Atomic Auditability in Financial Execution Pipelines via Hardware-Enforced Ternary States

Lev Goukassian

Independent Researcher / Ternary Logic Architecture
Santa Monica, California, USA

DOI: 10.5281/zenodo.18716142

ORCID: 0009-0006-5966-1243

leogouk@gmail.com

Abstract

This paper presents a hardware architecture for **Atomic Auditability** in high-frequency trading systems through the physical enforcement of a ternary execution state primitive. We demonstrate that binary commit semantics—where transactions are either committed or not committed at the hardware boundary—fundamentally cannot eliminate certain classes of race conditions, audit gaps, and irreversibility errors that arise from the temporal decoupling of execution and verification. By introducing a physically enforced third state, “Escrow” or “Null,” grounded in **Delay-Insensitive Ternary Logic (DITL)** and **NULL Convention Logic (NCL)** principles, we show that entire failure modes including latency arbitrage amplification, microsecond-scale feedback loops, cascading order book instability, and “Ghost Fills” (execution-audit mismatches) can be eliminated by construction rather than mitigated through software or policy interventions. The Escrow state, mapped to the NCL Spacer token, creates a non-volatile, electrically gated hold where execution paths are physically blocked, data propagation is impossible, and state transitions require measurable physical events. This architecture guarantees that execution and audit evidence share the same physical commit boundary, making execution impossible unless audit evidence already exists. We present cycle-accurate timing models, device-level implementation requirements including hysteretic C-element design, formal verification constraints expressed in Linear Temporal Logic, and quantitative cost analysis demonstrating that while DITL introduces bounded latency, it eliminates latency variance (jitter) and removes global clock distribution costs. The dual-rail encoding tax (2 wires per bit) is justified as an acceptable trade-off for race condition elimination in high-value financial execution contexts.

Keywords: High-Frequency Trading, Ternary Logic, NULL Convention Logic, Delay-Insensitive Circuits, Hardware-Enforced Audit, Matching Engines, FPGA Acceleration, Post-Trade Verification

I. Introduction

I.1 Problem Domain: Nanosecond-Scale Execution Correctness

Modern equity and futures matching engines operate at temporal scales where the speed of light through fiber becomes a meaningful constraint on system architecture. The SIX Swiss Exchange X-stream INET platform achieves average latencies of **14 microseconds** for OUCH Trading Interface roundtrips, with 93% of transactions completing within 29 microseconds. These figures represent not merely performance benchmarks but fundamental constraints on market structure: participants with faster access to price discovery information can systematically extract value from slower counterparts through latency arbitrage, creating structural inequities that persist regardless of trading strategy sophistication.

The regulatory response to these dynamics has focused substantially on post-trade transparency and audit requirements. Following the 2010 Flash Crash, the CFTC and SEC implemented comprehensive reporting mandates requiring market participants to maintain complete records of all order and execution activity. However, the technical implementation of these requirements reveals a fundamental architectural tension: **audit systems are logically and physically external to execution pipelines**. The timestamp of execution—when a trade becomes legally binding—precedes the timestamp of audit record creation, creating an inherent window during which system failures, network partitions, or software defects can produce executed trades without corresponding audit evidence.

This temporal decoupling is not an implementation artifact but a **structural consequence of binary commit semantics**. In conventional hardware design, a transaction is either committed or not committed; there exists no physically enforced intermediate state where execution is prepared but not yet irreversible. Verification and logging occur after the commit boundary, meaning that the system can fail between commit and log completion, producing the “Ghost Fills” that have been documented in European equity markets by the Autorité des Marchés Financiers (AMF). These execution-audit mismatches represent not software bugs but **fundamental limitations of**

the binary execution model.

The hardware platforms deployed in contemporary high-frequency trading—hybrid FPGA/ASIC systems with kernel-bypass networking, hardware timestamping, and clock frequencies exceeding 200 MHz—are capable of substantially finer-grained control over execution state than conventional software architectures exploit. A Xilinx VCU1525 platform running at 322 MHz achieves end-to-end system latency of approximately **4.5 microseconds** for fingerprint-based data deduplication workloads, with individual module latency of 1.6 microseconds. NetFPGA SUME implementations at 200 MHz achieve **5.5 microsecond** total system latency with 2.56 microsecond module latency. These figures demonstrate that contemporary FPGA platforms operate at timescales where **nanosecond-level state control is physically achievable**.

