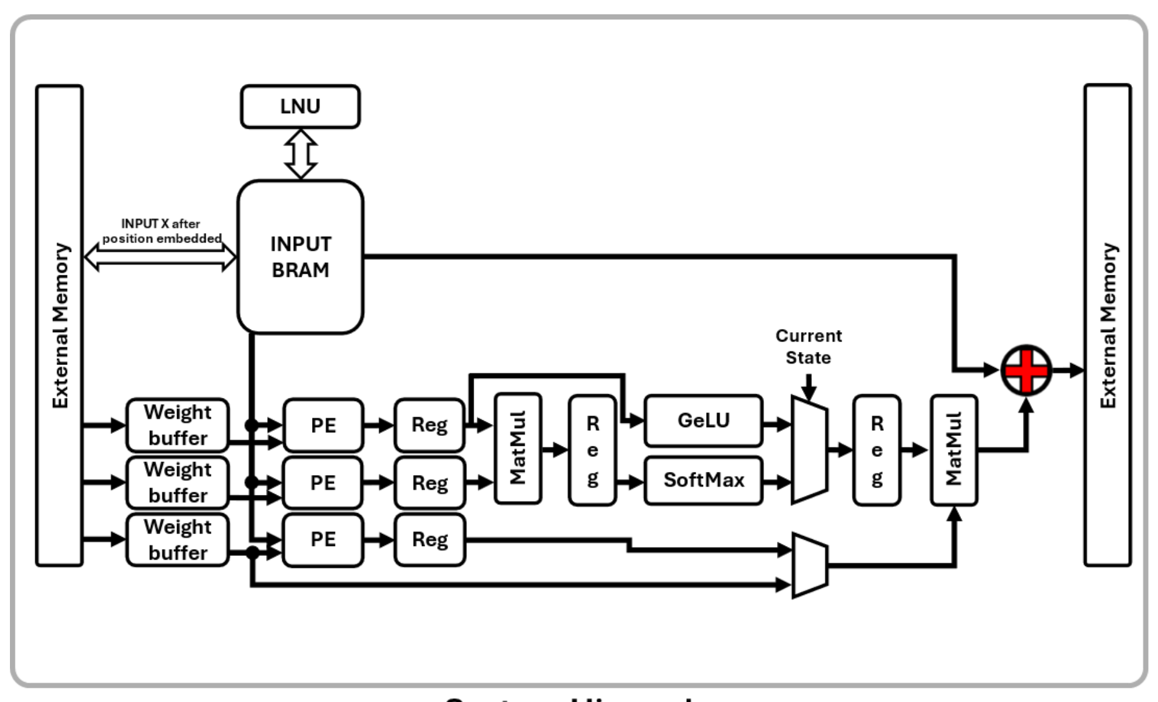


Proposed Exponential Module with three-stage pipeline structure

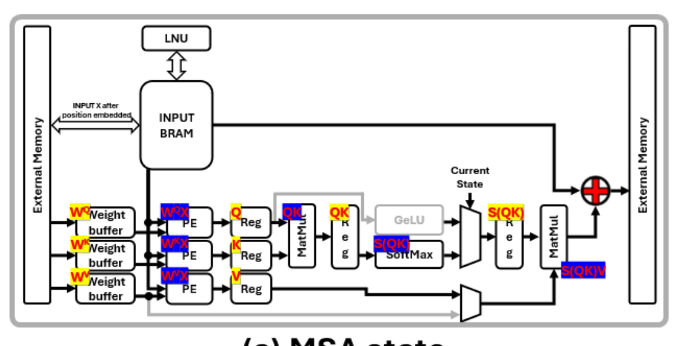
Proposed hardware architecture for ViT

This work focuses on the hardware implementation and optimization of the MSA module in ViT. The proposed system hierarchy, shown in below, is designed to efficiently process input tokens through a structured pipeline. First, the input tokens are multiplied by their corresponding weights using the PE block, after which the resulting Q, K, and V matrices are stored in registers. The designed MatMul module then performs the QK^T matrix operation, feeding the output into a non-linear SoftMax or GeLU function before executing the final matrix multiplication.

