The 29th LSI DESIGN CONTEST in OKINAWA 2026

（高信頼知的集積システム研究センターシンポジウム）

2026年コンテストテーマ:
Generative Adversarial Network

開催日: 令和8年3月6日(金)
沖縄県石垣市、大濱信泉記念館

主催: LSIデザインコンテスト実行委員会, 高信頼知的集積システム研究センター

共催: 琉球大学工学部, 九州工業大学情報工学部

協賛: 電子デバイス産業新聞（旧半導体産業新聞）, ギガファーム株式会社,
(株)レイドリクス, 九州計測器(株),
電子情報通信学会スマートインフォメディアシステム研究会,

後援: 国立沖縄工業高等専門学校, 沖縄職業能力開発大学校, CQ出版社

http://www.lsi-contest.com

目次

コンテストの概要・目的

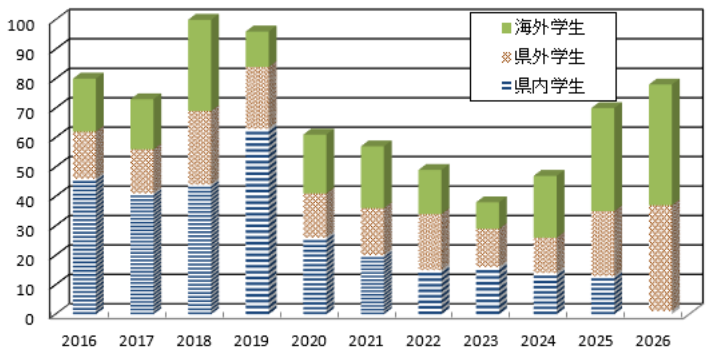
九州・沖縄さらに東南アジア地域の半導体産業および組み込み機器等のエレクトロニクス産業の振興を目的に、標題の学生向けの設計コンテストを毎年実施しております。主催は、琉球大学および九州工業大学の教員で構成するLSIデザインコンテスト実行委員会で実施しております。2026年は、国内外を含め77名（26チーム）を超えた応募がありました。

東シナ海および南シナ海沿岸のLSI産業拠点地域（九州、韓国、台湾、中国、シンガポール、フィリピン、マレーシア）は、世界の半導体市場の実に40％を越える高いシェアを有しており（九州以東の日本本土は含みません！）、その中心に位置する沖縄はそのビジネスチャンスが大変高いものと期待されます。そうした地理的好条件の沖縄にて、学生向けLSI設計コンテストを実施し、学生すなわち未来のエンジニアの設計スキルを上げることにより、将来的に沖縄や東南アジアでの企業誘致やベンチャー起業につなげたいと考えております。

今回は、AIの一つである「GAN (Generative Adversarial Network)」がテーマになります。処理の高速化，回路規模の削減を目指したハードウェア設計など、いろいろなアイデアが生まれることを期待しております。

本コンテストの主旨をご理解頂き、多くの学生の皆様（高専、大学、大学院生）の参加を期待しております。

学生参加数の推移



国内

琉球大学、千葉大学、九州工業大学、会津大学、大分県立工科短期大学、沖縄職業能力開発大学校、大阪工業大学、大阪大学、京都大学、近畿大学、高知工科大学、神戸大学、芝浦工業大学、職業能力開発大学校、東海大学、東京工業大学、東京都立科学技術大学、東京理科大学、東北大学、徳島大学、豊橋技術科学大学、長崎大学、新潟大学、広島大学、法政大学、山形大学、横浜国立大学、立命館大学、早稲田大学、明治大学、日本大学、沖縄高専、有明高専、木更津高専、久留米高専、豊田高専、北陸先端科学技術大学院大学

海外

Bandung Institute of Technology 、 Institut Teknologi Telkom 、Telecommunication College Bandung 、Telkom University (インドネシア)、Gadjah Mada University（インドネシア）、Sumatera Institute of Technology（インドネシア）、Univ. of Science、Ho Chi Minh city、Hanoi Univ. of Science (ベトナム)、Chosun 大学 (韓国)、Univ. of Valladolod (スペイン)、Univ. of Kaiserslautern (ドイツ)、NTI (エジプト)、McMaster 大学 (カナダ)、南西大学 (中国)

これまでの設計課題

■	2001 年：	「デジタルCDMA レシーバ」
■	2002 年：	「差集合巡回符号エラー訂正回路」
■	2003 年：	「静的ハフマン符号用の可変長デコーダ」
■	2004 年：	「共通鍵暗号AES 用SubByte 変換回路」
■	2005 年：	「デジタルFM レシーバ」
■	2006 年：	「2 次元積符号繰り返しデコーダ」
■	2007 年：	「64 点高速フーリエ変換」
■	2008 年：	「RSA 暗号デコーダ」
■	2009 年：	「Small RISC Processor」
■	2010 年：	「エラー訂正：BCH 符号」
■	2011 年：	「DCT」
■	2012 年：	「16/64/128-point Flexible FFT」
■	2013 年：	「SW・HW 協調設計を用いたノイズ除去システム」
■	2014 年：	「SW・HW 協調設計を用いたノイズ除去システム」
■	2015 年：	「正弦波発生回路」
■	2016 年：	「人物検出用パターンマッチング回路」
■	2017 年：	「人物検出動画像処理システム」
■	2018 年：	「ニューラルネットワーク（バックプロパゲーション）」
■	2019 年：	「深層学習（バックプロパゲーション）」
■	2020 年：	「畳み込みニューラルネットワーク（CNN）」
■	2021 年：	「強化学習」
■	2022 年：	「Deep Q-Network」
■	2023 年：	「局所的最大値／最小値」
■	2024 年：	「Autoencoder」
■	2025 年：	「Variational Autoencoder」

■ **2025 年：　「GAN」**

開 催 概 要

名　　称	:	「第 29 回 LSI デザインコンテスト in 沖縄 2026」
実行委員長	:	九州工業大学大学院　情報工学研究院　情報・通信工学研究系　名誉教授　尾知博
主　　催	:	LSI デザインコンテスト実行委員会、高信頼知的集積システム研究センター http://www.lsi-contest.com/
共　　催	:	琉球大学工学部、九州工業大学情報工学部
協　　賛	:	電子デバイス産業新聞（旧半導体産業新聞）、株式会社レイドリクス、 ギガファーム株式会社、九州計測器株式会社、 電子情報通信学会スマートインフォメディアシステム研究会、
後　　援	:	CQ 出版社、 沖縄職業能力開発大学校
実行事務局	:	九州工業大学情報工学部情報・通信工学科黒崎研究室 LSI デザインコンテスト実行委員会事務局
日　　時	:	2026 年 3 月 6 日（金）　13:00～18:00
会　　場	:	大濱信泉記念館 〒907-0004 沖縄県石垣市登野城 2-70
目　　的	:	実践的な課題を用いた学生対象のデジタル集積回路設計のコンテストであり、半導体集積回路設計能力の向上とともに国際的に交流することで学生の工学に関する視野を広めることを目的としている。
対　　象	:	国内大学・大学院生、高専学生、アジア地域大学生
来場予定者	:	100 名（入場無料）
募集方法	:	各大学・高専へのパンフレット送付、LSI 関連雑誌等へのリリース
ホームページ	:	http://www.LSI-contest.com/

【審査員】
実行委員長：　九州工業大学　黒崎 正行、尾知 博
審査委員長：　琉球大学　和田 知久
東京大学　藤田 昌宏
大阪大学　尾上 孝雄
千歳科学技術大学　宮永 喜一
鳥取大学　伊藤 良生
琉球大学　吉田 たけお
ディポネゴロ大学　Wahyul Syafei Amien
バンドン工科大学　Trio Adiono
九州職業能力開発大学校
沖縄職業能力開発大学校
他協賛企業・法人審査員　　　　　　　　　（敬称省略・順不同）

連絡先
住　所：〒820-8502 福岡県飯塚市川津 680-4　九州工業大学大学院情報工学研究院情報・通信工学研究系
電　話：0948-29-7667　　Email: support@LSI-contest.com
LSI デザインコンテスト実行委員会委員長
担当者　黒崎　正行

【設計テーマ】 GAN (Generative Adversarial Network)

第29回のテーマは、AIの中でも画像生成などに応用されている敵対的生成ネットワーク (Generative adversarial networks、GAN) です。今回は「GAN」をテーマとして、処理の高速化、回路規模の削減を目指したハードウェア設計を行うことを目的とします。Level1の課題では、3×3の画を生成する回路を設計します。Level2の課題では、生成器と識別器の回路設計を、Level3では、画素数を増やした画像の生成に関する回路を設計します。Level4の課題では問題や構造はすべて自由とし、よりユニークなGANを使用したシステムを設計します。

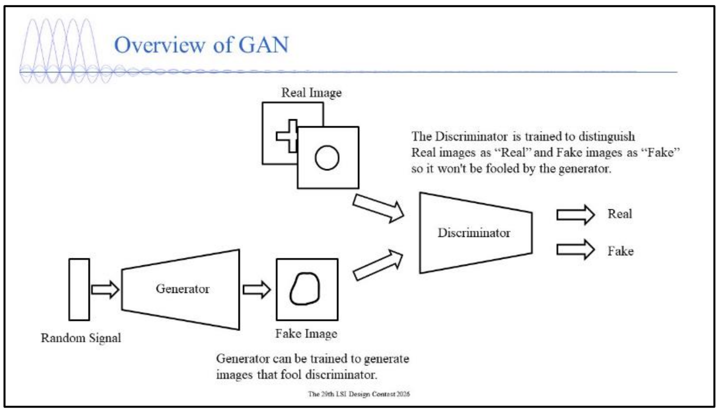
要求されている設計は、HDL (VHDLもしくはVerilogHDL) による設計と論理合成および検証結果です。

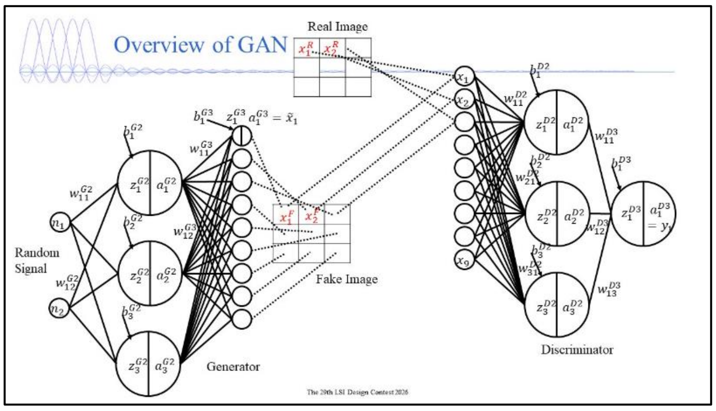
実装アルゴリズムと環境

■ GAN の概要

GAN は、データを生成する生成器 (Generator) と画像が生成された画像かそれともリアルな画像かの真偽を判定する識別器 (Discriminator) で構成され、それぞれが互いに競い合うことで、リアルな画像や動画を生成する手法です。

例として，3×3 の画像における GAN の構成図を示します。


Overview of GAN
Real Image
The Discriminator is trained to distinguish Real images as "Real" and Fake images as "Fake" so it won't be fooled by the generator.
Discriminator
Real
Fake
Generator
Random Signal
Fake Image
Generator can be trained to generate images that fool discriminator.
The 29th LSI Design Contest 2026



■ 実装環境

Synopsys® Synplify Pro® /Premier
Synopsys® Design Compiler®
Mathworks® MATLAB® /Simulink®
Xilinx Vivado® Design Suite
または設計環境に応じて、Synplify Pro/Premier or その他論理合成ツール、RTL hand coding(VHDL or Verilog-HDL)などの設計環境が挙げられます。

課題

- 1. Level1:初心者向け

単純なGANの生成器をHWに実装せよ

識別及び学習はソフトウェアでも可

- 2. Level2:中級者向け

単純なGANの生成器と識別器をHWに実装せよ

学習はソフトウェアでも可

- 3. Level3:上級者向け

大きな画像が扱えるGANを実装せよ

- 4. Level4:最上級者向け

Unlimited(応用的なGANでもOK)

審査：JUDGE

■　審査メンバーによる以下の 4 項目各 10 点で審査を実施（10 point each）

1）アカデミック的、新奇アイデア的な観点（Academic, New idea）

2）実用設計、産業面応用的な観点（Used in real life, Good for industry）

3) FPGA 等の実装レベルの観点（Good prototype by FPGA etc.）

4）プレゼンテーションの観点（Good presentation）

表彰：AWARD

■　優勝（電子情報通信学会 SIS 賞）SIS AWARD

【アカデミック的、新奇アイデア的な観点】の BEST

■　その他、2)、3)、4)の観点からも賞を贈呈します。

CGAN を用いた笑顔変換システム

チーム名：CouChannel

城間 龍弥　保 航聖　平良 迅識　玉城 ひより

Smile conversion system using CGAN

Shiroma Ryuya， Tamotsu Kosei， Taira Tokitsune， Tamaki Hiyori

2994-2 Ikehara Okinawacity Okinawa, 904-2141, Japan
Email address：j2421307@okinawa-pc.ac.jp　j242521310@okinawa-pc.ac.jp
j242521312@okinawa-pc.ac.jp j242521314@okinawa-pc.ac.jp

Abstract_ This paper presents a lightweight FPGA-oriented conditional GAN (CGAN) for smile conversion. To enable hardware implementation under resource constraints, training is performed on a PC while generator inference is designed for FPGA deployment. The network adopts a fully connected architecture (288→170→144) with fixed-point arithmetic. Experimental results using 12×12 emoji images confirm stable training and successful smile generation. The proposed architecture demonstrates the feasibility of compact CGAN inference suitable for FPGA-based implementation.

