SNN-Based Image and Audio Steganography with Encryption

Biswajit Kumar Sahoo, Dr. Pedro Machado, and Dr. Srinivas Boppu

Abstract

We propose a hardware-efficient Steganography framework that conceals audio data within PNG images using an encoding mechanism inspired by Spiking Neural Networks (SNNs). Each audio sample is digitized into decimal digits, mapped to spike patterns generated from Leaky Integrate-and-Fire neurons, and then encrypted by selecting a spike position whose modulo-16 remainder serves as the cipher. A 4-bit key identifies the spike index to guarantee unambiguous decryption. The encrypted digits are embedded into the two least significant bits of each RGBA channel, yielding a payload of 8 hidden bits per pixel while maintaining imperceptible visual quality. To suppress visual artifacts, a lightweight dithering stage introduces controlled noise before embedding. The system was implemented on a PYNQ-Z2 platform, integrating custom IP blocks with AXI DMA for high-throughput data transfer. Results demonstrate real-time operation, exact audio reconstruction, and higher embedding capacity than conventional LSB approaches, highlighting the feasibility of combining SNN-inspired encoding with FPGA acceleration for secure multimedia Steganography.

1 Introduction

Spiking Neural Networks (SNNs) [7] are brain-inspired models that process information using discrete spikes instead of continuous values. In this work, we use spike patterns generated from Leaky Integrate-and-Fire neurons in the NEST simulator [4] as the basis for encrypting audio data before embedding it into images. Steganography [3] is the art of concealing data inside media such that its presence is not obvious. Our objective is to develop a hardware-friendly Steganography system that embeds audio data into images, with emphasis on FPGA implementation for real-time performance. In our method, audio samples are digitized, mapped to SNN spike patterns, encrypted, and embedded into the two LSBs of RGBA channels of each image pixel. Each pixel carries 8 bits of hidden data, a higher payload compared to typical LSB methods [8] , where only three to five

bits per pixel are stored while maintaining imperceptibility. We used the PYNQ framework [1], which provides Python APIs and interfaces for hardware acceleration. A custom overlay [2] was built that included AXI DMA engines and our encryption/decryption IP cores, allowing high-level Python code to interact seamlessly with the FPGA fabric.

The NEST simulator [4], a widely used simulation framework for spiking neural networks, is used in our work. By configuring Leaky Integrate-and-Fire (LIF) neurons [5] with constant input currents[1] in NEST, spike trains were generated for each digit such that the number of spikes is equivalent to the digit. Furthermore, each digit (1–9) was assigned a unique spike position (time stamp) within the spike train such that the result of modulo 16 is unique for each spike position and non-zero. Since digit 0 has no spikes, it is not encrypted.

These biologically inspired spike patterns serve as the basis for the encryption scheme, and distinct spike patterns can be generated based on the parameters of the neurons.

2 Method Overview

2.1 Audio Digitization and SNN Mapping

Each audio sample (amplitude range:=$32,768$ to $+32,767$) is treated as sign plus number of digits. For instance, if amplitude is $+12,345$, then the digits will be $0, 1, 2, 3, 4, 5$ (0 for positive and 1 for negative numbers). Moreover, each decimal digit (1–9) corresponds to a fixed spike pattern, i. e., a list of spike positions between 0 and 60 time steps, if simulation time = 60. Digit 0 has no spikes and is left unencrypted. For instance, consider the digit 6; it will have 6 spikes, and exemplary spike positions are: $20, 28, 36, 44, 52, 59$. Similarly, there will be spike patterns for other digits as well.

2.2 Per-Pixel Payload and Capacity

RGBA images have four 8-bit channels. By overwriting the lowest 2 bits of each channel, we embed 8 hidden bits per pixel (1 byte). Two BCD digits are stored per pixel, and a single signed sample (sign and 5 digits) requires 3 pixels (24 hidden bits). This payload is higher than other standard LSB Steganography (often <= 5 bits/pixel) [8], making our method more storage-efficient while still imperceptible. By more storage capacity, we mean more payload capacity.

[1]The currents are chosen such that the number of spikes for each digit is equal to its mathematical value, so zero(0) has no spikes.