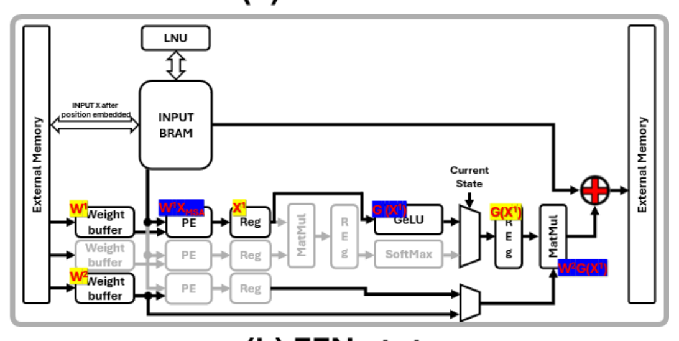
To maintain the residual connections in the original ViT algorithm, an adder combines the final MatMul output with the original input tokens. Additionally, for greater efficiency, the proposed system hierarchy is designed to support both the MSA(Fig. 2(a)) and FFN(Fig. 2(b)) layers in ViT, integrating both the SoftMax and GeLU blocks.



System Hierarchy



(a) MSA state



(b) FFN state

The proposed hardware architecture for ViT supports both the MSA and FFN modules. The top section presents the overall system hierarchy. (a) illustrates the data flow in the MSA stage, where input tokens are processed to generate Q, K, and V, followed by matrix multiplication and non-linear transformations. (b) shows the data flow in the FFN stage.

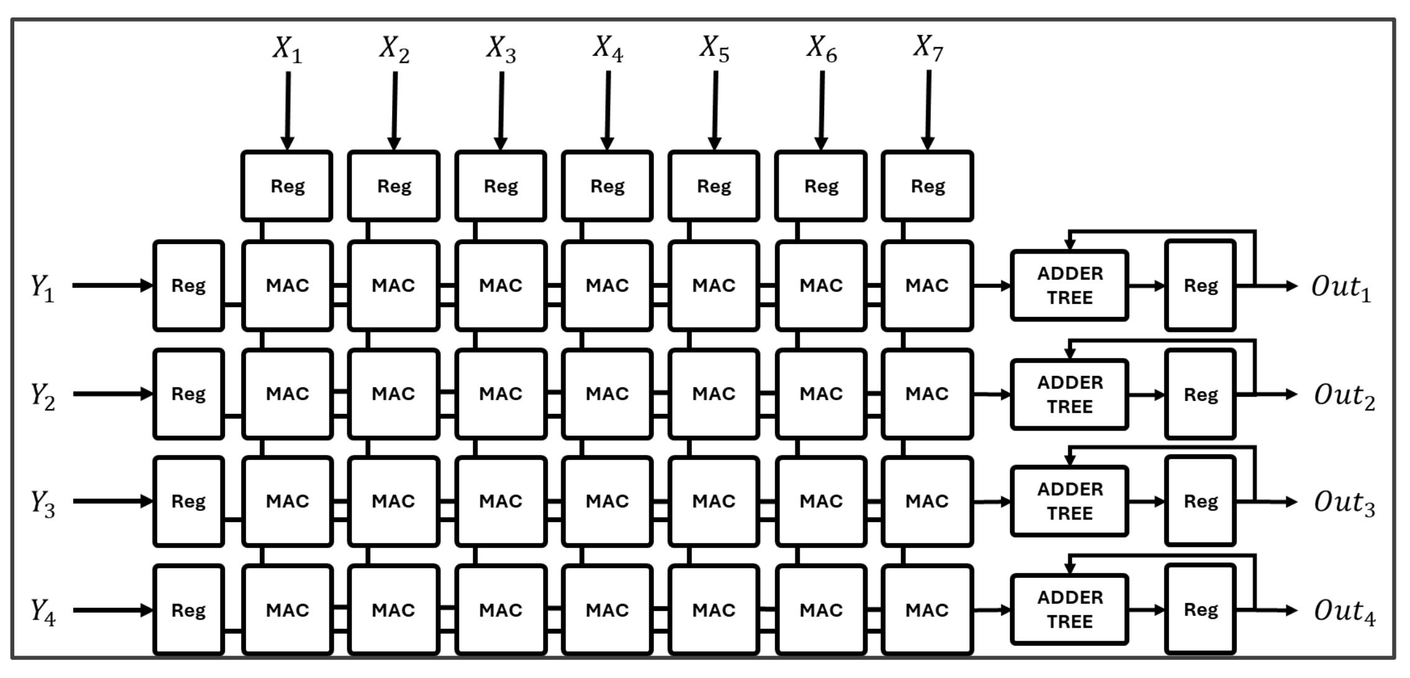
Hardware Implementation of MatMul

In hardware implementation of MatMul, partial sum accumulation and the tiling method are commonly used to efficiently perform matrix multiplication. Inspired by Column-Wise MatMul Algorithm proposed in Co-design of Attention Mechanism[12], this study adopts the row-wise MatMul Algorithm, as it better aligns with the preceding and succeeding stages that involve row-based operations.