Keywords_ Conditional GAN (CGAN), FPGA, Smile Conversion, Fixed-Point Arithmetic, Lightweight Neural Network

1. 背景

近年，顔画像の編集技術は発展しているが，集合写真において全員を自然な笑顔に変換することは依然として課題である．Conditional GAN（CGAN）は条件情報を入力とする生成器と識別器から構成される画像生成モデルであり，表情変換への応用が期待される．しかし，CGAN を FPGA 上で実装する場合，ハードウェア資源の制約が大きな課題となる．

本研究では，FPGA 推論を前提とした軽量 CGAN アーキテクチャを提案する．実画像への応用を最終目標とし，第一段階として低解像度の絵文字画像を対象に有効性を検証する．ネットワークの縮小，画像サイズ制限，固定小数点演算の導入を主要な設計方針とする．

2. システム概要

本研究では，学習処理を PC 上で行い，推論処理のみを FPGA 上で実行する構成を採用する．学習と推論を分離することで，FPGA 上では生成器の推論処理に特化した回路構成を実現する．

ネットワークはハードウェア実装を考慮し，生成器および識別器ともに全結合層のみで構成した軽量構造とした．

生成器の入力は条件画像（12×12=144 画素）とランダムノイズ（144 次元）を結合した 288 次元ベクトルとする．ネットワーク構成は 288→170→144 の全結合層構成とした．

推論時には，条件画像情報およびランダムノイズを生成器へ入力し，対応する笑顔画像を生成する．本研究では第一段階として，泣き顔を表す低解像度絵文字画像を条件入力として用いる．

本システムの全体構成を図 1 に示す．PC 上で学習を行い，FPGA では生成器推論のみを実行する分離構成としている．

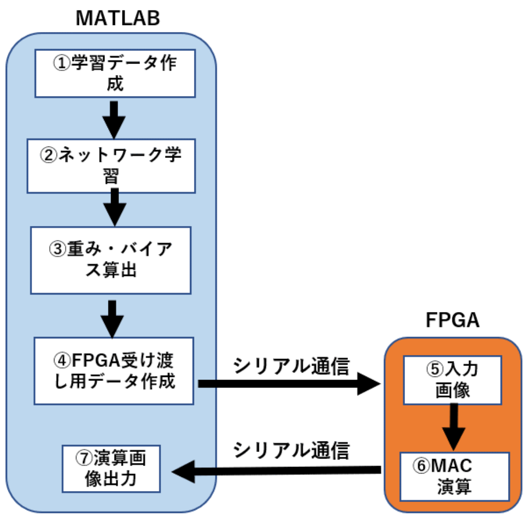


図 1 システムの全体構成

3. 実験結果

提案する軽量 CGAN に対して MATLAB 上で学習および推論検証を行った．生成器は 288→170→144 の全結合構成であり，総パラメータ数は 73,754 である．1 推論あたりの MAC 演算回数は 73,440 回である．

学習後，未使用入力画像を用いて推論を行い，条件画像（泣き顔絵文字）から対応する笑顔画像が生成されることを確認した（図 2）．

固定小数点化の影響を評価するため，MATLAB 上で 16bit 量子化モデルを用いた推論検証を行った．その結果，浮動小数点モデルと比較して生成画像に顕著な劣化は見られなかった．

さらに，本構成と一般的な畳み込み層を含む構成を比較すると，本ネットワークはパラメータ数および演算量を削減できることが確認された．これにより，FPGA 実装に適した軽量モデルであることを示した．

表 1 に生成器の構成および演算規模を示す．

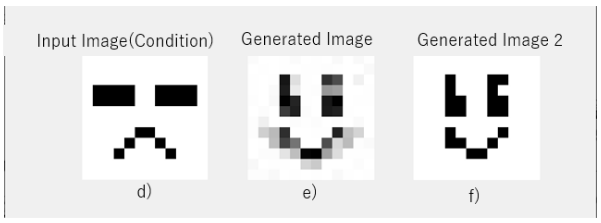


図 2 d)入力画像 e)出力結果 f)二値化後の画像

表 1 生成器構成および演算規模

項目	値
画像サイズ	12×12
入力次元	288
隠れ層ユニット数	170
出力次元	144
総パラメータ数	73,754
1 推論あたりの MAC 演算回数	73,440

4. FPGA アーキテクチャ設計

本研究では，生成器の推論処理を FPGA 上に実装可能な構造として設計した．図 3 に生成器推論回路のブロック構成を示す．本構成は，全演算を MAC 処理に統一した規則的なアーキテクチャである．

生成器は全結合層のみで構成し，畳み込み層を用いないことで回路構造の単純化を図った．これにより，各層の演算はすべて積和演算（MAC 演算）として実装可能である．

本構成における 1 推論あたりの演算量は 73,440 MAC であり，MAC 演算部の並列度を調整することで演算時間と FPGA 資源使用量のトレードオフ設計が可能である．仮に 170 並列 MAC 構成とした場合，1 層あたりの演算サイクル数は約 288 クロックとなる．このように，並列度設計によりリアルタイム動作への拡張可能性を有する．

固定小数点演算（16bit 想定）を採用することで，浮動小数点演算に比べ回路規模を抑制できる．活性化関数は区分線形近似により実装可能であり，追加の複雑な演算回路を必要としない．

本設計では，重みおよびバイアスを外部メモリまたは BRAM に格納し，入力データを逐次読み出すストリーミング構成を想定している．これにより，大規模な演算器を必要とせず，資源制約下でも段階的な演算処理が可能となる．また，中間結果はレジスタで保持し，各層ごとに順次演算を進める構成とする．

さらに，各層が同一形式の MAC 演算で記述できるため，モジュール化設計が容易であり，将来的な並列度拡張にも柔軟に対応可能である．演算精度と回路規模のバランスを考慮し，ビット幅は量子化評価に基づいて最適化する方針である．

以上より，本 CGAN 生成器は FPGA 上での軽量推論に適した実装指向の構造である．

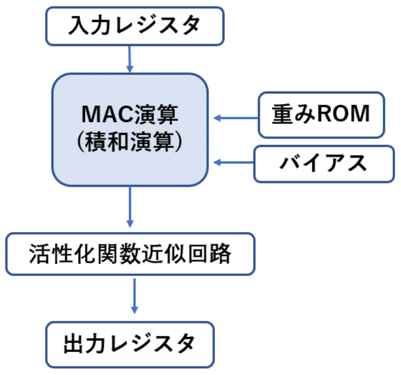


図 3 生成器推論回路の内部構成図

5. まとめ

本研究では，12×12 絵文字画像を対象とした軽量 CGAN を構築し，安定した学習および笑顔生成を確認した．さらに，288→170→144 構成および固定小数点 MAC 演算に基づく FPGA 向けアーキテクチャを設計した．本設計は FPGA 実装に適した構造であり，リアルタイム笑顔変換システムへの応用が期待される．

本研究は，CGAN を FPGA 実装へ適用するための軽量設計指針を示すものである．特に，全結合層構成による演算構造の単純化と固定小数点化の組み合わせは，資源制約下における実装可能性を高める有効なアプローチである．

Hardware/Software Co-design for Image Completion of Missing License Plate Digits

Akina Kohira
Department of Computer Science and Electronics, Kyushu Institute of Technology
kohira.ak@dsp.cse.kyutech.ac.jp

***Abstract*—**
In recent years, missing digits on license plates due to factors such as dirt have posed a challenge by reducing the accuracy of Automatic License Plate Recognition (LPR) systems. In this study, we developed a system to complete missing license plate images by constructing a fully connected model based on Pix2Pix, a type of Generative Adversarial Network (GAN), and implemented the pre-trained generator model onto hardware. Experimental results show that the system achieves high restoration accuracy, maintaining a Structural Similarity Index (SSIM) of 0.8 or higher for images with a missing rate of up to 70%. Furthermore, it was confirmed that the inference accuracy on hardware is maintained at a level equivalent to that of the software implementation.

1 Introduction

In recent years, Automatic License Plate Recognition (LPR) has become widespread. However, missing digits caused by mud or overexposure remain a critical issue that degrades recognition accuracy [1]. Since conventional image processing struggles to restore missing regions, Generative Adversarial Networks (GANs), which can learn context to generate and complete high-quality images, have gained significant attention. However, it is difficult to satisfy both the high real-time requirements of LPR systems and the enormous computational costs of GAN inference through software processing alone. Therefore, this research focuses on FPGAs, which excel at parallel processing. By implementing the image completion model on hardware, we aim to realize efficient, high-quality image completion for LPR systems that balances both inference accuracy and processing speed.

2 Algorithm

This system employs the Pix2Pix learning framework, a conditional GAN suitable for tasks that involves converting an input image to a target image. Specifically, as shown in Fig. 1, the missing digit image is used as the input to the generator, and adversarial learning is performed so that the discriminator distinguishes between the ”completed image” and the ”ground truth image.” Considering the FPGA implementation, the network consists of a lightweight fully connected model rather than a convolutional structure like U-Net.

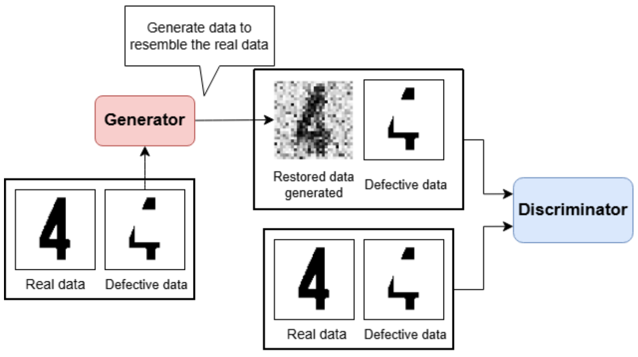


Fig. 1: Processing flow of missing image completion

3 System Overview

3.1 Learning Model Configuration

The overview of the GAN model designed in this study is shown in Fig. 2. The loss function combines the GAN loss with the L1 loss, which represents the error between the generated image and the ground truth image. Furthermore, for training, we utilized a dataset of 1,200 pairs artificially generated via scripts.

3.2 FPGA Implementation Configuration

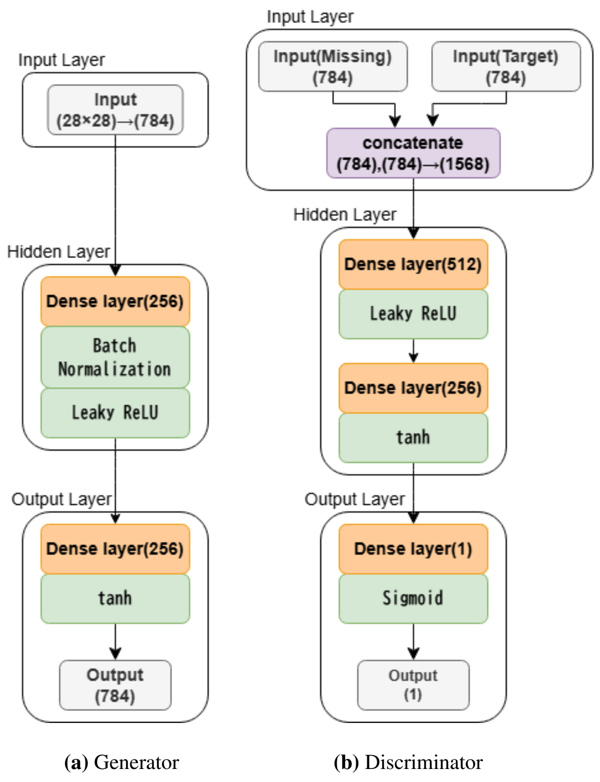
Considering the capacity limitations of the FPGA, we designed an architecture that reuses a ”16-input, 1-output” arithmetic unit through time-division multiplexing.

Furthermore, batch normalization in the intermediate layers of the generator involves division and square root operations, which increase circuit size. Therefore, following prior research [2], we fused the pre-trained mean μ, variance σ^2, and parameters γ, β into the weights $\mathbf{W}$ and bias b of the preceding fully connected layer to create new weights $\mathbf{W}'$ and bias b'.This allowed for inference on the FPGA using only simple multiply-accumulate operations ($\mathbf{W}'x + b'$) while incorporating the batch normalization process.

4 Experiments and Evaluation

4.1 Restoration Accuracy and SSIM Evaluation

Fig. 3 shows the result of inputting an image of the digit ”5” with a 50% missing rate into the pre-trained generator model on software. The images show, from left to right: the input missing image, the generated image, and the ground truth image. This confirms that the shape is appropriately



(a) Generator **(b)** Discriminator

Fig. 2: GAN model

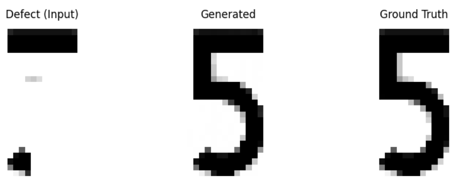


Fig. 3: Restoration results for digit ”5” at 50% missing rate

restored even when the characteristic strokes of the digit are missing.

To verify the performance of this system, we conducted a quantitative evaluation using SSIM, an index that indicates the structural similarity of images. Fig. 4 shows the average SSIM values calculated using 500 test images for each missing rate. The experimental results confirm that a high SSIM is maintained up to a missing rate of approximately 70%, enabling image restoration with practical accuracy.

4.2 Evaluation of the FPGA Model

Fig. 5 shows the restoration results obtained by performing hardware inference using the same input data as the experiment in Fig. 3. Although the hardware implementation includes minor errors due to fixed-point arithmetic, the SSIM was 0.977 (compared to 0.996 for software). This indicates that the hardware can achieve restoration capabilities equivalent to those of the software.

4.3 Execution Time Evaluation

Table 1 shows the inference time when executing with the digit ”3” at a 30% missing rate as input.

The measurements revealed that the hardware inference time was significantly longer than that of the software. This

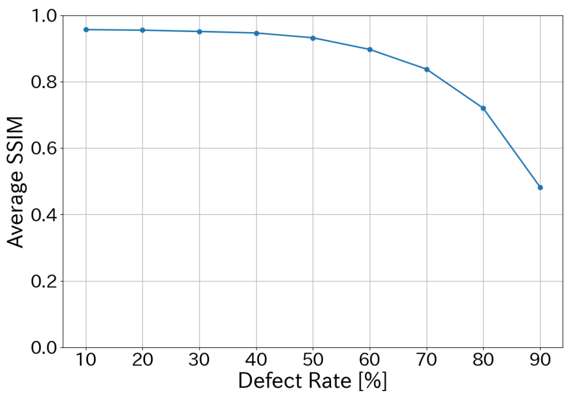


Fig. 4: Missing rate and average SSIM values

(a) SW generation result (b) HW generation result

Fig. 5: Comparison of digit ”5” at 50% missing rate

is due to communication overhead caused by adopting the PIO (Programmed I/O) method for data transfer between the CPU and FPGA. Furthermore, since the FPGA resource utilization remains at only a few percent, speedups are possible in the future by introducing DMA transfer and parallelizing the arithmetic units.

Table 1: Comparison of execution times

Execution Environment	Inference Time [ms]
Software	16.1934
FPGA	361.4970

5 Conclusion

We successfully constructed a GAN-based image completion model and implemented it on an FPGA. Regarding accuracy, we confirmed a high restoration capability equivalent to software, although challenges remain in data transfer speeds. Future work will include extending the system to target entire license plate images and improving the hardware-side parallelization and transfer methods.

References

[1] M. Arshid, M. R. Azam, and Z. Mahmood. A comparative study on detection and recognition of nonuniform license plates. *Big Data and Cognitive Computing*, Vol. 8, No. 11, p. 155, 2024.

[2] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2704–2713, 2018.

Real-Time Neural Audio Synthesis: A GAN-Based Waveform Generator on FPGA (Basys 3)

1 Introduction

This project presents a GAN-based audio waveform generator targeting a low-cost FPGA (Basys 3, $150). A two-layer generator network produces 64 audio samples per waveform period from a 4-dimensional latent vector. Eight timbres—sine, sawtooth, square, triangle, kick drum, snare, bass, and pluck—are trained offline and stored in BRAM for run-time selection.