Tab. 2: Lists the spike patterns for each digit, chosen spike position for each digit, and its remainder with modulus 16, as well as its index

Digit	**Spike Positions in Spike Pattern**	**Chosen Spike Position/ time-stamp**	**Remainder with modulo 16 (map value)**	**Key**
0	NA	NA	0	NA
1	59	59	11	0
2	39, 59	39	7	0
3	31, 45, 59	45	13	1
4	26, 37, 48, 60	37	5	1
5	23, 32, 41, 50, 59	50	2	3
6	20, 28, 36, 44, 52, 59	44	12	3
7	18, 25, 32, 39, 46, 52, 59	52	4	5
8	17, 23, 29, 35, 41, 47, 53, 59	41	9	4
9	15, 21, 26, 32, 37, 43, 48, 54, 59	54	6	7

2.3 Dithering for Visual Quality

To prevent visible artifacts after LSB replacement, a low-level pseudo-random noise (0–2) is added to each channel before LSB replacement. This ensures gradients remain smooth and image quality is preserved.

2.4 Encryption

The Fig. 1 shows a graph between current and number of spikes for a single neuron. Here, each step (level) may represent a digit that can have a range of input current values as shown in Fig. 1. We choose any current value in the range associated with that level, and then simulate the neuron for 60 time-steps to generate the spike patterns. Now, one spike position (time-stamp) is chosen from each digit's spike pattern set such that the modulo 16 is unique for each such chosen spike position and the remainder is non-zero. The result of the modulo 16 is a 4-bit encrypted value of that digit. Digit 0 is left unencrypted. A 4-bit KEY, representing the index of the selected spike position within the digit's pattern, is also generated to ensure correct decryption. For instance, the chosen spike position for 6 is 44, then it is at the third position (starting from zero) in the spike pattern of $20, 28, 36, 44, 52, 59$. Therefore, the KEY for the digit 6 is 3. Thus, each digit is represented as (remainder, KEY) pairs, while two digits (8 bits) are packed into one pixel's LSBs. The KEY is sent before sending the data to the receiver. Please refer to Tab. 1 for the remainder and KEY pairs of each digit.

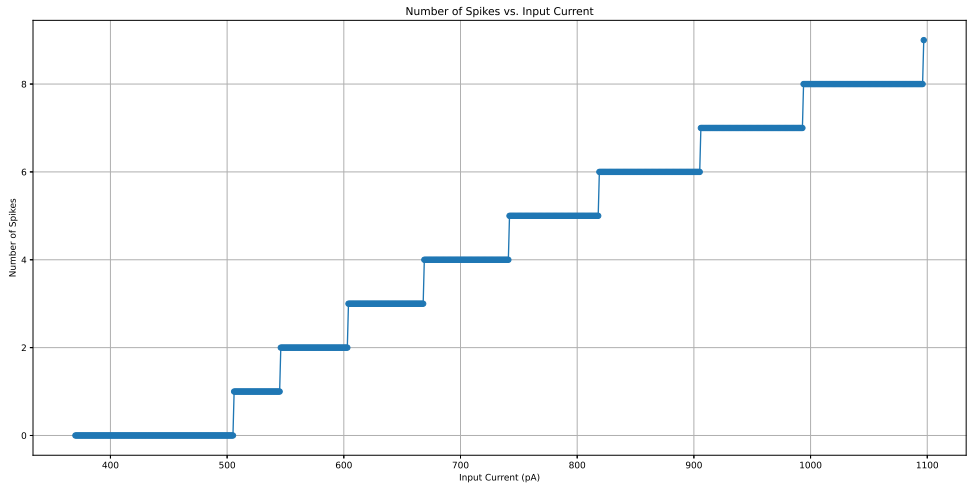

Fig. 1: Spike patterns generated using the NEST simulator with LIF neurons.