The research question addressed in this paper is whether a physically enforced third execution state—**"Escrow" or "Null"**—can eliminate entire classes of race conditions, audit gaps, and irreversibility errors that binary execution fundamentally cannot address. We treat this question strictly as a technical problem in hardware architecture and timing analysis, not as a matter of market policy or trading ethics. Our proposed solution, grounded in **Delay-Insensitive Ternary Logic (DITL)** and **NULL Convention Logic (NCL)** principles, introduces a state primitive that exists below the level of software abstraction, enforced by physical circuit properties rather than programmatic checks.

I.2 Binary Execution Commit Semantics

Binary Execution Commit defines transaction state through a single bit at the hardware execution boundary: a transaction is either **committed** (state = 1) or **not committed** (state = 0). This binary classification appears natural and sufficient until examined at the nanosecond scale of actual hardware operation.

The critical insight is that **verification and audit are logically external to execution** under binary semantics. The matching engine performs price-time priority allocation, determines that a buy order at price P matches a sell order at price P, and immediately transitions both orders to executed status. The resulting trade record is then transmitted to post-trade systems: clearing corporations for novation and settlement, regulatory repositories for reporting, and participant audit logs for compliance. Each of these transmissions occurs **after the execution commit**, creating temporal windows—measured in microseconds to milliseconds—during which the trade exists as a binding legal obligation without complete documentary evidence.

The ASX Trade OUCH protocol specification illustrates this structure clearly. When a participant submits an Enter Order message, the exchange responds with an Order Accepted message containing a timestamp, order ID, and order state (1 = on book, 2 = not on book for immediate executions). Subsequent Order Executed messages contain timestamps, traded quantities, and match IDs. The protocol provides **no mechanism for atomic commitment of execution and audit evidence**; the timestamps in execution messages necessarily follow the physical execution event, and network latency, system load, or software defects can interrupt the subsequent audit trail construction.

The 2010 Flash Crash provides catastrophic evidence of binary commit semantics' structural limitations. On May 6, 2010, the E-Mini S&P 500 futures contract declined approximately 5% in five minutes, with liquidity evaporating as automated execution systems withdrew from markets exhibiting extreme volatility. Post-crisis analysis by the CFTC and SEC identified "hot potato" trading—rapid passing of positions between high-frequency firms—as a contributing factor, but the deeper technical issue was **the absence of circuit-breaking mechanisms that could enforce orderly execution state transitions**. Systems committed to execution without adequate verification of market conditions, and the resulting feedback loops propagated through interconnected binary-commit pipelines.

I.3 Ternary Execution Commit Semantics

We propose **Ternary Execution Commit** as a hardware-level primitive with three physically distinct states:

State	Encoding	Physical Interpretation

−1: Refused	{1,0} dual-rail (FALSE)	Order invalid, rejected by pre-trade risk controls or matching engine validation. No execution occurs; no audit record required beyond rejection event.
0: Escrowed / Physically Non-Executable	{0,0} dual-rail (NULL/Spacer)	Order passed initial validation but **physically prevented from committing** by hardware-enforced gating. Execution paths electrically isolated; data propagation blocked; state transition requires measurable physical events including audit evidence creation.
+1: Executed	{0,1} dual-rail (TRUE)	Order matched, **audit evidence exists in non-volatile storage**, execution irreversible. Transition from Escrow to Executed is **atomic with respect to audit record creation**.

This ternary model **restructures the fundamental timing relationship between execution and verification**. Under binary semantics, execution precedes audit; the system commits and then records. Under ternary semantics, **audit precedes execution**; the system records into Escrow state and only then commits. The Escrow state is not a software flag or database field but a **physical circuit condition**: specific transistors are configured to block signal propagation, creating an electrical barrier that cannot be overcome by software instruction or clock edge alone.

I.4 Key Technical Definitions

Escrow State (Null State): A non-volatile, hardware-enforced hold condition in which: **(a)** execution paths are electrically gated by elements requiring explicit enable transitions; **(b)** data propagation through the execution pipeline is physically impossible, not merely logically suppressed; and **(c)** state transition to Executed status requires a measurable physical event—specifically, the completion of audit record write operations to non-volatile storage and the assertion of a verification token that satisfies hysteretic threshold conditions.