The design is fully hand-coded in Verilog-2001 using Q8.8 fixed-point arithmetic. Audio output is provided through an 8-bit PWM DAC, and a UART interface streams waveform data to a PC visualizer.

2 System Architecture

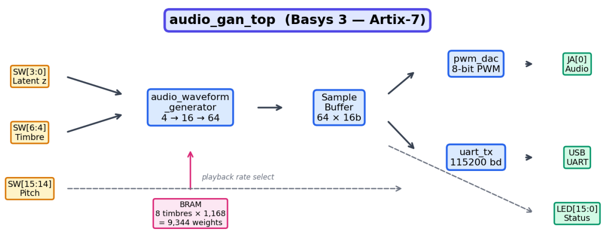


Figure 1: System block diagram on Basys 3.

The generator (`audio_waveform_generator`) implements a fully-connected MLP:

$$\boxed{\text{4 inputs}} \rightarrow \boxed{\text{16 hidden (ReLU)}} \rightarrow \boxed{\text{64 outputs } (\sigma)}$$

- **Fixed-point**: Q8.8 signed (16-bit), 32-bit MAC accumulators
- **ReLU**: sign-bit multiplexer (zero logic cost)
- **Sigmoid**: 256-entry combinational LUT, time-multiplexed across all 64 output neurons
- **Weights**: BRAM via `$readmemh`, 1,168 params per timbre, 9,344 total for 8 timbres (≈18.3 KB)

2.1 RTL Modules

- `audio_gan_top` – top-level controller, I/O mapping, 12-state FSM
- `audio_waveform_generator` – neural network inference engine with multi-timbre weight banking
- `pwm_dac` – 8-bit PWM DAC (390 kHz carrier at 100 MHz)
- `uart_tx` – 115,200 baud transmitter for PC waveform streaming

2.2 Audio Output Path

Generated samples are buffered and looped at a selectable fundamental frequency: 125, 250, 500, or 1,000 Hz. An external RC low-pass filter ($f_c \approx 15$ kHz) reconstructs the analog waveform from the PWM output.

2.3 I/O Mapping

Control	Function
SW[3:0]	Latent vector (16 timbral variations)
SW[6:4]	Timbre select (8 sounds)
SW[15:14]	Pitch (125–1000 Hz)
PMOD JA[0]	PWM audio output
USB-UART	Waveform data to PC
LED[15:0]	Status + VU meter

3 Training

A Python/NumPy golden model implements the full GAN training loop with backpropagation and BCE loss. Xavier initialization seeds all weight matrices. Each timbre is trained independently for 3,000 epochs against analytically generated target waveforms. Converged weights are quantized to Q8.8 and exported as hexadecimal `.mem` files for BRAM initialization.

4 Results

4.1 Waveform Accuracy (Golden Model)

Table 1 reports the MSE between target and generated waveforms after Q8.8 quantization in the Python golden model.

Table 1: Reconstruction MSE per Timbre

Timbre	MSE	Timbre	MSE
Square	0.0000	Kick	0.0055
Snare	0.0013	Sine	0.0084
Pluck	0.0049	Triangle	0.0108
Sawtooth	0.0120	Bass	0.0132

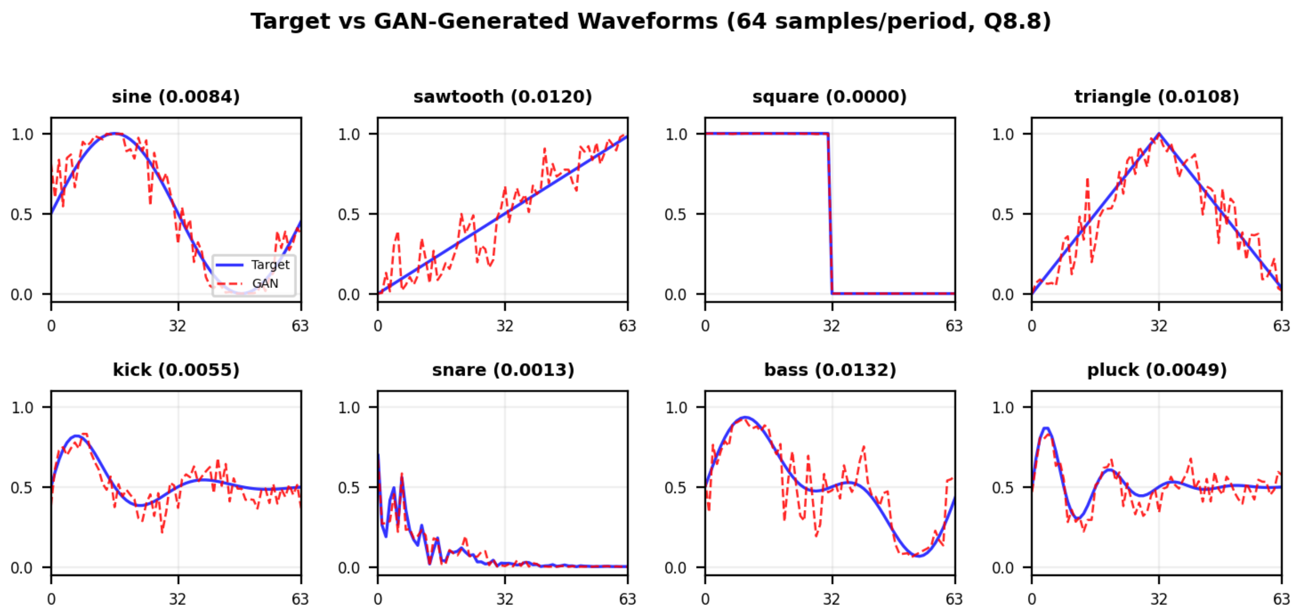


Figure 2: Target vs. GAN-generated waveforms (golden model, 64 samples/period).

The 29 th LSI Design Contest in Okinawa 2026

WGAN-GP を用いた通信挙動異常監視システム

チーム名：NANOJAPRIME

奥間　匠　　伊佐瑛斗　　親川悠莉　　町田駿弥

Communication Behavior Anomaly Monitoring System Using WGAN-GP

Takumi Okuma,　　Akito Isa,　　Yuuri Oyakawa，　　Shunnya Matida

Department of Advanced Electronics Information Technology for Production System，
Okinawa Polytechnic College

2994-2 Ikehara，　Okinawacity，　Okinawa，　904-2141，　Japan

Email address：　j242521305@okinawa-pc.ac.jp　　j242521302@okinawa-pc.ac.jp
j242521306@okinawa-pc.ac.jp　　j242521310@okinawa-pc.ac.jp

***Abstract*— This paper presents a communication behavior anomaly monitoring system based on WGAN-GP. Normal traffic feature distributions are learned using WGAN-GP, and the trained Critic model is implemented as an IP core on FPGA for high-speed inference. Network packets are captured and converted into feature data, which are processed in real time on the FPGA. By combining machine learning and hardware acceleration, the proposed system enables fast and efficient communication behavior monitoring.**

Keywords—WGAN-GP, FPGA implementation, IP core, packet capture, real-time processing, network anomaly monitoring

１．はじめに

近年，ネットワークサービスの拡大に伴い通信形態は多様化し，サイバー攻撃も高度化している．従来のシグネチャ型検知は事前に定義された攻撃には有効であるが，学習データに含まれない通信挙動への対応は困難である．本研究では正常通信分布を学習したWGAN-GP を用い，推論処理を FPGA 上で実行する通信挙動監視システムを開発する．

２．目標

2.1　WGAN-GP とは

WGAN-GP は通信特徴量の分布を学習する生成モデルであり，判別器の代わりに Critic を用いて入力データと学習分布との Wasserstein 距離を連続値で評価する手法である．この距離は入力データと学習分布との乖離を表す指標として利用される．また，Gradient Penalty により Lipschitz 制約を満たし，学習の安定性を向上させている．

2.2　WGAN-GP によるシステムの有用性

通信挙動は攻撃手法により多様に変化するため，攻撃パターンを事前定義する手法では未学習の攻撃や想定外の通信挙動への対応が困難である．正常通信分布を学習した WGAN-GP により観測通信との乖離を評価することで，攻撃種別に依存しない検知を実現する．

３．システム概要

本システムはユーザ PC と FPGA から構成される通信挙動監視システムである．ユーザ PC 上でパケットを取得し特徴量へ変換した後，シリアル通信によりFPGA へ送信する．FPGA 上では学習済み WGAN-GP の Critic により正常通信分布との距離を算出し，その結果をユーザ PC へ返送する．ユーザ PC では Python により境界値に基づく異常判定を行い，監視 UI に表示する．このように推論処理の一部をハードウェアにオフロードすることで，処理負荷を分散しつつ継続的な通信監視を可能としている．本システムは厳密なリアルタイム制御を目的とするものではないが，一定時間内での応答を想定したソフトリアルタイム処理を前提として設計している．

本システムの構成を図 1 に示す．

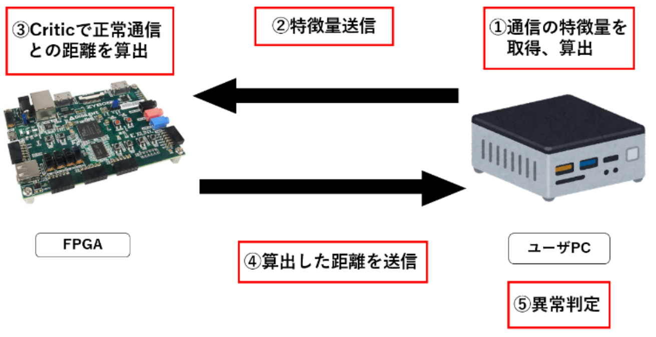


図 1　システム構成

４．開発環境

tshark によってネットワーク通信からパケット情報を取得し，MATLAB によって通信データの特徴量変換および WGAN-GP の学習・評価を行う．Vivado は評価ボードである ZYBO_Z7-20 での FPGA 回路を設計実装するために使用する．

また，評価ボードは ZYBO_Z7-20 を使用し，FPGA上で Critic による異常判別処理を行う．

本システムの開発環境を以下の表 1 に示す．

表 1　開発環境

OS	・Windows 11 - Pro
開発ソフト	・tshark ・MATLAB R2020b ・Vivado 2020.2
評価ボード	・DIGILENT 社製 ZYBO_Z7-20
言語	・MATLAB

5．WGAN-GP の構成

5.1　学習アルゴリズム

学習には CICIDS2017_improved データセットを用い，通信パケット情報から抽出された特徴量を入力とする．本システムでは評価用データセットの一例として CICIDS2017_improved を使用し，その中から通信量，通信間隔，パケット長など，通信挙動の変化を反映しやすい統計的特徴を中心に 9 種類の特徴量を選定した．これらは正常通信と異常通信の挙動差が現れやすく，かつ FPGA 実装を考慮した際に計算量を抑えられる点を基準として選定している．生成器は，正常通信データから得られた特徴量分布を学習し，正常通信の特徴分布を模倣した特徴量を生成する．一方，Critic は入力された特徴量が正常通信分布に従うかを評価し，Wasserstein 距離に基づいて正常通信との距離を算出する．入力特徴量 xに対する正常通信分布との距離 $d(x)$は，正常通信分布からの相対的な距離を表す指標であり，WGAN-GP における Critic D の出力として式(1)で定義する．

$$d(x) = D(x) \qquad 式(1)$$

本システムでは 正常通信のみを用いて学習を行い，正常通信データに対して算出された距離の分布を基に境界値 θを設定する．入力通信に対する異常 $A(x)$は，式(2)に定義する．

$$A(x) = \max(0,\ d(x) - \theta) \qquad 式(2)$$

この異常は正常通信からの逸脱量を表しており，値が大きいほど通信挙動の異常性が高いことを示す．図 2 に WGAN-GP の構成図を示す．

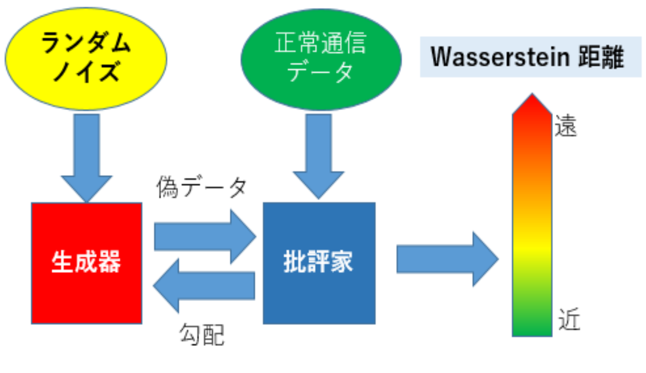


図 2　WGAN-GP 学習構成

6．FPGA 実装構成

6.1 FPGA の構成

本システムでは，WGAN-GP における推論処理のうち，Critic の積和演算部を FPGA（ZYBO_Z7-20）上に実装した．ZYBO_Z7-20 の Processing System（PS）と Programmable Logic（PL）を用いた HW/SW 協調設計構成とし，演算負荷の高いニューラルネットワークの計算を PL 側で処理する．

PC 側で算出された通信の特徴量は，シリアル通信によって FPGA に送られる．FPGA では，あらかじめ学習しておいたモデルのパラメータを用いて，入力された特徴量と正常通信の分布との「距離」を計算する．この距離が大きいほど，通常とは異なる通信であることを示す．

計算結果は再び PC 側へ返送され，そこで異常度への変換や表示を行う．本構成では，計算量の大きい部分を FPGA に担当させることで処理の高速化を図り，判定や表示などの柔軟な処理はソフトウェア側で行うことで，全体としてリアルタイムに動作する異常度評価システムを実現している．

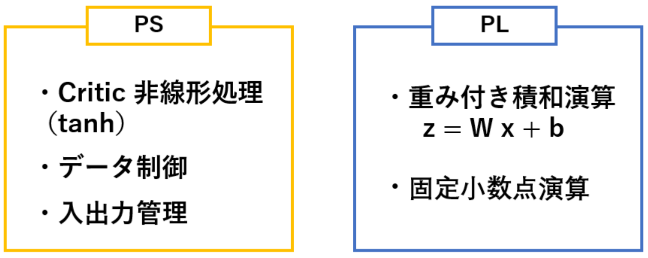


図 3　FPGA 実装構成

7．まとめ

本システムでは，正常通信のみを用いて学習した WGAN-GP を適用し，9 種類の通信特徴量に基づく通信挙動異常監視システムを開発した．Critic が算出する正常通信分布との距離を評価指標とし，正常通信の距離分布に基づいて境界値を設定することで異常判定を行った．また，Critic の線形演算部を FPGA 上に IP コアとして実装し，HW/SW 協調による推論処理を実現した．

実験の結果，本システムは特にポートスキャン攻撃に対して距離の増加が顕著に確認され，有効に判別できることを示した．一方で，他の攻撃挙動に対しては十分な分離性能を示すには至っていない．今後は，通信特性をより反映する特徴量の再選定を行い，検出性能の向上を図るとともに，活性化関数を含めたさらなるハードウェア化を検討する．さらに，処理の並列化や回路最適化を進めることで，リアルタイム監視への適用可能性を高め，実環境での長時間連続運用を想定したシステムへの改良が期待できる．

Quantized 16-bit 32×32 RGB GAN Generator on Low-end FPGA

1nd Muhammad Hafiz
School of Electrical Engineering
Telkom University
mhafiz@student.telkomuniversity.ac.id

2st Rega Arzula Akbar
School of Electrical Engineering
Telkom University
regaarzula@student.telkomuniversity.ac.id

3nd Ersha Anandita Larasati
School of Electrical Engineering
Telkom University
praetorsha@student.telkomuniversity.ac.id

***Abstract*—This final report presents a Cyclone V FPGA implementation of a quantized 32×32×3 GAN generator in fixed point 16-bit numerical format with VGA output on DE10-Standard. The design is fit for the hardware such as using banked weight ROMs, two-way parallel MAC execution, `altsyncram` Megafunction, and latency-aligned output readout. Testbenches that produces PPM images look correct, but on-board VGA still appears visually off, indicating a likely memory-latency alignment issue.**

***Index Terms*—FPGA, GAN accelerator, fixed-point arithmetic, VGA, Cyclone V**

I. Introduction

GAN deployment on low-cost FPGAs is attractive for deterministic edge inference, but mapping a convolutional generator is constrained by memory bandwidth and RAM read latency. This work designs memory organization around FPGA primitives to preserve both throughput and timing consistency between simulation and hardware displayed on VGA.