2.5 Decryption

At the receiver, the encrypted value and KEY are used with the original spike patterns to unambiguously reconstruct digits, enabling lossless audio recovery. The decrypter should know the exact neuron model with parameters used so that it can regenerate spike patterns, which are needed for decoding. Once the image is received, all the embedded remainders in the image need to be extracted. Let us say, if the extracted remainder is r, then possible spike positions $\{r, r{+}16, r{+}32, r{+}48\}$ in the range of $0{-}60$ (simulation time) are generated to identify which digit's spike pattern contains one of these positions. Now, the KEY (ordinal index of the spike in that digit's pattern) is used to resolve ambiguities and identify the correct digit corresponding to an encrypted remainder. In this way, scanning through all the image pixels and decrypting reminders to the original digits, the audio data embedded in the image can be reconstructed. The digit 0 passes through unchanged.

3 FPGA implementation

Our design was implemented on the PYNQ-Z2 board, which features a Xilinx Zynq-7020 SoC with ARM Cortex-A9 Processing System (PS) and Programmable Logic (PL).

3.1 NEST on PYNQ-Z2

To generate spike patterns, the NEST simulator was built from sources directly on the ARM of the PYNQ-Z2. This process is required to resolve dependencies and customize build flags due to limited memory and package availability on armhf. After successful

build and installation, NEST was used to run LIF neuron models on the board itself, ensuring consistency between software simulation and hardware embedding. This makes the platform self-contained, eliminating dependency on external PCs for spike pattern generation.

3.2 Image Resolution Support

Currently, our design supports 1920×1080 (Full HD) images in PNG format having 4 channels (RGBA). While this is sufficient for the project, higher-resolution support (e. g., 4K) would require DDR bandwidth optimization and larger buffer management.

3.3 Dataflow Architecture

PS (ARM) runs Python code in Jupyter notebooks, manages DDR buffers, invokes NEST, and orchestrates transfers. PL (FPGA fabric) contains custom IP cores connected via AXI4-Stream and AXI4-Lite. AXI DMA handles the streaming of audio digits and image pixels between DDR and the IP blocks. Fig. 2 shows the block diagram of our implementation, illustrating the dataflow.

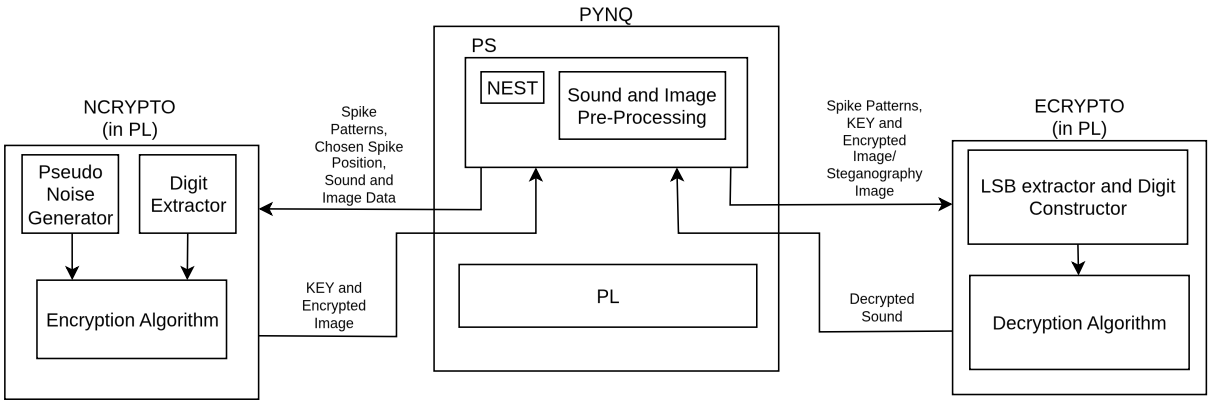

Fig. 2: Architecture of the proposed system implemented on PYNQ, illustrating the encryption (PL), pre-processing and NEST-based processing (PS), and decryption (PL) modules, along with data exchange of spike patterns, keys, and images.

3.4 Key IP Components

Encryptor has a *Digit Extractor* based on the Binary-to-BCD (Double Dabble [6]) algorithm that efficiently converts binary audio samples into decimal digits suitable for mapping. An *Encryption Logic block* then implements the mod-16 remainder mapping and 4-bit KEY generation derived from pre-computed spike patterns, ensuring secure representation of each digit. A lightweight *Pseudo Noise Generator* introduces bounded random noise (0–2) into each channel to preserve visual quality and prevent banding

artifacts. The *LSB Embedder* masks the two least significant bits of each RGBA channel and inserts the encrypted data, achieving a payload of 8 hidden bits per pixel.