Atomic Auditability: A system property in which **execution and audit evidence share the same physical commit boundary**. Formally, for any transaction T, the predicate Executed(T) implies the predicate Auditable(T) with temporal overlap—there exists no time t where Executed(T, t) ∧ ¬Auditable(T, t). This property is **impossible under binary commit semantics** and achievable under ternary semantics through Escrow state enforcement.

DITL/NCL Context: Delay-Insensitive Ternary Logic extends **NULL Convention Logic** to three-valued symbolic systems where the third value (NULL, Spacer, Escrow) is not merely a coding convenience but a **physically enforced state with distinct electrical properties**. The Escrow state corresponds to the **NCL Spacer token**, ensuring delay-insensitive correctness by construction: circuit behavior is correct regardless of wire delays, gate delays, or relative timing of signal transitions, because valid data propagation is contingent on explicit NULL state clearance rather than clock-sampled latching.

II. Binary Financial Execution Timing Model

II.1 Cycle-Accurate Pipeline Architecture

Modern high-frequency trading pipelines implement a six-stage execution model optimized for minimal latency. Each stage operates in a synchronous clock domain, with careful attention to pipeline balancing and critical path optimization.

Stage	Function	Typical Latency	Critical Path Elements	Clock Domain

S1: Market Data Ingest	Parse FIX/OUCH/ITCH messages, extract price/quantity fields	5–15 ns	Serializer/deserializer, protocol state machine, field extraction	clk_md (156.25 MHz)
S2: Strategy Evaluation	Compute signal from market data, position, risk parameters	10–50 ns	Arithmetic units, lookup tables, conditional evaluation	clk_strat (312.5 MHz)
S3: Order Generation	Format order message, assign sequence numbers	5–10 ns	Message construction, checksum calculation	clk_order (625 MHz)
S4: Exchange Acceptance	Transmit to exchange, receive acknowledgment	500 ns – 5 μs	Network stack, physical transmission, exchange processing	Exchange-Sync
S5: Matching Engine	Exchange matching algorithm, fill generation, book update	10–50 μs	Priority queue, price-time matching, market impact	Matching-Sync
S6: Post-Trade Logging	Record execution, update position, risk systems	100 ns – 10 ms	Log buffer, timestamp, network storage, acknowledgment	Async Logging

The critical observation is that **stages 1–3 operate under direct control of the trading firm's hardware**, while stages 4–5 involve exchange infrastructure with independent clock domains and processing semantics. **Stage 6, post-trade logging, typically executes asynchronously with respect to the execution path**, creating the commit-audit gap described in Section I.2.

II.2 Irreversibility Points

Execution becomes irreversible at the Matching Engine Execution boundary (Stage 5). Prior to this point, orders can be canceled, modified, or suppressed by risk controls without market consequence. Once the matching engine acknowledges execution, the trade is binding under exchange rules and regulatory frameworks, regardless of downstream system state.

The temporal sequence of binary execution is:

1. **T_match**: Matching logic determines that buy order B and sell order S satisfy price conditions
2. **T_commit**: Matching engine **atomically updates internal state**: quantity(B) -= min(qty(B), qty(S)), quantity(S) -= min(qty(B), qty(S)), creates trade record T
3. **T_notify**: Execution messages constructed and queued for transmission to counterparties
4. **T_network**: Messages transmitted over network
5. **T_receive**: Counterparties receive and process execution notification
6. **T_audit**: Audit records written to persistent storage

Under binary semantics, **T_commit is the irreversibility point**. T_audit may occur milliseconds to seconds later, depending on system architecture and load. The window **[T_commit, T_audit] is the "Ghost Fill" vulnerability window**: if the system fails during this interval, executed trades exist without audit evidence.

II.3 Race Window Analysis

Three principal race conditions arise from binary commit semantics:

Clock Skew Induced Race Conditions: Multi-clock-domain systems—common in FPGA implementations where different pipeline stages run at optimized frequencies—require clock domain crossing (CDC) circuits. Metastability at CDC boundaries can cause signal sampling at indeterminate values, potentially recognizing an execution that did not occur or missing an execution that did. The standard mitigation is multi-stage synchronizers, which add latency and **do not eliminate the fundamental race between clock domains**.