II. Proposed Architecture and Advantage

A. Generator Core

The generator core follows a four-stage upsampling convolution pipeline producing 32×32 RGB output. The design uses:

- **Two-phase MAC pipeline**: phase 0 issues addresses, phase 1 consumes valid data and updates accumulators.
- **Banked weights with two-way channel parallelism**: each convolution weight tensor is split into two banks so two input channels are processed per MAC step.
- **MLAB activation storage**: intermediate activations are mapped to MLABs to support required read access patterns.
- **Q6.10 (Fixed Point 6-bits integer and 10-bits fractional) saturating arithmetic**: all partial sums are quantized with explicit saturation on LUT based ReLU and tanh stages.

The main advantage is practical throughput improvement under memory-port constraints while also parallelizing by 2 (`N_PAR=2`) at the channel loop level provides substantial cycle reduction versus a scalar channel path which are under resource utilization constraints.

Listing 1. Core helper functions.

```
function integer fb_w1; input integer oc, ic, ky, kx;
  begin fb_w1 = (oc*(CH4/N_PAR)*9) + ((ic/N_PAR)*9) +
↪  (ky*3) + kx; end
endfunction

function signed [15:0] relu16; input signed [15:0] v;
  begin relu16 = v[15] ? 16'sd0 : v; end
endfunction

function signed [15:0] sat16; input signed [31:0] v;
  begin if (v > 32'sd32767) sat16 = 16'sd32767;
       else if (v < -32'sd32768) sat16 = 16'sh8000; else
↪  sat16 = v[15:0]; end
endfunction
```

Listing 2. Parallel MAC and out_ram readout alignment.

```
prod0 = $signed(z_mem0[ic>>1]) * $signed(W0_b0_q);
prod1 = $signed(z_mem1[ic>>1]) * $signed(W0_b1_q);
psum  = (prod0 >>> MAC_SHIFT) + (prod1 >>> MAC_SHIFT);

wire signed [15:0] out_q;
assign out_rdata = out_q;
altsyncram out_ram (
  .address_a(out_addr_mux), .clock0(clk),
  .data_a(out_wdata_mux), .wren_a(out_wren),
  .q_a(out_q)
);
// outdata_reg_a = "UNREGISTERED"
```

The generator FSM runs from 4 layers (`L0, L1, L2, L3`) with each layer having `INIT`, `MAC`, and activation function `RELU/TANH` and ends with `S_DONE`, giving deterministic stage latency.

B. Latency-Consistent Output Readout

An important contribution is fixing simulation–hardware mismatch at output RAM readout. The corrected implementation directly uses the RAM output wire in synthesis mode:

With Q6.10 operands, each multiplication produces a Q12.20 product (20 fractional bits). The design sets `MAC_SHIFT=10` so products are shifted back to Q6.10 scale before accumulation. Besides that, the readout contract is one-cycle memory latency from presented address to valid `out_rdata`. The VGA wrapper enforces this contract with `ST_RDPRIME` followed by `ST_READOUT`, so pixel index and channel boundaries (R/G/B) remain aligned to the same cycle model used by synthesis.

Listing 3. Wrapper helper functions.

```
function [15:0] lfsr_next;
  input [15:0] cur;
  begin lfsr_next = {cur[14:0],
↪   cur[15]^cur[13]^cur[12]^cur[10]}; end
endfunction

function [7:0] q6_10_to_pixel;
  input signed [15:0] val; reg signed [31:0] pv;
  begin pv = $signed(val) + 32'sd1024; if (pv < 0) pv =
↪   0;
      if (pv > 2048) pv = 2048; pv = (pv * 255) >>> 11;
      q6_10_to_pixel = (pv > 255) ? 8'd255 : pv[7:0];
↪   end
endfunction

z_wdata <= $signed({{4{lfsr[11]}}, lfsr[11:0]}) +
           $signed({{4{lfsr2[11]}}, lfsr2[11:0]});
```

Listing 4. Wrapper readout FSM and RGB channel mapping.

```
ST_RDPRIME: begin
  fb_r[0] <= q6_10_to_pixel(out_rdata);
  rd_addr <= 12'd1;
  rd_cnt  <= 12'd1;
  state   <= ST_READOUT;
end

if (rd_cnt < 12'd1024)      fb_r[rd_cnt[9:0]] <=
↪   q6_10_to_pixel(out_rdata);
else if (rd_cnt < 12'd2048) fb_g[rd_cnt[9:0]] <=
↪   q6_10_to_pixel(out_rdata);
else                         fb_b[rd_cnt[9:0]] <=
↪   q6_10_to_pixel(out_rdata);
```

III. DE10 VGA Integration

The top wrapper integrates:

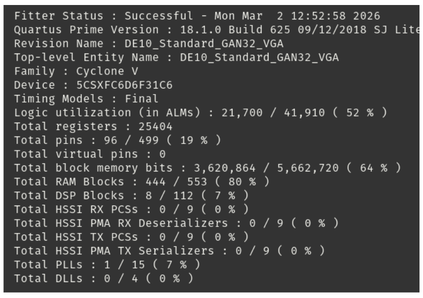
- 25 MHz GAN/VGA domain from PLL,
- random latent vector loading (`Z_DIM=64`) using dual LFSR summation,
- 3072-word sequential readout (R then G then B),
- Q6.10-to-8b conversion and 8× scaled centered VGA rendering.

Dual LFSR is used to generate a triangular-like latent distribution by summing two pseudo-random signed values.

The wrapper FSM is `ST_IDLE` → `ST_ZLOAD` → `ST_START` → `ST_WAIT` → `ST_RDPRIME` → `ST_READOUT`.

IV. Implementation Results

Timing closes across corners with worst setup slack 13.525 ns, worst hold slack 0.156 ns, TNS = 0, and 25 MHz-clock Fmax of 37.77 MHz (slow corner).



```
Fitter Status : Successful - Mon Mar  2 12:52:58 2026
Quartus Prime Version : 18.1.0 Build 625 09/12/2018 SJ Lite
Revision Name : DE10_Standard_GAN32_VGA
Top-level Entity Name : DE10_Standard_GAN32_VGA
Family : Cyclone V
Device : 5CSXFC6D6F31C6
Timing Models : Final
Logic utilization (in ALMs) : 21,700 / 41,910 ( 52 % )
Total registers : 25404
Total pins : 96 / 499 ( 19 % )
Total virtual pins : 0
Total block memory bits : 3,620,864 / 5,662,720 ( 64 % )
Total RAM Blocks : 444 / 553 ( 80 % )
Total DSP Blocks : 8 / 112 ( 7 % )
Total HSSI RX PCSs : 0 / 9 ( 0 % )
Total HSSI PMA RX Deserializers : 0 / 9 ( 0 % )
Total HSSI TX PCSs : 0 / 9 ( 0 % )
Total HSSI PMA TX Serializers : 0 / 9 ( 0 % )
Total PLLs : 1 / 15 ( 7 % )
Total DLLs : 0 / 4 ( 0 % )
```

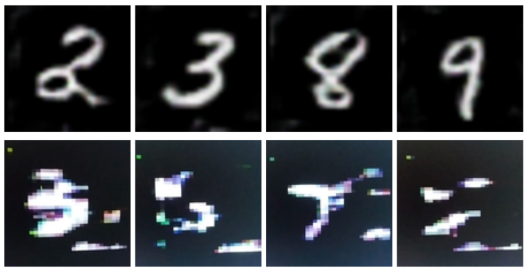


Fig. 1. Normal MNIST qualitative comparison (top: simulation outputs, bottom: hardware VGA captures).

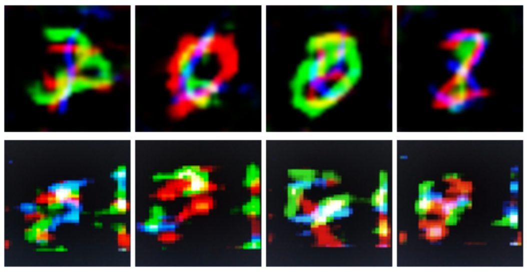


Fig. 2. Stacked-MNIST qualitative comparison (top: simulation outputs, bottom: hardware VGA captures).

A. *Throughput and End-to-End Latency*

The core requires about 10.5M cycles; at 25 MHz this is about 0.42 s per image (about 2.38 fps). Readout of 3072 samples takes about 123 μs, so end-to-end latency is compute-dominated.

B. *Simulation vs Hardware Observation*

A key experimental observation is:

- **PPM testbench simulation**: output images appear correct and stable.
- **On-board VGA**: visual output still appears off or inconsistent.

Functional simulation and timing closure alone are insufficient for guaranteeing image-correct hardware behavior in latency-sensitive display pipelines. It also motivates explicit cycle-accurate readout validation under the exact synthesized RAM behavior.

C. *Stacked MNIST RGB Capability*

To evaluate RGB channel capability, we also tested stacked-MNIST style outputs. Simulation preserves distinct color content, while on-board VGA still shows visible mismatch.

V. Conclusion

This report presented a practical FPGA GAN generator implementation optimized for Cyclone V constraints. The original advantage is the hardware-aware co-design of banked memory organization and parallel MAC scheduling, enabling a feasible 32×32×3 quantized generator with modest DSP usage and closed timing. Concrete compilation/timing results demonstrate implementability, while the simulation-hardware visual gap shows display correctness requires strict latency.

FETEL22: A High-Quality Integer-Only GAN Implementation on FPGA via Quantization and Hardware-Software Co-Design

Minh-Quang Nguyen, Minh-Thien Nguyen, and Thanh-Bao Ho
Faculty of Electronics and Telecommunications
University of Science, Ho Chi Minh City, Vietnam
Vietnam National University, Ho Chi Minh City, Vietnam
Email: {25C1505777, 25C4202122, 25C4200247}@student.hcmus.edu.vn

***Abstract*—While Generative Adversarial Networks (GANs) are essential for high-quality image synthesis, their deployment on resource-constrained edge devices is hindered by high parameter counts and extreme floating-point computational demands. In the LSI Design Contest 2026, we propose a system that reduces model size, eliminates floating-point operations, and controls memory traffic without sacrificing perceptual quality. Our approach introduces a 16-bit symmetric quantization scheme with 64-bit accumulation. To mitigate quantization degradation, we implement a Knowledge Distillation pipeline enhanced by Relative High-Frequency (RHF) tracking and the discriminator weakening strategy. The hardware architecture utilizes AXI4-Stream, centralized DMAs, and reconfigurable dataflows to maximize on-chip data reuse. We complete all the hardware IPs with high-performance algorithms such as Image-to-column, tiling, and line buffer. The results show that these are effective and compatible with the proposed C model. Our system reduces the model size by 50% (35MB to 18MB) and achieves a Kernel Inception Distance (KID) of 9.939, successfully producing high-quality 1024x1024 images natively on hardware.**

I. Introduction & Related Work

While high-quality image synthesis models [5] remain critical for latency-sensitive applications like real-time super-resolution, their reliance on massive parameter counts and intensive floating-point architectures restricts them to high-end GPUs. Deploying these models on hardware, in this case, FPGAs, requires aggressive quantization and custom data paths. However, mapping GANs [3] to hardware is exceptionally challenging because the generator's spatial upsampling creates severe memory bottlenecks, and standard integer quantization often introduces visible high-frequency artifacts.

Prior works have explored lightweight architectures like MobileStyleGAN [1] and Knowledge Distillation [4, 8] to reduce model complexity. However, standard post-training quantization is insufficient for GANs [3] due to the generator's extreme sensitivity to weight perturbations. In this work, we bridge the gap between algorithmic complexity and edge hardware constraints. We demonstrate that generative models can be executed efficiently using exclusively integer arithmetic by combining a specialized Quantization-Aware Training (QAT) pipeline with a reconfigurable, streaming-oriented hardware architecture, but still preserve high-quality details, as shown in Figure 1.

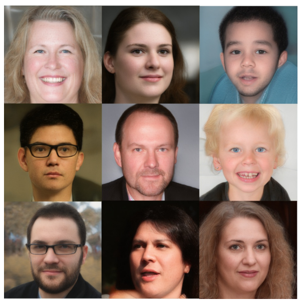


Fig. 1: The figure shows the results from our work.

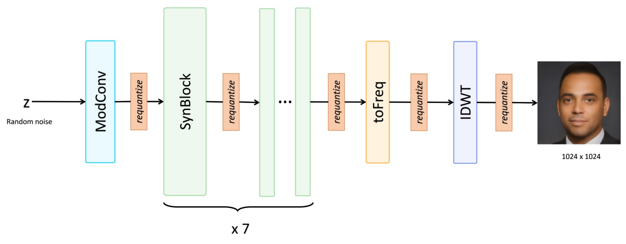


Fig. 2: The figure illustrates the high-level architecture of our model. From a random noise vector, we process it in the integer domain through the layers and output a 1024x1024 image. After each layer, we add a requantization module.