The decryptor IP is based on the algorithm discussed in Section 2.5. Within the IP, the *LSB extractor and Digit Constructor* block retrieves 4 bits to construct each digit, which is then passed to the *Decryption Algorithm* block. In this block, the possible spike positions are determined, and the KEY is applied to recover the original digit. After processing six such digits (the sign bit and five numerical digits), a complete sound sample is reconstructed.

4 Results and Discussion

The Fig. 3 is one of many images taken from the internet for testing our design.

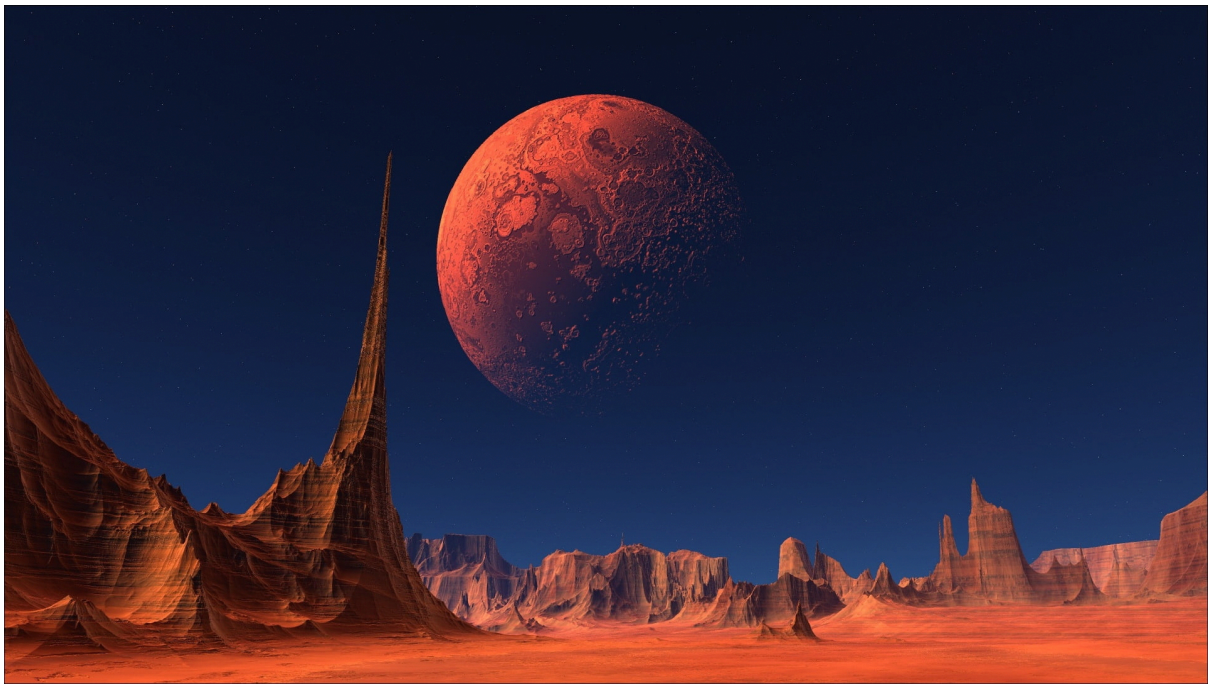

Fig. 3: Original Image used for encryption (source: Internet).

The Fig. 4 is the output of our algorithm, which has the sound embedded in it. It can be seen that there is no visual difference between this and the original image Fig. 3 and the encrypted image Fig. 4.