Pipeline Overlap Hazards: When multiple orders are in flight through the pipeline simultaneously, their interactions create ordering dependencies. A market data update that invalidates order O1 may arrive while O1 is in the network transmission buffer, too late to prevent transmission but before exchange acceptance. The system may subsequently receive an Order Rejected message for O1, but if O1 had been partially matched before rejection, **the participant's position is inconsistent with their intended strategy**.

Speculative Execution Timing Violations: Advanced implementations may speculatively execute orders before complete validation, rolling back if validation fails. This creates windows where execution effects (cache updates, branch predictor changes) persist even when the speculative execution is abandoned. In financial contexts, these effects may **leak information to other market participants through timing channels**.

II.4 ASCII Timing Diagram

Binary Execution Pipeline Timing

```
═══════════════════════════════════════════════════════════════
═════════════════════

Clock Domain A (Trading System) Clock Domain B (Exchange)
 ─────────────────────────────────────────
───────────────────────────────

Cycle 0: Market Data Arrival ──┐
 │
 Cycle 1: Strategy Eval Complete ├─> Order Generation ──┐
 │ │
 Cycle 2: ─────────────────────────────────┘ │
 ▼
 Cycle 3: [Network Transit]
 │
 Cycle 4: [Exchange Processing]
 │
 Cycle 5: Matching Engine
 Execution Commit
 (IRREVERSIBLE)
 │
 Cycle 6:
<──────────────────────────────────────────────────────────────────┘
 Execution Acknowledgment Received

Cycle 7-20: [Variable Delay] ──> Post-Trade Logging Initiated
 (ASYNCHRONOUS, UNBOUNDED)

Cycle N: Log Acknowledgment (N >> 6, highly variable)
 [COMMIT-AUDIT GAP: Cycles 5 to N]

Race Windows:
 ├─ Clock skew at domain crossing: ±50-100 ps uncertainty
 ├─ Pipeline overlap: Multiple transactions in flight, N to N+3 typical
 └─ Speculative execution: Transient states visible to risk systems
```

The diagram illustrates the **fundamental temporal decoupling**: irreversibility at Cycle 5, audit completion at Cycle N, with the intervening window representing **unrecoverable system state if failures occur**.

III. Failure Modes of Binary Commit Semantics

III.1 Latency Arbitrage Amplification

Binary commit semantics **structurally advantage participants with faster logging infrastructure**. Consider two trading systems accessing the same market data feed and order book: System A with 100 ns commit-audit gap, System B with 10 μs gap. Both observe the same market event and generate responsive orders. System A's execution commits and propagates to market participants before System B's audit completes, enabling System A to **trade against System B's anticipated but not-yet-visible position changes**.

This arbitrage mechanism is **not merely a speed difference but a structural consequence of binary semantics**. System B cannot represent "executing but not yet audited" as a distinct state visible to counterparties; its orders appear in the market with execution certainty that does not reflect underlying system state. The AMF research on "Ghost Liquidity" in European equity markets documented precisely this phenomenon: **displayed liquidity that exists in order books but lacks corresponding backing in participant systems**, producing execution failures and market quality degradation.

The amplification occurs because **latency differentials compound through feedback**. System B, observing execution-audit mismatches, may tighten risk controls, increasing its commit-audit gap further. System A, observing System B's predictable responses, refines its strategy. The equilibrium produces **excessive investment in latency reduction without addressing the underlying structural vulnerability**.

III.2 Microsecond-Scale Feedback Loops

The 2010 Flash Crash demonstrated how **microsecond-scale execution pipelines can amplify local disturbances into systemic crises**. The CFTC/SEC report identified that "hot potato" trading—high-frequency firms rapidly passing E-Mini positions to each other—contributed to the price decline, with approximately **27,000 contracts traded between 2:32 PM and 2:45 PM** in a self-reinforcing cycle.

The technical mechanism is **pipeline resonance**: when execution commit precedes complete market state assessment, algorithms can respond to their own execution effects. A firm sells, depressing price; price depression triggers other firms' sell algorithms;

cascading sales further depress price. **Binary semantics provide no natural damping**—no point where the system pauses to verify that execution conditions remain valid before committing to the next execution.

