Frequency-Domain Logic Encoding for Protocol-Enforced Systems

Christopher Hirschauer
c.hirschauer@outlook.com

August 30, 2025

Abstract

This paper proposes a novel control architecture that replaces traditional binary logic with frequency-domain encoding to achieve enhanced semantic density, forensic traceability, and protocol-level enforcement. By mapping logical states to discrete frequency bands, the system enables multi-valued logic, intent encoding, and resilient plugin validation. The tradeoff between raw execution speed and semantic throughput is quantified, and implementation pathways are outlined for both software simulation and hardware integration.

Keywords: Protocol-Based Computing, Frequency Encoding, Semantic Logic, Control Architecture, Signal Enforcement

1 Introduction

Traditional computing systems rely on binary logic—voltage levels representing 0 and 1—to encode and execute instructions. While efficient, this model lacks semantic expressiveness and is vulnerable to default behaviors, abstraction leakage, and audit failure. This paper introduces a frequency-based logic model where each logical state is represented by a distinct frequency, enabling multi-bit encoding per signal pulse and enforcing control protocols at the signal layer.

2 Frequency-Based Logic Model

2.1 Encoding Scheme

- Binary Model: 2 frequencies $\rightarrow$ 1 bit per pulse
- Hex Model: 16 frequencies $\rightarrow$ 4 bits per pulse
- Extended Model: 2^n frequencies $\rightarrow$ n bits per pulse

Each frequency f_i maps to a semantic intent:

- $f_1 = 1$ kHz $\rightarrow$ Audit-only
- $f_{10} = 10$ kHz $\rightarrow$ Override Approved
- $f_{16} = 16$ kHz $\rightarrow$ Force Rollback

2.2 Control Enforcement

Execution is gated by frequency validation. Instructions are only executed if the correct frequency signature is present, ensuring explicit approval and forensic traceability.

3 Semantic Throughput vs Raw Speed

3.1 Semantic Gain

Hex-frequency encoding delivers 4x semantic payload per signal compared to binary. Reduces signal count per instruction by 50

3.2 Speed Tradeoff

Signal discrimination latency: 30–50% overhead. Synchronization and filtering introduce MHz-scale bottlenecks. Net result: slower execution, but higher control fidelity.

4 Implementation Pathways

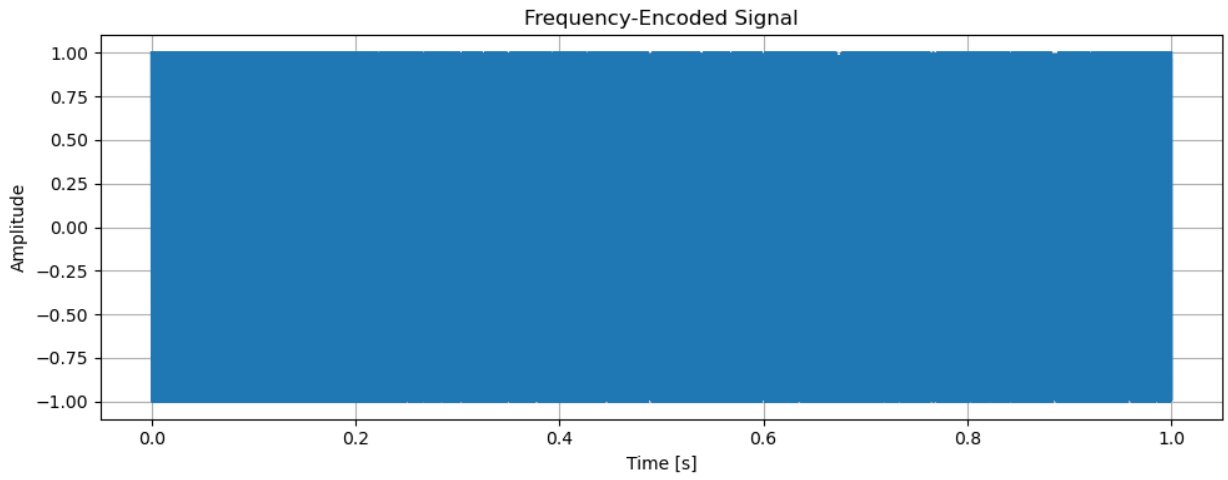
4.1 Software Simulation

```
import numpy as np
from scipy.signal import chirp
import matplotlib.pyplot as plt

fs = 100000  # Sampling frequency
T = 1        # Duration in seconds
f_start = 1000
f_end = 16000

t = np.linspace(0, T, int(fs*T))
signal = chirp(t, f0=f_start, f1=f_end, t1=T, method='linear')

plt.plot(t, signal)
plt.title('Frequency-Encoded Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.grid(True)
plt.tight_layout()
plt.savefig('frequency_encoded_signal.png')
```



4.2 Hardware Integration

- FPGA or DSP-based frequency discriminators
- Oscillator banks for signal generation
- Analog-to-digital converters for real-time validation

5 Error Modeling and Resilience

Frequency misclassification modeled via Gaussian noise and band overlap. Error correction via Hamming codes or CRC tagging. Redundant fallback frequencies for recovery modes.

6 Applications

- Plugin Recovery Protocols: Frequency-tagged rollback and override signals
- AI Instruction Enforcement: GPT directives encoded in waveform pulses
- Forensic Logging Systems: Signal-level audit trails with semantic tagging

7 Conclusion

Frequency-domain logic encoding offers a transformative approach to control architecture, enabling multi-valued logic, semantic compression, and protocol-level enforcement. While it introduces latency, the tradeoff is justified in systems demanding resilience, auditability, and override protection. This model redefines the relationship between signal and intent, paving the way for next-generation protocol-driven computing.