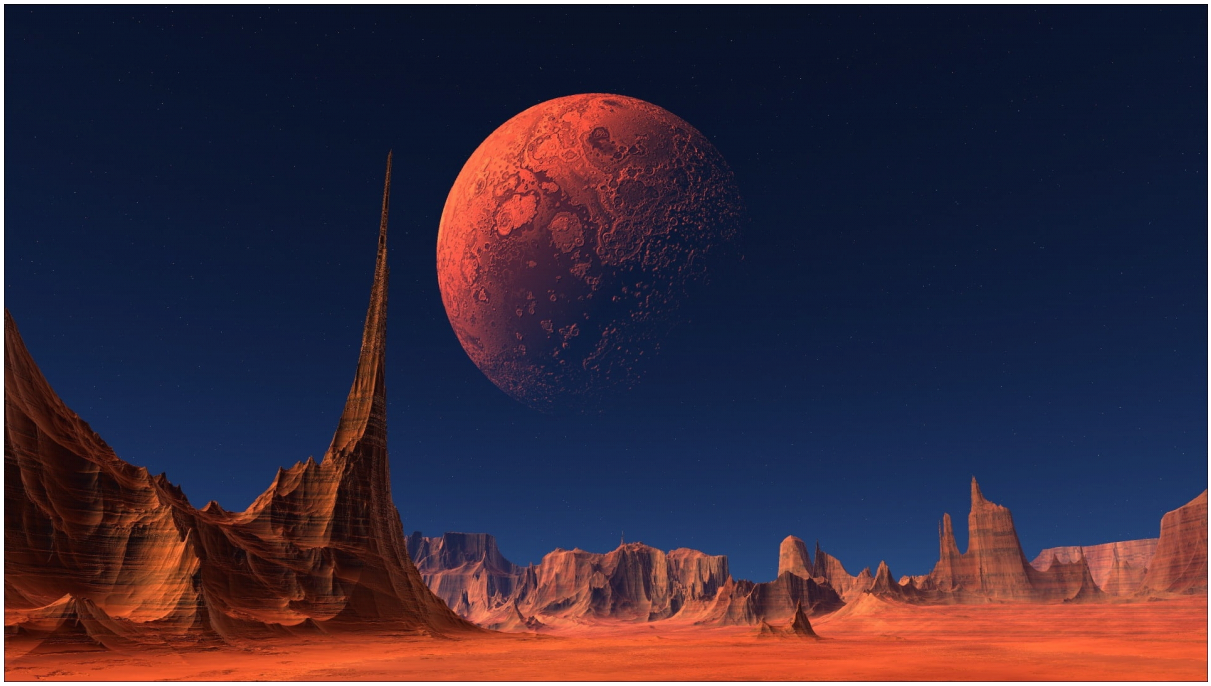

Fig. 4: Encrypted image which has sound embedded in it.

The Fig. 5 shows the decrypted audio from the image, showing that our audio decryption is completely lossless.