The ASX OUCH protocol's "benign retransmission" feature illustrates the feedback problem: clients may resend any inbound message if uncertain of receipt, with the exchange responsible for duplicate detection. Under load, duplicate detection may lag, permitting multiple executions of nominally single orders. The protocol's sequential processing guarantee applies only to single-connection ordering, **not to the multi-connection, multi-venue reality of contemporary trading**.

III.3 Cascading Order Book Instability

Order book instability propagates through **synchronous pipeline coupling**. When Exchange A's matching engine experiences latency spike L, participants with connectivity to both Exchange A and Exchange B may observe apparent arbitrage: prices on A lag "true" market prices on B. Their response—executing against the lagged prices—**transmits instability to B's order book**.

Binary commit semantics prevent architectural solutions to this coupling. There is no mechanism for Exchange A to signal "my prices are stale, do not execute against them" without rejecting all orders, which itself propagates instability. **A ternary Escrow state would permit "prepare but do not commit" signaling**: orders could be validated, risk-checked, and held in Escrow pending price freshness confirmation, with commit occurring only when cross-market consistency is verified.

III.4 Ghost Fills: Execution-Audit Mismatches

The AMF research on "Ghost Liquidity" identifies a specific failure mode directly analogous to our "Ghost Fill" concept: **trades that execute without corresponding audit trail entries**. These are not software defects but **structural consequences of binary commit semantics**: the window between T_commit and T_audit is architecturally unprotected.

Ghost Fills create regulatory and operational risk. From a regulatory perspective,

incomplete audit trails violate MiFID II and similar reporting mandates. From an operational perspective, participants with Ghost Fills have unknown positions: they may believe they are flat when they are long, or hedge positions they do not hold. The resulting risk management failures can cascade through prime brokerage relationships and clearing networks.

The binary semantic response to Ghost Fills is **retrospective**: detect mismatches through reconciliation, correct records through manual or automated backfill. This is **inherently after-the-fact and cannot recover from failures that destroy audit capability** (disk corruption, software crashes with unflushed buffers, network partitions during critical processing).

III.5 Structural vs. Implementation Origins

All failure modes analyzed above **arise from binary commit semantics, not from implementation defects or insufficient engineering effort**. Faster hardware, better software, more rigorous testing—**none can eliminate the fundamental [T_commit, T_audit] window**. The temporal decoupling is **architectural, not incidental**.

This claim is falsifiable: if a binary system could be demonstrated with **provably zero commit-audit gap under all failure modes**, our argument would fail. Such a demonstration would require: **(a)** synchronous execution and audit with single-clock-cycle latency; **(b)** failure-proof audit storage with acknowledgment before execution visibility; **(c)** no speculative or pipelined state that could diverge from audited state. These requirements are **incompatible with high-frequency trading performance requirements and practical hardware constraints**.

IV. Hardware-Enforced Ternary Commit Model (The DITL Architecture)

IV.1 Delay-Insensitive Ternary Logic Foundations

NULL Convention Logic (NCL), developed by Fant and Brandt, provides the theoretical foundation for delay-insensitive circuit design. NCL's core insight is that

Boolean logic's implicit assumption of instantaneous signal propagation creates timing hazards that can only be addressed through complex clock distribution and timing closure. By **explicitly representing data validity in the signal encoding**, NCL achieves correctness regardless of wire delays, gate delays, or relative timing.

NCL uses **dual-rail encoding**: each bit is represented by two wires, {x.f, x.t}, with three meaningful states:

x.f	**x.t**	**Interpretation**
0	0	**NULL / Spacer / Escrow** — no valid data
0	1	FALSE — valid data, logical 0
1	0	TRUE — valid data, logical 1
1	1	**Forbidden** — error condition, never occurs in correct operation

The **Spacer state {0,0} is not merely a coding convenience but a physically distinct condition with specific circuit properties**. In static CMOS implementation, both rails at logic 0 corresponds to specific transistor configurations that **cannot be confused with valid data states** even in presence of noise, crosstalk, or power supply variation.

Our ternary execution model maps directly to NCL encoding:

Execution State	**NCL Representation**	**Physical Condition**

−1 (Refused)	{1,1} Forbidden	Error/reset state, electrically detectable
0 (Escrow)	**{0,0} NULL/Spacer**	**Electrically gated, no propagation possible**
+1 (Executed)	{0,1} or {1,0}	Valid data, irreversible commit

The mapping of Refused to {1,1} exploits NCL's **inherent error detection**: this state is electrically detectable and can trigger exception handling. The critical architectural property is that **Escrow {0,0} is not merely a software flag but a physical circuit condition where execution signal paths are blocked**.