II. Method

A. Optimize model and extract golden data

To enable edge deployment, we eliminate costly floating-point operations by adapting the MobileStyleGAN network [1] into a fully integer-based framework with $x'_{out} = clamp\left(round(\frac{x_{out}}{scale}), q_{min}, q_{max}\right) \times scale.$

We adopt a 16-bit symmetric quantization scheme with PTQ and QAT EMA [7]. Generative tasks require 16-bit granularity to produce smooth gradients and realistic textures. The zero-point is fixed at zero, allowing MAC operations without offset corrections, saving significant DSP resources. Maintaining unclamped 64-bit partial sums prevents saturation and nonlinearities that corrupt spectral content in deep generators, casting to lower precision only at the final output stage using Round-to-Nearest, Ties-to-Even (RNTE) bitwise logic.

To preserve texture fidelity lost during quantization, we employ a **Quantization-Aware Distillation** [6] pipeline. We introduce a Relative High-Frequency (RHF) tracking metric to track the quantized student model and capture the spectral texture of the full-precision teacher.

We also propose a **Discriminator Weakening Strategy** to prevent the full-precision discriminator from overpowering the quantized generator during training. This includes using label smoothing to penalize overconfidence, enforcing a warm-up phase where discriminator weights are frozen, and maintaining a lower learning rate for the discriminator.

The trained model with the architecture, shown in Figure 2, is then converted to a **C model**. This model generates the golden data, consisting of intermediate feature maps and the quantized weights for the hardware validation.

B. Hardware Architecture

Directly instantiating all layers of a CNN on edge devices exhausts DSP blocks and on-chip memory. To address this, we adopt a reusable hardware architecture prioritizing flexible routing over structural replication.

1) Reconfigurable Dataflow & Streaming: The system routes data through a small set of parameterized blocks shared across layers, as shown in Figure 3. FIFO buffers isolate data streams between blocks, ensuring safe runtime reconfiguration by the MicroBlaze controller. Memory bottlenecks are eliminated using the AXI4-Stream protocol integrated with DMAs, which are centralized and reused multiple times during the running time.

2) Image-to-column and tiling: The format of the feature maps is $H \times W \times C$, allowing us to view it in the continuous memory with $(H \cdot W) \times C$. This is beneficial as we can convert the pointwise convolution to a matrix multiplication. To reduce the BRAM usage, we adopt tiling with tile size 16×16.

3) 3-line buffer and sliding window: Depthwise convolutions are implemented using a fully partitioned 3-line buffer and sliding window architecture, allowing parallel access to all 9 window elements in a single clock cycle.

4) IDWT Upsampling: We adopt the Inverse Discrete Wavelet Transform (IDWT) [2] using Haar wavelet filters. This reduces resource usage and explicitly manages high-frequency details to mitigate the computational cost of standard transposed convolutions.

III. Experimental Results

TABLE I: Quantitative comparison using the KID metric and the sizes of the models.

Model	KID ↓	Model's size ↓
MobileStyleGAN	**7.652**	35MB
Quantized MobileStyleGAN (Ours)	9.939	**18MB**

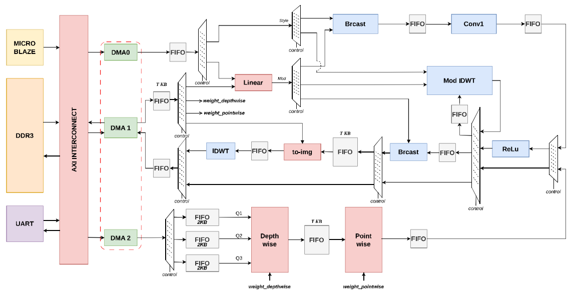


Fig. 3: This figure shows the hardware architecture of the proposed GAN accelerator. All FIFOs without explicitly specified depths have a default capacity of one data frame, which corresponds to 32 bits in this design. to_image is a pointwise convolution and modIDWT fuses Bcast with IDWT.

A. Model Compression and Output Quality

As shown in Table I, our approach reduces the size by nearly 50%, compressing the original from 35MB down to 18MB. Despite the aggressive 16-bit quantization, the model maintains KID of 9.939, validating that our RHF tracking and 64-bit accumulation effectively preserve perceptual quality.

TABLE II: Resource utilization and performance summary of HLS IPs on VC707.

IP	LUT	FF	DSP	Run-Time (ms)
Linear	1,248	1,060	5	0.72
Bcast	2,360	2,531	6	40.11
Idwt	1,794	2,363	4	220.81
Conv	2,550	2,648	5	148.37
Depthwise	4,262	4,442	22	266.48
Pointwise	6,470	4,622	20	592.05
ReLU	3,326	3,477	19	10.38

B. Hardware Resource Utilization

We maintained efficient and competitive DSP and LUT consumption as shown in the Table II as the model operates on the 16-bit integer domain. The average Fmax is approximately 275 MHz with IP small resources.

IV. Conclusion

This work successfully demonstrates high-resolution image synthesis on edge devices using strictly integer arithmetic. By combining a 16-bit symmetric Quantization-Aware Training pipeline, edge model techniques, such as image-to-column, line buffer, and a reconfigurable AXI4-Stream hardware architecture, we achieved a 50% reduction in model size with high-performance IPs, which are compatible with the proposed algorithm. while preserving structural integrity and perceptual quality. Future work will focus on optimizing FIFO buffer allocations to further reduce on-chip memory consumption, integrating the IPs, and evaluating the full system, making it more compatible with edge devices.

References

[1] Sergei Belousov. *MobileStyleGAN: A Lightweight Convolutional Neural Network for High-Fidelity Image Synthesis*. 2021. arXiv: 2104.04767 [cs.CV]. URL: https://arxiv.org/abs/2104.04767.

[2] Rinon Gal et al. *SWAGAN: A Style-based Wavelet-driven Generative Model*. 2021. arXiv: 2102.06108 [cs.CV]. URL: https://arxiv.org/abs/2102.06108.

[3] Ian J. Goodfellow et al. "Generative Adversarial Networks". In: *arXiv:1406.2661* (2014).

[4] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the Knowledge in a Neural Network". In: *arXiv:1503.02531* (2015).

[5] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE]. URL: https://arxiv.org/abs/1812.04948.

[6] Jangho Kim et al. *QKD: Quantization-aware Knowledge Distillation*. 2019. arXiv: 1911.12491 [cs.CV]. URL: https://arxiv.org/abs/1911.12491.

[7] PyTorch Team. *MovingAverageMinMaxObserver*. Version 2.4.1. 2024. URL: https : / / pytorch . org / docs / stable / ao_quantization . html # torch . ao . quantization . MovingAverageMinMaxObserver.

[8] Kiet Tran et al. "A Hardware-Friendly Approach for ECG Classification Based on Deep Learning Algorithm". In: July 2025, pp. 3–16. ISBN: 978-3-031-98160-9. DOI: 10.1007/978-3-031-98161-6_1.

Area-Efficient Unified Systolic Array for GAN-Based EEG Artifact Removal

Dharma Anargya Jowandy
Electrical Engineering
Bandung Institute of Technology
Bandung, Indonesia
13223075@std.stei.itb.ac.id

Rizmi Ahmad Raihan
Electrical Engineering
Bandung Institute of Technology
Bandung, Indonesia
13223051@std.stei.itb.ac.id

Aryo Wisanggeni
Informatics Engineering
Bandung Institute of Technology
Bandung, Indonesia
13523100@std.stei.itb.ac.id

***Abstract*—We present a unified systolic array architecture for GAN-based EEG artifact removal, capable of executing both 1D convolution and transposed convolution. Batch Normalization is fused into convolution weights, eliminating batch normalization hardware, with Q9.14 fixed-point quantization reducing computational complexity while preserving fidelity. On Pynq-Z1, the design achieves 100% DSP utilization, 33.93% BRAM usage, and 2.002 W on-chip power, with Pearson correlation exceeding 0.99 against floating-point software models.**

***Index Terms*—Generative Adversarial Network, Unified Convolution and Transpose Convolution Architecture, EEG Artifact Removal.**

I. Introduction

EEG is widely used for medical diagnostics, brain–computer interfaces (BCIs), and sleep analysis [1, 2], yet artifact contamination significantly degrades signal quality [3]. Manual denoising is unsuitable for real-time applications, and deep learning methods [4] are constrained by ambiguous ground truth, artifact variability, and high computational cost, limiting deployment on edge devices. To address these challenges, we propose a GAN-based temporal U-Net for EEG reconstruction and implement it on hardware for edge inference, adopting the EEGdenoiseNet methodology [5] to ensure consistent ground-truth construction. To further align model complexity with hardware constraints, we apply Batch Normalization fusion and fixed-point quantization [6], reducing computational and resource overhead while preserving signal fidelity.

II. Proposed Architecture

A. GAN Overview

EEG denoising is formulated as a conditional signal-to-signal reconstruction problem. Because EEG signals exhibit strong local temporal correlations, effective restoration requires localized analysis of neighboring samples. Generative Adversarial Networks (GANs) address the over-smoothing issue of conventional L1/L2 regression by modeling the conditional distribution of clean signals, where the adversarial objective preserves high-frequency components and realistic structures. Convolutional kernels enable efficient local temporal analysis with hierarchical feature extraction while using fewer parameters, making the approach suitable for hardware-efficient real-time deployment.

B. Generator Structure

By using a GAN-Based model, the generator structure will be distinct than seed-based traditional models such as [7]. This is because the generator needs to accept the noisy EEG input and fully denoise artifacts such that the output is clean EEG signal. To achieve this, compared to other decoder-only generator architecture [8], our proposed generator will also employ encoder to detect the patterns of the noisy input to be mapped to the clean EEG. The generator is composed of four primary components: encoder, bottleneck, decoder, and output adjustment.

The model uses computation following this equation for inference:

$$z = \sum_i w_i x_i + b \tag{1}$$

where w_i are the convolution weights, x_i are the input activations, and b is the convolution bias.

For the purpose of training using multiple discriminators, the term ”critic” will be adopted for the “n critic” training method. The model itself is relatively simpler than the generator, with an append module, encoder, and output adjustment.

Other than that, batch normalization fusion onto Convolution weights was done to reduce computation while taking advantage of the statics being frozen in inference. The fused operation can be expressed compactly as:

$$y = \sum_i \left(\frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} w_i \right) x_i + \left(\frac{\gamma(b - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta \right) \tag{2}$$

where w_i and b are the original convolution weights and bias, μ and σ^2 are the running mean and variance, and γ, β are the BatchNorm scale and shift parameters.

III. Hardware Architecture

A. Related Works

Several related works on hardware implementations of U-Net [9, 10] rely on High-Level Synthesis (HLS) to translate the computational model into hardware yet offer limited discussion on data flow organization, memory hierarchy, and resource reuse. Additionally, literature regarding temporal U-Net architectures supporting both 1D convolution and 1D transposed convolution remain scarce. To address this, we propose a unified architecture efficiently supporting both operations within a temporal U-Net framework.

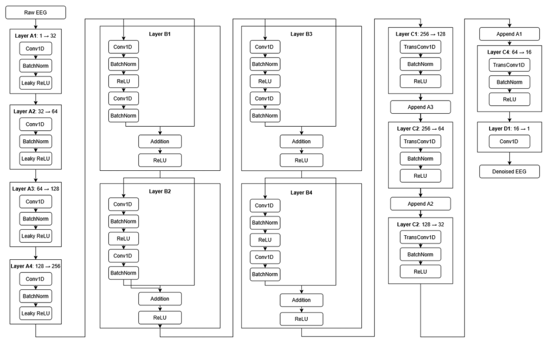


Fig. 1: Generator model

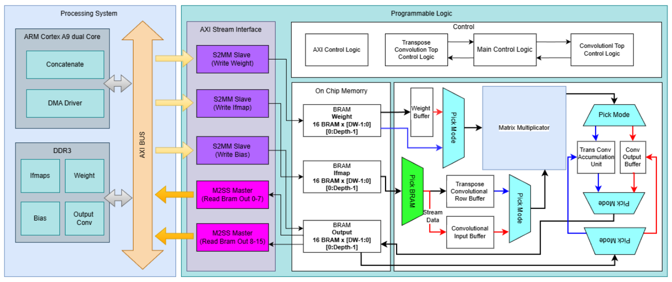

Processing System
ARM Cortex A9 dual Core
Concatenate
DMA Driver
DDR3
Ifmaps
Weight
Bias
Output Conv
AXI BUS
Programmable Logic
AXI Stream Interface
S2MM Slave (Write Weight)
S2MM Slave (Write Ifmap)
S2MM Slave (Write Bias)
M2SS Master (Read Bram Out 0-7)
M2SS Master (Read Bram Out 8-15)
Control
AXI Control Logic
Transpose Convolution Top Control Logic
Main Control Logic
Convolutionl Top Control Logic
On Chip Memorry
BRAM Weight 16 BRAM x [DW-1:0] [0:Depth-1]
BRAM Ifmap 16 BRAM x [DW-1:0] [0:Depth-1]
BRAM Output 16 BRAM x [DW-1:0] [0:Depth-1]
Weight Buffer
Pick Mode
Pick BRAM
Matrix Multiplicator
Transpose Convolutional Row Buffer
Convolutional Input Buffer
Stream Data
Trans Conv Accumulation Unit
Conv Output Buffer

Fig. 2: Unified accelerator architecture. The central FSM configures the systolic array for convolution (red), transposed convolution (blue), or shared paths (black).

B. Overview of Proposed Hardware Design

We implement the generator based on the output-stationary AXON systolic array [11], which supports both one-dimensional convolutional and transposed convolutional layers within a unified architecture as shown in Fig. 2.

Our system implements the U-Net architecture directly within the system control logic. As a result, no convolution nor transpose convolution parameters are given by PS. Consequently, the PS is only responsible to supply input data and recieve the output results in a deterministic order. The PS is not required to build ifmaps nor channel maps for both convolution and transposed convolution operations as these operations are done on PL. As such, the system lessens PS dependency.

IV. Implementation

TABLE I: FPGA Resource Utilization

Resource	Estimation	Available	Utilization (%)
LUT	47474	53200	89.24
LUTRAM	1993	17400	11.45
FF	37301	106400	35.06
BRAM	47.50	140	33.93
DSP	220	220	100.00
BUFG	1	32	3.13

For the entire system, timing is constrained at 20ns clock period. Results as follows:

TABLE II: Transpose Convolution Timing Analysis Summary

Metric	Setup	Hold
Worst Negative Slack (WNS)	0.648 ns	—
Worst Hold Slack (WHS)	—	0.014 ns

Power Consumption: **2.002W**, Mainly from PS7 (1.262W).