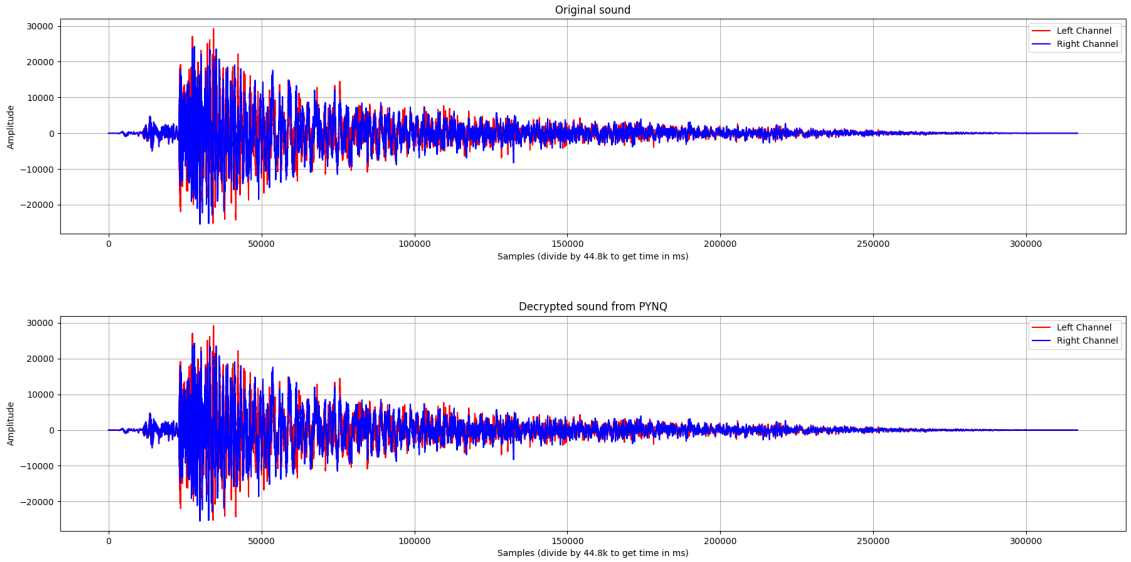

Fig. 5: Comparison of original sound and decrypted sound.

Tab. 3 and Tab. 4 show the post-implementation utilization of the encrypter and decrypter, respectively. The reason for the high usage of LUT in the decrypter is due to nested for loops in the Verilog logic. The decryptor searches through all patterns to identify the remainders and also checks them against the KEY. This requires nested

Tab. 3: Encryptor Resource utilization

Resource Type	Utilization	Available	Utilization (%)
LUT	5185	53200	9.75 %
LUTRAM	185	17400	1.06 %
FF	4266	106400	4.01 %
BRAM	3	140	2.14 %
BUFG	2	32	6.25 %

Tab. 4: Decryptor Resource utilization

Resource Type	Utilization	Available	Utilization (%)
LUT	34768	53200	65.35 %
LUTRAM	185	17400	1.06 %
FF	3909	106400	3.67 %
BRAM	3	140	2.14 %
BUFG	1	32	3.13 %

loop logic in Verilog, and during synthesis, the loop unrolling increases the overall LUT utilization.

5 Conclusion

This paper presented a novel steganographic system that combines SNN-inspired encoding with FPGA-based acceleration to hide audio data within images. By exploiting spike patterns generated from LIF neurons using the NEST simulator, each digit of an audio sample is mapped to a remainder–key pair that ensures secure and reversible encryption. Embedding the encrypted digits into the least significant bits of RGBA image channels provides a payload of 8 bits per pixel while preserving visual quality. The complete design was realized on a PYNQ-Z2 board, with custom encryptor and decryptor IPs integrated via AXI DMA for efficient dataflow. Experimental results confirm that the proposed method achieves real-time performance and guarantees lossless audio recovery. While the decryptor logic is resource-intensive due to nested search operations across spike patterns, it demonstrates correct and reliable reconstruction. Future work will address architectural optimizations to reduce resource utilization, extend support to higher resolutions and video streams, and explore adaptive encoding strategies for enhanced robustness and security.

References

[1] PYNQ — pynq.io. `https://www.pynq.io/`. [Accessed 30-08-2025].

[2] PYNQ Overlays — Python productivity for Zynq (Pynq) — pynq.readthedocs.io. `https://pynq.readthedocs.io/en/latest/pynq_overlays.html`. [Accessed 30-08-2025].

[3] Weiqi Luo, Kangkang Wei, Qiushi Li, Miaoxin Ye, Shunquan Tan, Weixuan Tang, and Jiwu Huang. A comprehensive survey of digital image steganography and steganalysis. *APSIPA Transactions on Signal and Information Processing*, 13(1):–, 2024.

[4] Ruben A. Tikidji-Hamburyan, Vignesh Narayana, Zeki Bozkus, and Tarek A. El-Ghazawi. Software for brain network simulations: A comparative study. *Frontiers in Neuroinformatics*, 11:46, 2017.

[5] Wikipedia contributors. Biological neuron model — Wikipedia, the free encyclopedia, 2025. [Online; accessed 25-August-2025].

[6] Wikipedia contributors. Double dabble — Wikipedia, the free encyclopedia, 2025. [Online; accessed 27-August-2025].

[7] Wikipedia contributors. Spiking neural network — Wikipedia, the free encyclopedia, 2025. [Online; accessed 25-August-2025].

[8] Kevin Alex Zhang, Alfredo Cuesta-Infante, Lei Xu, and Kalyan Veeramachaneni. Steganogan: High capacity image steganography with gans, 2019.