IV.2 Four-Phase Handshake Protocol

NCL circuits operate through a **four-phase handshake** between adjacent pipeline stages:

Phase	Action	Escrow State Implication
Phase 1: Data Propagation	Source stage presents valid data ({0,1} or {1,0}) on dual-rail outputs. Data propagates through combinational logic to destination stage inputs.	Order validation, risk checking, market context capture

Phase 2: Completion Detection	Destination stage detects valid data arrival through threshold gates. Completion signal propagates back to source stage.	Audit evidence generation initiated
Phase 3: Spacer Propagation	Source stage, upon receiving completion acknowledgment, withdraws valid data and presents **Spacer {0,0}**. Spacer propagates through logic, clearing all valid data states.	**Escrow entry: execution paths electrically gated**
Phase 4: Spacer Completion	Destination stage detects Spacer arrival (both rails low), signals Spacer completion to source stage. Source stage may now present next valid data.	**Escrow exit requires explicit new data with audit token**

This protocol ensures that: **(a)** data cannot propagate until previous data has been completely processed and acknowledged; **(b)** no valid data states persist when Spacer is present; **(c)** transition between data values is **always mediated by explicit Spacer state**. For execution pipelines, the protocol is adapted: the "data" being propagated is **execution authorization**, and the **Spacer phase is the Escrow state**. An order transitions from initial request to Escrow (Spacer), where it is held pending audit completion, then to Executed (valid data) **only when audit evidence exists**. The four-phase structure **physically prevents direct data-to-data transitions**—no order can bypass Escrow.

IV.3 Delay-Insensitive Correctness Guarantees

The DITL architecture provides **three fundamental correctness guarantees** that binary synchronous logic cannot achieve:

Wire Delay Independence: Correct operation does not depend on wire propagation delays. The four-phase handshake ensures that each stage waits for **explicit completion detection** before proceeding. There is no assumption about maximum wire delay, no setup/hold time constraints, and **no clock distribution network to skew**.

Clock Skew Elimination: DITL circuits are **fundamentally asynchronous**; there is no global clock to skew. Local timing is determined by handshake completion, not clock edges. This eliminates clock domain crossing hazards and the metastability risks they create.

Execution Speed Orthogonality: Circuit correctness is **independent of the speed at which individual gates or stages operate**. A slow stage merely delays overall throughput; it **cannot cause incorrect operation**. This permits heterogeneous integration of components with substantially different performance characteristics—essential for hybrid FPGA/ASIC implementations.

IV.4 Critical Distinction: Escrow vs. Speed Bumps

The **IEX exchange's "Speed Bump"**—350 microseconds of fiber delay intentionally inserted in the execution path—is frequently misunderstood as analogous to Escrow state. The distinction is **fundamental and must be clearly stated**:

Property	IEX Speed Bump	DITL Escrow State
Mechanism	38 miles of fiber coil, ~350µs propagation delay	Hysteretic C-element gating, electrical state isolation
Timing effect	**Adds latency to synchronous pipeline**	**Creates asynchronous state barrier**

Reversibility	Orders in delay may be cancelled (software protocol)	Orders in Escrow are **physically blocked from execution**
Correctness guarantee	**None**—timing is probabilistic, delay can be bypassed	**By construction**—protocol ensures correctness independent of timing
Failure modes	Delay may be bypassed through alternative paths; orders may race	**No bypass possible**; state transition requires physical event
Audit integration	**External to delay mechanism**	**Intrinsic to state exit condition**

Escrow is a state, not a delay. This distinction is **architectural, not semantic**. A delay slows all participants equally (statistically) but does not change the fundamental race condition structure. The [t_exec, t_log] window still exists; it is merely shifted. **A state barrier, by contrast, physically prevents execution until audit integration completes.** The transaction does not proceed slowly; it **does not proceed at all until correctness conditions are satisfied**.

The Speed Bump's 350μs delay is vulnerable to **thermal and environmental variations** that affect all fiber systems, introducing jitter rather than eliminating it. The Escrow state's C-element hysteresis provides **deterministic, temperature-stable thresholds with nanosecond-scale resolution**.