Firstly, consider the matrix dimensions involved in the MatMul of the deit_tiny model. For instance, the multiplication could be between matrices of size 196 × 196 or 196 × 64 and 64 × 196. The designed MatMul module processes four rows simultaneously. In each cycle, it loads four elements from the same column of matrix M_1 and seven elements from the same row of matrix M_2 to perform multiply-accumulate operations.

Once all elements in a row of M_1 are multiplied with the corresponding elements in a column of M_2, a single output element in the result matrix O is computed. This accumulated result, held at the output of the adder tree, is then output along with a valid signal indicating completion.

In the DeiT-Tiny model, the input dimensions for MatMul are 196 × 64 and 196 × 196. Since a subsequent step in the computation involves row-wise operations, a row-wise MatMul module is proposed.



Proposed MatMul module.

Result & Conclusion

This work presents a hardware-friendly MSA accelerator based on a modified version of the I-ViT approximation.[1] Using the Fashion MNIST dataset for software validation, the quantized and optimized model achieved an accuracy of 77.0%, with only a slight reduction of 0.9% compared to the original I-ViT architecture (77.91%) on the DeiT-Tiny baseline. Notably, the design eliminates reciprocal operations and greatly simplifies the multiplication and division processes, enhancing hardware efficiency. In the proposed hardware architecture illustrated above, the MSA and FFN are shared between both the MSA and FFN same computing architecture, reducing hardware redundancy compared to traditional architectures that allocate separate resources for each stage.

Despite these achievements, the design still faces certain limitations when compared to more advanced accelerators. The SoftMax module alone consumes over 90% of the total logic and flip-flop resources, suggesting that further optimizations could substantially reduce area and power without significantly compromising accuracy. Additionally, the current implementation focuses only on the MSA component. However, based on the proposed shared architecture for MatMul and nonlinear modules, we can reasonably estimate the performance for the full ViT pipeline. At 150 MHz, our MSA accelerator completes the computation for one head in 0.632 ms. Given that the DeiT-Tiny model uses 3 heads, the full MSA stage would require approximately 1.896 ms. Although the FFN stage hardware is not yet fully implemented, we estimate the FFN would require an additional 0.621 ms based on the MSA and FFN shared architecture. Thus, the total processing time for one image would be approximately 2.517 ms.

In comparison, running the original FP32 DeiT-Tiny model on an RTX 2080Ti GPU requires around 5.99 ms per image, while the I-ViT integer-approximated model achieves inference in 1.61 ms.[1]Although our hardware accelerator improves inference speed by roughly 2.34× over the original model, which seems less than the 3.72× achieved by I-ViT, this discrepancy is expected. In software, all attention heads can be computed in parallel, while in our hardware accelerator, the heads are computed sequentially due to resource sharing and pipeline scheduling. Therefore, the direct comparison of runtime is not entirely one-to-one, and our design still offers a promising balance between hardware cost, modular reuse, and performance. Future work will focus on fully integrating the ViT accelerator, optimizing across all stages, and improving throughput and resource efficiency.

Reference

[1] Z. Li and Q. Gu, "I-vit: Integer-only quantization for efficient vision transformer inference," in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 17065-17075.
[5] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-bert: Integer-only bert quantization," in International conference on machine learning, 2021: PMLR, pp. 5506-5518.
[8] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," arXiv preprint arXiv:2010.11929, 2020.
[9] K. Han et al., "A survey on vision transformer," IEEE transactions on pattern analysis and machine intelligence, vol. 45, no. 1, pp. 87-110, 2022.
[12] X. Zhang, Y. Wu, P. Zhou, X. Tang, and J. Hu, "Algorithm-hardware co-design of attention mechanism on FPGA devices," ACM Transactions on Embedded Computing Systems (TECS), vol. 20, no. 5s, pp. 1-24, 2021.
[X] Doug Zongker, "Chicken Chicken Chicken: Chicken Chicken," Annals of Improbable Research, 12 (5): 16—21, 2006.