V. Experimental Result

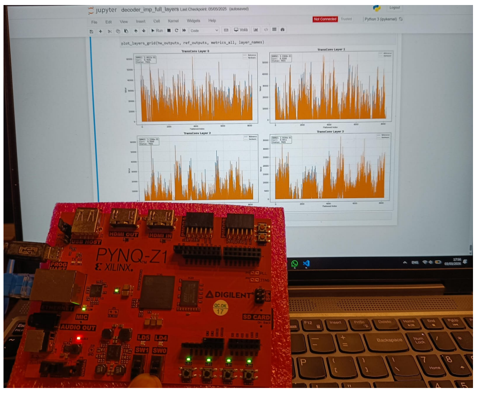


Fig. 3: FPGA Implementation of Transposed Convolution

At the time of reporting, the decoder module has been successfully implemented and functionally verified on the Pynq-Z1 FPGA platform. The hardware validation utilizes bottleneck-layer outputs generated by the corresponding software model as input stimuli. Table III presents a detailed comparison between the hardware implementation results and those obtained from the software reference model. The implementation is shown in Fig. 3.

TABLE III: Hardware vs Reference Comparison per Layer

Metric	Layer 0	Layer 1	Layer 2	Layer 3	Average
RRMSE	0.248120	0.333685	0.322061	0.291938	0.298951
Correlation	0.958084	0.928047	0.935992	0.941088	0.940803

VI. Acknowledgement

We'd like express our deepest gratitude to Nicslabs and STEI ITB for sponsoring our project and attendance in Japan.

VII. Conclusion

The proposed unified systolic array demonstrates that a single hardware core can support full temporal U-Net inference without two distinct hardwares. The architectural simplifications introduced (i.e., BN fusion and fixed-point quantization) improve edge-device application while preserving signal fidelity. Along with the mentioned validation metrics as shown in the table above, suggest that the implementaion was successful in mimicing software models. These results suggest that GAN-based EEG artifact removal is viable on resource-constrained devices.

References

[1] D. L. Schomer and F. H. Lopes da Silva, *Niedermeyer's Electroencephalography: Basic Principles, Clinical Applications, and Related Fields*. Lippincott Williams & Wilkins, 2011.

[2] J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, and T. M. Vaughan, "Brain-computer interfaces for communication and control," *Clinical Neurophysiology*, vol. 113, no. 6, pp. 767–791, 2002.

[3] J. A. Urigüen and B. Garcia-Zapirain, "Eeg artifact removal—state-of-the-art and guidelines," *Journal of Neural Engineering*, vol. 12, no. 3, p. 031001, 2015.

[4] X. Jiang, G.-B. Bian, and Z. Tian, "Deep learning for eeg signal classification: A comprehensive review," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 27, no. 9, pp. 1822–1831, 2019.

[5] H. Zhang, M. Zhao, C. Wei, D. Mantini, Z. Li, and Q. Liu, "Eegdenoisenet: A benchmark dataset for end-to-end deep learning solutions of eeg denoising," 2021. [Online]. Available: https://arxiv.org/abs/2009.11662

[6] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[8] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *International Conference on Learning Representations (ICLR)*, 2016.

[9] R. Ma, T. Hong, Z. Chen, and M. Kadoch, "U-net hardware acceleration design based on fpga," in *2023 International Conference on Information Processing and Network Provisioning (ICIPNP)*, 2023, pp. 435–439.

[10] J. Posso, H. Kieffer, N. Menga, O. Hlimi, S. Tarris, H. Guerard, G. Bois, M. Couderc, and E. Jenn, "Real-time semantic segmentation of aerial images using an embedded u-net: A comparison of cpu, gpu, and fpga workflows," 2025. [Online]. Available: https://arxiv.org/abs/2503.08700

[11] M. M. R. Nayan, R. Raj, G. B. Shaik, T. Krishna, and A. J. Naeemi, "Axon: A novel systolic array architecture for improved run time and energy efficient gemm and conv operation with on-chip im2col," 2025. [Online]. Available: https://arxiv.org/abs/2501.06043

SpikeGAN: An Energy-Efficient Spiking Generative Adversarial Network Design

Yuga Hanyu, Atharv Sharma and Aruki Komatsuzaki
School of Computer Science and Engineering, University of Aizu, Fukushima, Japan
Email: {m5291018, m5292015, s1310244}@u-aizu.ac.jp

I. Overview

Figure 1 shows the overall design flow. A compact ANN-based GAN model is trained on the MNIST dataset using 100-dimensional noise input and a three-layer generator consisting of 1200, 1200, and 784 neurons. Training was performed with 600 images per batch for 15 epochs. After training, only the generator network is converted to a Spiking Neural Network (SNN) to enable spike-based hardware execution.

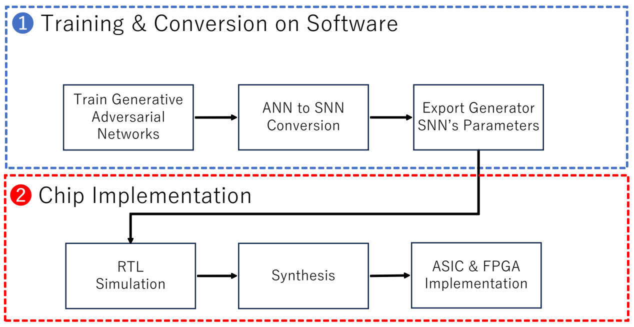

❶ Training & Conversion on Software
Train Generative Adversarial Networks
ANN to SNN Conversion
Export Generator SNN's Parameters
❷ Chip Implementation
RTL Simulation
Synthesis
ASIC & FPGA Implementation

Fig. 1. Overall design flow of GAN training and chip implementation

Figure 2 shows the GAN training and conversion from ANN generator and SNN generator. The ANN-to-SNN conversion preserves the original network structure while replacing ReLU activations with Integrate-and-Fire (IF) neurons. Weight normalization is applied using maximum activation per layer to maintain functional equivalence. Scalar inputs are encoded into spike trains, and neuron firing rates approximate ANN activations over multiple timesteps. This approach allows the SNN to reproduce the behavior of the trained ANN while enabling hardware-friendly spike processing.

II. Hardware Architecture

Figure 3 shows the architecture of chip implementation, and Figure 4 shows the computing core that stores the learned weights for each layer. Here, we did not utilize the STDP part, a learning mechanism for SNNs, since the learning was conducted by the source ANN model. The design consists of data readings for spike and weight for each layer, crossbar-based weight memory, arrays of IF neurons, and global control logic. Input spikes are multiplied by stored weights and accumulated into membrane potentials. When the membrane potential exceeds the threshold, an output spike is generated, and the potential resets. Layer-by-layer spike propagation

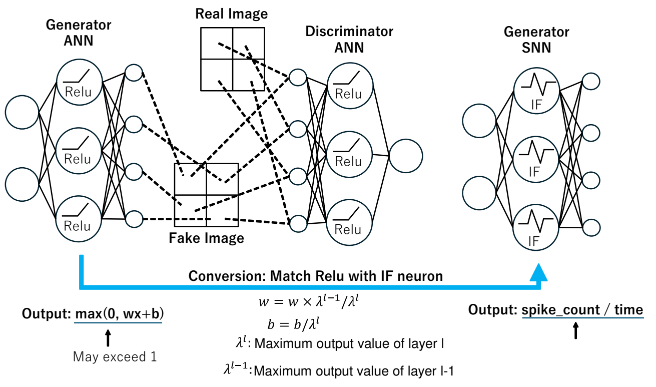

Generator ANN
Real Image
Discriminator ANN
Generator SNN
Relu
Fake Image
IF
Conversion: Match Relu with IF neuron
Output: max(0, wx+b)
May exceed 1
$w = w \times \lambda^{l-1}/\lambda^{l}$
$b = b/\lambda^{l}$
λ^{l}: Maximum output value of layer l
λ^{l-1}:Maximum output value of layer l-1
Output: spike_count / time

Fig. 2. GAN training and conversion from ANN to SNN

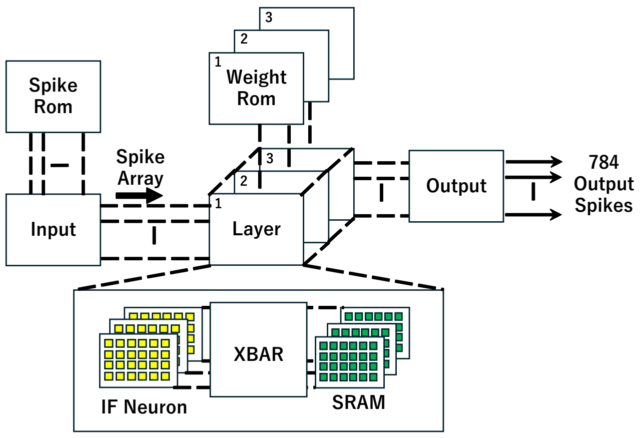


Fig. 3. Architecture of chip implementation

produces the final output image. The same RTL design is used for both FPGA implementation and ASIC synthesis.

III. Implementation Results

Once the converted SNN generator is obtained and runs successfully on the software, the model parameters, including the trained weights for each layer, are exported for RTL simulation. The model architecture of the original ANN generator is summarized in Table. I. The model architecture remains consistent for the ANN and SNN.

Fig. 5 compares the output images obtained from software and RTL simulation. The Mean Squared Error (MSE) value for (a) and (b) is 3.02e-05.

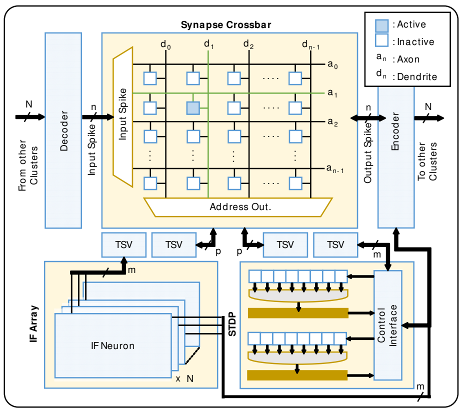
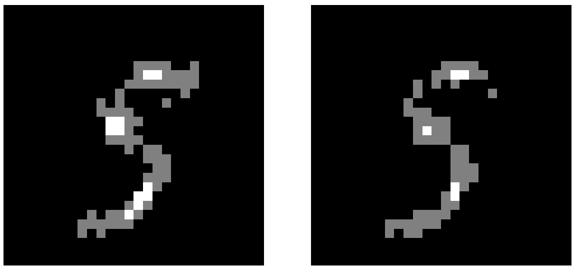


Fig. 4. Computing Core

TABLE I
MODEL ARCHITECTURE AND TRAINING PARAMETERS

Parameter	Value
Training Data	MNIST Handwritten Digits
Training Numbers	600 Images per Batch $\times$ 15 Epochs
Input Size	100
Layer 1	1200 Neurons
Layer 2	1200 Neurons
Layer 3	784 Neurons



(a) Software output 1 (b) RTL simulation output 1

Fig. 5. Output image comparison between software and RTL simulation

A. FPGA Implementation

The FPGA implementation was conducted using Vivado 2019.1 and Artix-100T FPGA evaluation board was used. Due to the memory constraint, the number of neurons in Layers 1 and 2 were changed from 1200 to 64 neurons.

Fig. II shows the resource utilization. The proposed design occupies 2,952 flip-flops (23.28% utilization) and 46,920 LUTs (74.01% utilization), on the targeted FPGA Chip.

B. ASIC Implemenation

In order to calculate the area consumption of GAN model Design Compiler Shell is used. Design Compiler Shell is command-line interface of Synopsys Design Compiler used to perform logic synthesis, where RTL descriptions are translated into optimized gate-level netlists based on timing, area, and power constraints.
A clock period of 2 ns is specified, and synthesis is performed using the CMOS PDK45 technology node.

TABLE II
RESOURCES UTILIZATION OF FPGA IMPLEMENTATION

Resource	Utilization	Available	Utilization (%)
LUT	46920	63400	74.01
LUTRAM	7296	19000	38.40
FF	29523	126800	23.28

TABLE III
AREA SUMMARY OF ASIC SYNTHESIS

Category	Area
Combinational logic	151,198.92 μm^2
Sequential logic	71,773.18 μm^2
Memories	485,376 bits
Total	222,972.10 μm^2

Table. III shows the area consumed by combinational logic, sequential logic, and memory, resulting to a total of around 222,972 μm^2 by combinational and sequential logic. For the total number of bits required by memory:

$$\text{Layer } 1 = 100 \times 64 = 6{,}400 \text{ bits} \tag{1}$$

$$\text{Layer } 2 = 64 \times 64 = 4{,}096 \text{ bits} \tag{2}$$

$$\text{Layer } 3 = 64 \times 784 = 50{,}176 \text{ bits} \tag{3}$$

Combining the bits of all these 3 layers, we get 485,376 bits in total for memory.

IV. CONCLUSION

This work presents SpikeGAN, a hardware-oriented framework that converts an ANN-based GAN generator into an SNN suitable for spike-based implementation. The proposed approach maintains output equivalence between software and RTL, achieves successful FPGA deployment, and demonstrates ASIC synthesis feasibility in 45 nm technology. The results indicate the potential of SNN-based generative models for energy-efficient AI hardware systems.

Hardware-Software Co-Design of a Quantized HiFiC GAN Accelerator for Learned Image Compression on FPGA

Anh Truong-Quoc*, Dung Do-Viet*, Phuc Nguyen-Duc*,
Dang Nguyen-Dac-Hai*, Minh Nguyen-Duc*†, Chi Hoang-Phuong*
*School of Electrical and Electronic Engineering, Hanoi University of Science and Technology, Vietnam
†Corresponding Author: minh.nguyenduc1@hust.edu.vn

***Abstract*—We designed a hardware-software co-design system for a HiFiC GAN accelerator that performs high-fidelity image compression on FPGA. By employing FP16 quantization for both weights and activations, we achieved a significant reduction in memory bandwidth while maintaining reconstruction quality. The system fits within the FPGA's resources and provides a faster computation time compared to software-based execution.**

***Index Terms*—FPGA, Convolutional Neural Network, HLS, GAN, Optimization**

I. Introduction

Learned image compression, particularly High-Fidelity Generative Image Compression (HiFiC) [1], produces perceptually superior reconstructions at low bitrates compared to traditional codecs[cite: 1471]. However, the immense computational complexity of GANs remains a barrier for deployment on resource-constrained edge devices[cite: 1472]. While FPGAs offer high parallelism, direct 32-bit floating-point (FP32) implementation results in prohibitive resource consumption[cite: 1473].

We propose a hardware-software co-design on the AMD-Xilinx ZCU104 platform using an **FP16** quantization strategy to halve memory bandwidth while preserving the dynamic range essential for GANs[cite: 1474]. By leveraging High-Level Synthesis (HLS) at **250 MHz**, our architecture achieves a **3.87× end-to-end speedup** over the ARM Cortex-A53 baseline[cite: 1475]. Remarkably, our hardware engine processes individual Residual Blocks nearly **5× faster** than a high-performance Intel Core i9 processor, proving its efficiency for generative compression on edge hardware[cite: 1476, 1477].

II. Post-Training Mixed-Precision Quantization (FP16) Methodology

Post-Training Mixed-Precision Quantization (FP16) is a retraining-free technique that maps tensors from FP32 to FP16 to reduce storage and increase throughput. This approach enforces FP16 arithmetic in the computational core while preserving FP32 semantics at system interfaces for compatibility.

A. Theoretical Formulation and IO Architecture

FP16 quantization is modeled as a projection onto representable binary16 numbers: $\hat{x} = Q_{16}(x) = \mathrm{fl}_{16}(x)$. While FP16 reduces memory bandwidth, it increases susceptibility to underflow/overflow due to a reduced exponent range.

To maintain system compatibility, the mixed-precision boundary mapping is expressed as:

$$x_{\text{in}}^{16} = \text{cast}_{32\to16}(x_{\text{in}}), \qquad x_{\text{out}} = \text{cast}_{16\to32}(x_{\text{out}}^{16}), \quad (1)$$

where x_{in} and x_{out} are the external FP32 input and output tensors. Internal operators process only FP16 data, allowing significant hardware acceleration without modifying existing pre/post-processing stages.

III. Proposed Architecture

A. Hardware-Software Design Flow and Partitioning

The HiFiC model is deployed using a co-design methodology where the system is partitioned based on computational workload. The Encoder and Hyperprior networks are executed in software on the Processing System (PS), while the Generator's nine residual blocks—the primary bottleneck—are offloaded to the Programmable Logic (PL) as a custom accelerator (Fig. 1).

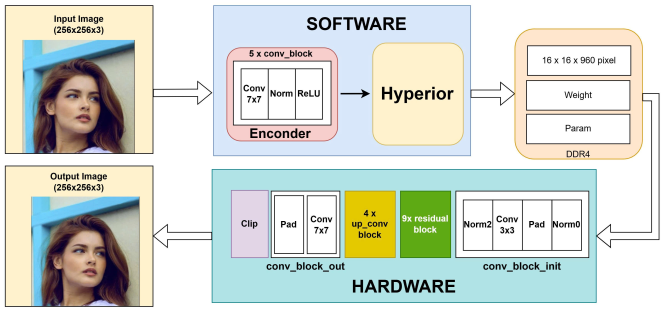


Fig. 1. System-level hardware-software co-design and design flow.

The interaction follows a coarse-grained model via shared DDR4 memory. The PS generates the latent representation ($16 \times 16 \times 960$) and configures the PL via AXI-Lite. The PL autonomously fetches data via AXI4-Master, performs pipelined computations, and writes results back to DDR.

B. *Pipelined and Parallel Convolution Architecture*

Our design utilizes a two-line buffer mechanism to stream feature maps in raster-scan order, enabling parallel window generation and maximizing on-chip data reuse. A dedicated Processing Element (PE) Cluster processes all valid convolution windows on a row simultaneously, mapping arithmetic operations directly to FPGA DSP blocks (Fig. 2).

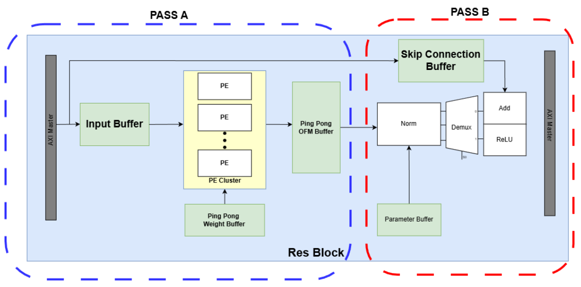


Fig. 2. PE Cluster architecture and two-pass execution strategy.

The architecture employs a two-pass execution strategy to hide memory latency: leftmargin=*

- **Pass A:** Focuses on convolution production, writing results to a Ping-Pong OFM buffer.
- **Pass B:** Concurrently reads from the buffer to perform normalization, ReLU, and residual addition.

To sustain a fully streaming dataflow, we utilize HLS pragmas (`PIPELINE II=1` and `UNROLL`), ensuring the innermost loops initiate a new operation every clock cycle while exploiting spatial parallelism across PEs.

IV. Experimental Results

A. *Quantization Results and Evaluation*

We evaluated the impact of transitioning from FP32 to FP16 on a 256 × 256 test image. As summarized in Table I, the compression performance of the HiFiC model is strictly preserved after quantization.

TABLE I
Comparison of FP32 and FP16 HiFiC Models

Metric	**FP32 Baseline**	**FP16 (Proposed)**	**Delta**
Bitstream Size (Bytes)	6,522	6,517	-5
Compression Ratio	3.97×	3.97×	0.00
PSNR (dB)	19.08	19.05	-0.03
SSIM	0.8315	0.8314	-0.0001

The degradation in PSNR (0.03 dB) and SSIM (-0.0001) is negligible and falls below the threshold of human perceptual visibility. The reconstructed images from the FP16 hardware accelerator are virtually indistinguishable from the FP32 baseline, validating that our quantization strategy effectively balances computational efficiency with high-fidelity output quality.

B. *Hardware Implementation and Performance*

The accelerator was implemented on the ZCU104 board, operating at a clock frequency of **250 MHz**. Table II details the post-implementation hardware resource utilization.

TABLE II
Resource Utilization on ZCU104 Platform

Resource	**Available**	**Used**	**Utilization (%)**
LUT / FF	230,400 / 460,800	57,291 / 73,774	24.86 / 16.01
BRAM Tile	312	105	33.65
DSP48E	1,728	48	2.77

Latency analysis indicates that the nine Residual Blocks constitute the primary bottleneck (88.17% of CPU workload). By offloading these blocks to the PL, execution time for the sequence is reduced to 193.5 seconds. The total co-design inference time is 357.3 seconds, yielding an approximately **3.87× speedup** over the ARM Cortex-A53 software execution (1384.8 seconds). Notably, the ResBlock processing speed is nearly 5 times faster than a high-performance Intel Core i9 processor.

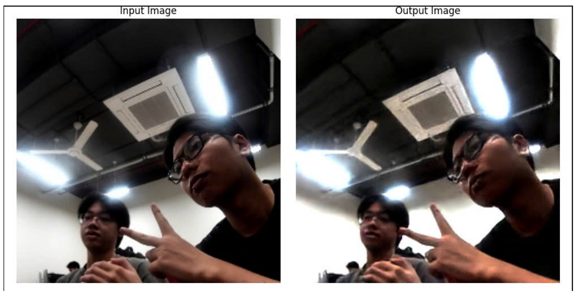


Fig. 3. Hardware demonstration of the HiFiC pipeline on the ZCU104 board.

Practical applicability was validated through a hardware demonstration: the system captures input via a USB 3.0 camera, executes the generative pipeline, and streams reconstructed images to an external monitor (Fig. 3).

V. Conclusion

We presented a hardware-software co-design for the HiFiC GAN model. By offloading intensive blocks to a custom FP16 engine, we achieved a 3.87× end-to-end speedup with negligible quality loss. Low resource utilization (e.g., 2.77% DSP) provides headroom for future parallelism and ultra-high-resolution support.

References

[1] F. Mentzer, G. Toderici, M. Tschannen, and E. Agustsson, “High-fidelity generative image compression,” *arXiv preprint arXiv:2006.09965*, 2020.

Implementation of GAN based Generator on FPGA for Voice Conversion with Batch Normalization Folding and Serialized MACs

Muhammad Farhan, Hifzhi Dinullah, Randy Revaldo Pratama, Infall Syafalni, Nana Sutisna, and Trio Adiono
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
Email: {13222114, 13222112, 13222012}@std.stei.itb.ac.id, {nsutisna, tadiono}@itb.ac.id

***Abstract*—This paper presents of an implementation for GAN based voice conversion generator on 80×80 mel-spectrograms inside an FPGA. The design integrates BN folding, dyadic residual scaling, channel-serialized MACs with 8-way DSP multiplexing, and a sync BRAM bottleneck reusing one convolution engine across three residual blocks. Brevitas 8-bit QAT [13] achieves NMSE = 0.16. Post-implementation on Zynq XC7Z020 at 100 MHz yields 2.271 W at 52.80% LUT and 80.45% DSP utilization.**

***Index Terms*—CycleGAN, voice conversion, FPGA, quantization-aware training, Brevitas.**

I. Introduction

Voice conversion (VC) modifies speaker identity while preserving linguistic content [1], enabling speech synthesis [2] and enhancement [3]. GAN-based methods CycleGAN-VC [5], StarGANv2-VC [6] achieve state-of-the-art quality [4] but are computationally prohibitive for edge devices. Prior FPGA GAN accelerators use zero-skipping [8] or memory tiling [9]; FastWave [10] and ROLIN [11] target voice synthesis. Non-autoregressive VC on low-cost FPGAs remains unexplored [7].

We propose an integer-only architecture integrating: (1) BN folding, (2) dyadic shift-add residual scaling, (3) channel-serialized MAC with 8-way DSP multiplexing, and (4) ping-pong BRAM bottleneck with single-engine reuse, using Brevitas W8A8 QAT [13].

II. Proposed Design

A. Model Architecture

The generator adopts an Encoder–Bottleneck–Decoder structure derived from CycleGAN-VC [5], processing 80×80 single-channel mel-spectrograms. Only the inference-only generator is implemented on-chip. Table I details the layer configuration.

B. BN Folding and Quantization

We fold BN parameters running mean μ, variance σ^2, learned scale γ, and offset β offline into the convolution bias [12]:

$$b_{\text{fused}} = \beta + \gamma\,(b_{\text{conv}} - \mu)\,/\,\sqrt{\sigma^2 + \epsilon} \tag{1}$$

TABLE I
Generator Architecture (80×80 Mel-Spectrogram Input)

Stage	Operation	K/S/P	Ch	Output
Encoder 0	Conv2D	3/2/1	$1\to32$	40×40
Encoder 3	Conv2D	3/2/1	$32\to32$	20×20
Bottleneck	ResBlock×3	3/1/1	32	20×20
Decoder 0	DeConv2D	3/2/1	$32\to32$	40×40
Decoder 3	DeConv2D	3/2/1	$32\to1$	80×80

Using Brevitas W8A8 QAT [13], the integer output is:

$$q_{\text{out}} = \text{Clamp}[(\textstyle\sum w\cdot x + b_{\text{int}}) \times M_{\text{int}} \gg s] \tag{2}$$

where b_{int} is the quantized fused bias, M_{int} is a 32-bit integer rescaling multiplier, s is the barrel-shift amount, $S_{\text{in}}/S_w/S_{\text{out}}$ are the Brevitas input/weight/output scales, α_{bn} is the folded BN gain, and $M_{\text{float}} = S_{\text{in}}\cdot S_w\cdot\alpha_{\text{bn}}/S_{\text{out}} \approx M_{\text{int}}\cdot 2^{-s}$ ensures bit-exact SW/HW correspondence.

C. Dyadic Residual Scaling

Residual rescaling uses zero-DSP shift-add: $x_{\text{aligned}} \approx \sum_{k\in\mathcal{K}}(x \gg k)$. Factors 0.50 and 0.75 map to $x\gg1$ and $(x\gg1) + (x\gg2)$.

III. Hardware Implementation

Fig. III shows the pipeline on the Zynq XC7Z020 (PYNQ-Z1) at 100 MHz. PS7 streams data via AXI DMA (32↔8-bit converters); all modules use AXI-Stream handshaking with 8-bit weights and 32-bit fused biases/multipliers.

CycleGAN generator hardware pipeline on Zynq XC7Z020.

Each Encoder/Decoder layer pipelines a stream padder, $(K-1)$-row BRAM line buffers (3×3 windows), and a serialized MAC with 8-way DSP time-multiplexing. The Bottleneck reuses one conv engine for 3 ResBlocks via sync BRAMs (PASS$_1$: conv A→B; PASS$_2$: dyadic residual add back to A), eliminating 5 redundant instantiations. The Decoder reuses Conv2D with zero-insertion upsampling and 180° kernel flip for transposed convolution.

The core operation of these convolutions lie on Fig 1, 2, 3. Fig 1 shows the line buffer architecture to form the window

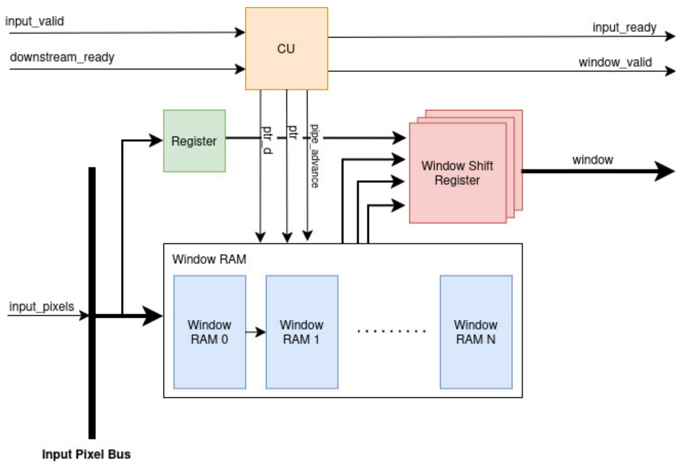


Fig. 1. Design of Line Buffer

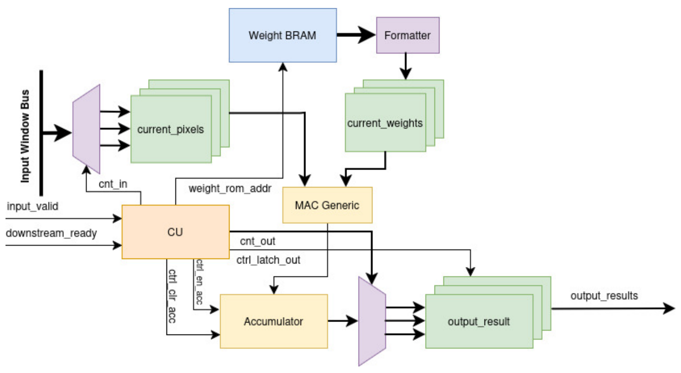


Fig. 2. Design of Serialized MAC

needed for the convolution operation, while Fig 2 shows the architecture that executes multiplication and accumulation using only one core of MAC, consisting of Kernel Size' DSP. While there is also padder and upsampler, those operations only control the data needed for the core operation between real input pixels or zero. Lastly, the bottleneck layer that consists of 3 convolution operations that have the exact architecure, are implemented with resource sharing, shown by Fig 3. It uses 2 BRAMs as the memory to store the input of the convolution, and the output of the convolution operation.

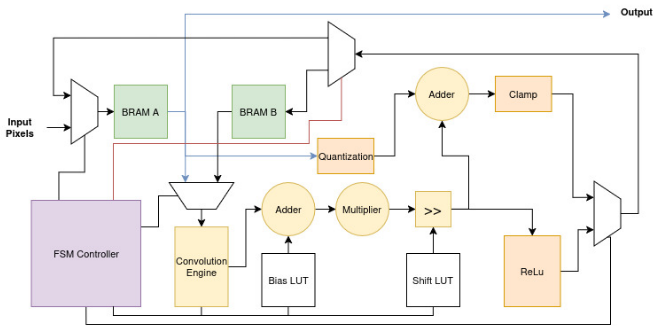


Fig. 3. Design of Bottleneck Layer

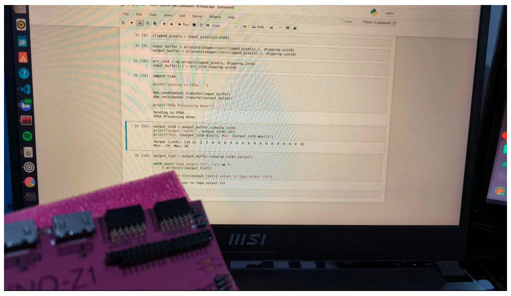


Fig. 4. FPGA deployment setup.

TABLE II
POST-IMPLEMENTATION RESULTS (ZYNQ XC7Z020, 100 MHZ)

Resource	Used	Available	Util. (%)
LUT / FF	28,092 / 34,919	53,200 / 106,400	52.80 / 32.82
BRAM	61	140	43.57
DSP48E1	177	220	80.45
Power: 2.271 W — Junc. Temp: 51.2°C (margin 33.8°C)			
WNS: 0.472 ns (setup) / 0.01 ns (hold), 0 failures / 95,335 EP			

IV. FPGA IMPLEMENTATION AND DEPLOYMENT

We implement the system on FPGA. The sound recorded by mic on host side, will be converted into mel spectogram and send it into FPGA. The result will be catch by the host system and then reconstruct using vocoder. The result we got approx 666.18ms/chunk, or thoughput 1.47 chunk/sec which 1 chunk 80x80. Fig. 4 shows the end-to-end setup. The host captures audio via microphone, extracts $80{\times}80$ mel-spectrograms, and streams each chunk to the Zynq PS through AXI DMA. The converted output is returned to the host for vocoder-based waveform reconstruction. Measured latency is 666.18 ms per chunk, yielding a throughput of 1.47 chunks/s.

V. RESULTS AND ANALYSIS

Experimental result shows that 8-Bit QAT on 231 unseen utterances yields NMSE$=$0.16 (σ=0.03), confirming fidelity under 8-bit quantization. Table II reports post-implementation utilization. The bitstream deployed on PYNQ-Z1 achieves *bit-exact* agreement with the PyTorch simulation, confirmed via PS-side Jupyter Notebook orchestrating AXI-DMA transfers.

VI. CONCLUSION

We presented an FPGA architecture for CycleGAN voice conversion integrating BN folding, dyadic scaling, serialized MACs, and ping-pong BRAM bottleneck. W8A8 QAT preserves quality (NMSE = 0.16); post-implementation on Zynq XC7Z020 at 100 MHz yields 2.271 W at 52.80% LUT / 80.45% DSP with timing closure (WNS = 0.472 ns), validating GAN deployment on low-cost FPGAs.

ACKNOWLEDGMENT

The authors would like to express their gratitude to NIC-SLAB and the School of Electrical Engineering and Informatics, Institut Teknologi Bandung, for providing the necessary support and research facilities.

References

[1] F. M. Mukhneri, I. Wijayanto, and S. Hadiyoso, “Voice conversion for dubbing using linear predictive coding and hidden Markov model,” *J. Southwest Jiaotong Univ.*, vol. 55, no. 4, 2020.

[2] C. Miao *et al.*, “EfficientTTS 2: Variational end-to-end text-to-speech synthesis and voice conversion,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 32, pp. 1650–1661, 2024.

[3] K. Byun *et al.*, “VC-ENHANCE: Speech restoration with integrated noise suppression and voice conversion,” *arXiv:2409.06126*, 2024.

[4] S. Dhar *et al.*, “GAN-based voice conversion: Techniques, challenges, and recent advancements,” *arXiv:2504.19197*, 2025.

[5] T. Kaneko and H. Kameoka, “Parallel-data-free voice conversion using cycle-consistent adversarial networks,” *arXiv:1711.11293*, 2017.

[6] Y. A. Li *et al.*, “StarGANv2-VC: A diverse, unsupervised, non-parallel framework for natural-sounding voice conversion,” in *Proc. Interspeech*, 2021.

[7] P. Xiyuan *et al.*, “A review of FPGA-based custom computing architecture for CNN inference,” *Chinese J. Electron.*, vol. 30, no. 1, pp. 1–17, 2021.

[8] A. Yazdanbakhsh *et al.*, “FlexiGAN: An end-to-end solution for FPGA acceleration of GANs,” in *Proc. DAC*, 2018, pp. 1–6.

[9] S. Liu *et al.*, “Memory-efficient architecture for accelerating generative networks on FPGA,” in *Proc. FPT*, 2018, pp. 30–37.

[10] S. S. Hussain *et al.*, “FastWave: Accelerating autoregressive CNNs on FPGA,” in *Proc. ICCAD*, 2019, pp. 1–8.

[11] J. Lee, J. Lee, and H.-J. Yoo, “A GAN training accelerator with selective layer retraining and reordering layers for instance normalization,” *IEEE J. Solid-State Circuits*, vol. 56, no. 10, pp. 2968–2980, 2021.

[12] J. A. Pineiro and J. D. Bruguera, “High-speed double-precision computation of reciprocal, division, square root, and inverse square root,” *IEEE Trans. Comput.*, vol. 51, no. 12, pp. 1377–1388, 2002.

[13] G. Franco, A. Pappalardo, and N. J. Fraser, “Xilinx/brevitas,” Zenodo, 2025, doi: 10.5281/zenodo.3333552.

General-Purpose Matrix Accelerator for Real-Time GAN Inference

Kai Kumano[1], Yushi Hasegawa[1], Mone Kasahara[1], Shosei Yabuki[2]
1: Graduate School of Engineering, Chiba University, Japan
2: Department of Electrical and Electronic Engineering, Chiba University, Japan
Email: kai.kumano@chiba-u.jp

Abstract

We propose "Matrix Accelerator", a high-frequency FPGA accelerator specialized for GAN inference, aimed at accelerating post-rendering processing in 3D game scenes. Convolutional and transposed convolutional operations are unified into matrix multiplications and implemented on the Xilinx Alveo U55C. We achieved stable operation at 600 MHz and realized parallel computation across 120 blocks through a hierarchical cache structure and spatiotemporal parallelization. Evaluation results confirm a performance improvement of up to 15.5 times compared to a high-end GPU (NVIDIA RTX pro-6000). A demonstration video is available at: https://youtu.be/zd_Zkgok_J8

Keywords- *Generative Adversarial Networks (GAN), AlveoU55C, Domain-Specific Architecture, Speculative Carry Propagation*

I. Introduction

In recent years, real-time 3D rendering has faced an increasingly pronounced trade-off between improved image quality and growing computational cost, as illustrated by a high-fidelity scene rendered in Unreal Engine (Fig. 1). Post-processing techniques that generate low-resolution images and subsequently enhance them using AI have proven effective, with GANs demonstrating strength in restoring high-frequency components[1][2]. However, GAN generators consist of numerous convolution operations, resulting in substantial computational demand and memory bandwidth requirements. To address this challenge, convolution operations are transformed into matrix multiplications, and a high-frequency, domain-specific architecture [3] is implemented on an FPGA to enable real-time inference.

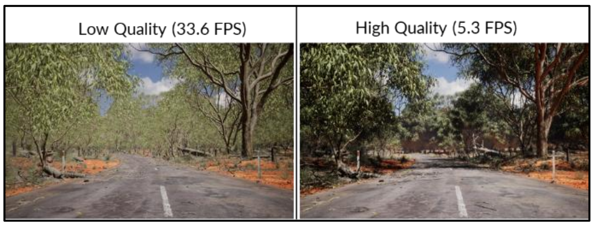


Fig.1 Example of a high-fidelity real-time 3D scene.

II. System Overview

The proposed system consists of a host PC and an Alveo U55C accelerator [4]. On the host side, feature maps and weights are transformed into matrix form using the image-to-column (im2col) method [5] and transferred to High Bandwidth Memory (HBM) via PCIe. Within the FPGA, a hierarchical memory architecture is adopted in which data are supplied from HBM to each computation Block through L3 and L2 caches. Fig. 2 illustrates the hardware configuration, including the Super Logic Region (SLR) structure employed. An L3 cache implemented with UltraRAM is placed within each SLR, while BlockRAM-based L2 caches and multiple computation Blocks are implemented in each die. By replicating this structure across multiple SLRs, a total of 120 Blocks operate in parallel.

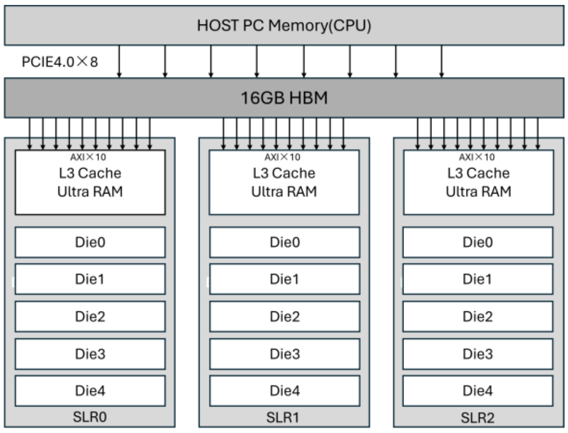


Fig.2 Hardware Architecture.

III. Hardware System

The proposed Matrix Accelerator is designed for high-frequency operation and achieves stable performance at 600MHz. Each Block is responsible for 16×16 matrix multiplication and contains 16 Units with a total of 64 Cores internally.

A. Speculative Carry Propagation

To avoid the wiring delay of multiplexers in a variable-precision adder, Carry8 primitives are directly connected to propagate the carry speculatively to the most significant stage, and the result is determined afterward using a logical mask. This approach enables stable high-frequency operation.

B. Spatiotemporal Convolution

As shown in Fig. 3, instead of increasing the computation circuitry spatially (i.e., by enlarging the area), the design expands the computation along the temporal axis through time-division processing, thereby minimizing data movement while achieving high parallelism. Furthermore, by adopting a four-Tile architecture and introducing phase-shifted control in which the computation start timing of each Tile is staggered by one clock cycle, data hazards are physically avoided and computational resources requiring four clock cycles are utilized with high efficiency.

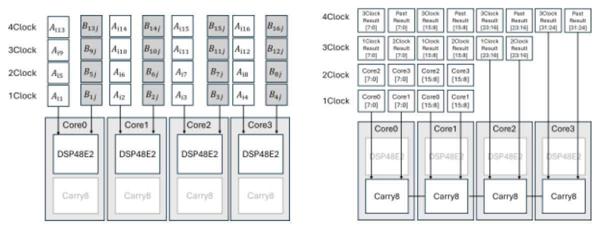


Fig. 3 Multiplication (left) and addition (right) units inside the Core.

C. Hierarchical Cache Architecture

As illustrated in Fig. 4, the cache architecture is organized into four hierarchical levels L3, L2, L1, and registers (REG) to promote the reuse of weights and feature maps and thereby reduce the frequency of HBM accesses. By integrating these spatial and temporal optimizations into a unified pipeline and minimizing control overhead, the system successfully maximizes overall throughput without stalling the 600 MHz computational performance.

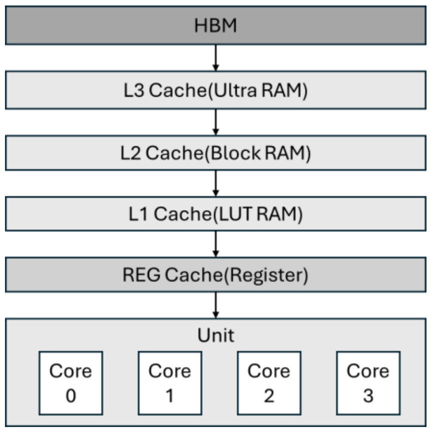


Fig. 4 Hierarchical cache architecture.

IV. Implementation

The accelerator was designed at the RTL level, and control was implemented using XRT. By overlapping data loading from HBM to the L3 cache with computation, data transfer and processing are executed in parallel. The architecture was optimized to maximize overall system throughput, achieving stable operation at 600 MHz even under worst-case conditions.

V. Evaluation

Performance measurements were conducted in the established evaluation environment (Table.1). With 120 blocks operating in parallel, high throughput was confirmed not only in theoretical performance but also in measured values. As shown in the table, this architecture achieved an effective performance improvement of approximately 15.5 times compared to a high-end GPU. While existing GPUs waste about 99% of their processing time on data movement and preprocessing (such as im2col), this result is due to our design completely hiding this overhead through a hierarchical cache structure and spatiotemporal parallelization. Through this logic-free data flow, we have demonstrated the true nature of computational efficiency in the post-GPU era.

Table.1 Inference Time Comparison (input: Full HD).

Device	Processing time [s]	im2col [s]
Our Accelerator	**0.0924**	**-**
EPYC 9755 (16-core)	0.2313	1.3472
RTX PRO 6000	0.0136	1.4361

VI. Conclusion

We have designed a high-frequency, matrix-arithmetic-specific FPGA accelerator and applied it to GAN inference. We demonstrated high-throughput and high-efficiency inference processing through an SLR-based large-scale parallel structure, speculative carry propagation, spatiotemporal parallelization, and hierarchical cache architecture.

As future work, we plan to fully implement the variable-precision architecture, particularly extending it from the adder to the multiplier to enhance computational flexibility. In addition, we will further develop the spatiotemporal design concept toward a more dataflow-adaptive architecture that fundamentally minimizes data movement.

References

[1] Goodfellow, et al., "Generative adversarial networks," Communications of the ACM, Vol. **63**, No. 11, pp. 139-144, 2020.
[2] C. Ledig, et al., "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," CVPR, 2017.
[3] J. L. Hennessy and D. A. Patterson, "A New Golden Age for Computer Architecture," Communications of the ACM, 2019.
[4] AMD Xilinx, "Alveo U55C Data Center Accelerator Cards Data Sheet (DS978)" v1.**20**, 2024.
[5] Y. Jia, "Learning Semantic Image Representations at a Large Scale," Ph.D. dissertation, UC Berkeley, 2014

また来年沖縄で、お会いしましょう。

OKINAWA

コンテストに関してのお問合せ：
九州工業大学情報工学部情報・通信工学科尾知研究室内
LSIデザインコンテスト実行委員会事務局

TEL：0948-29-7667
http://www.lsi-contest.com