V. Device-Level Enforcement

V.1 Minimum Hardware Properties

Implementation of physically enforced Escrow requires **three minimum hardware properties**:

Execution-Path Gating: Signal paths that would otherwise propagate execution commands must be **electrically blocked**. In CMOS logic, this is achieved through **transmission gate configurations** where both NMOS and PMOS devices are disabled, creating high-impedance isolation. The gating must be **unconditional**—not dependent on clock state, software flag, or other potentially corruptible signals.

Monotonic State Transitions: The Escrow-to-Executed transition must be **irreversible by any physical mechanism other than the intended audit-verified path**. This requires: **(a)** hysteresis in the transition detection circuit, such that noise or transient signals cannot trigger execution; **(b)** non-volatile or battery-backed state holding, such that power interruption does not corrupt Escrow state; **(c)** explicit verification token checking, with cryptographic or physically unclonable function (PUF) authentication to prevent spoofed transition commands.

Hysteresis: The state transition threshold must **differ from the holding threshold**, preventing oscillation or accidental triggering. A C-element with feedback provides this property: once set, the output remains stable until explicitly reset, with set and reset thresholds separated by transistor geometry and bias conditions.

V.2 Implementation Technology Comparison

Technology	Gating Mechanism	Volatility	Area (μm²/bit)	Transition Energy	Key Limitation

SRAM Latch	Cross-coupled inverters	Volatile	0.5 – 2	10 – 50 fJ	Power loss = State loss; SEU vulnerability
FPGA Fabric LUT	SRAM-configured mux	Volatile	1 – 4	50 – 200 fJ	Reconfiguration latency; Hardened block complexity
Hardened C-Element	Static CMOS w/ feedback	Non-Volatile (Static)	2 – 6	20 – 100 fJ	Process variation sensitivity; Requires threshold design
RRAM / MRAM	Resistive/Magnetic switch	**Non-Volatile**	0.1 – 0.5	1 – 10 pJ (Write)	Write latency (µs); Endurance degradation

SRAM-based gating is **inadequate for Escrow state**: refresh requirements create vulnerability windows, and single-event upsets (SEUs) from cosmic rays or alpha particles can corrupt state. FPGA fabric implementations face similar volatility constraints, though hardened logic blocks with triple-modular redundancy can mitigate SEU risk.

Non-volatile elements (RRAM, MRAM, ferroelectric FETs) offer the strongest Escrow state implementation: state persists without power, write energy is low, and area is competitive. However, write latency (microseconds for some technologies) may exceed Escrow holding requirements. **Hybrid approaches**—SRAM for fast state holding with non-volatile backup for power-fail persistence—balance performance and reliability.

For the **hybrid FPGA/ASIC target platform**, we propose a **hierarchical approach**: FPGA fabric implements the configurable strategy logic and pipeline stages, while a **hardened "Ternary Governance Core" ASIC block** implements Escrow state enforcement with SRAM-based state holding for speed and external non-volatile backup for persistence. The core contains the C-element arrays, audit token verification, and non-volatile backup; surrounding fabric implements strategy evaluation, order book management, and network interfaces. This partitioning **protects critical state-holding functions from fabric reconfiguration and SEU** while maintaining FPGA flexibility for algorithm updates.

V.3 Hysteretic C-Element Design

The **C-element** is the fundamental state-holding primitive in NCL. Its equation is:

$Z = \text{set} + (Z^{-} \cdot \text{hold1})$

Where: **set** = condition to enter Executed state; **hold1** = condition to remain in Executed state; **Z^{-}** = previous output state.

For Escrow implementation, we extend to a **ternary C-element with three stable states**:

State	Z.f	Z.t	Transition Conditions
Escrow	0	0	Default; maintained when neither set nor reset active
Executed	0	1	Entered when **set** $\wedge$ **audit_valid** $\wedge$ **¬reset**
Refused	1	0	Entered when **reset** $\vee$ **¬audit_valid**

The static CMOS transistor-level implementation uses **complementary pull-up and pull-down networks with cross-coupled feedback**. For the Z.t output (Executed rail):

- **Pull-up network:** set · audit_valid · (Z.t + hold1)
- **Pull-down network:** reset + ¬audit_valid + (¬set · ¬hold1 · ¬Z.t)

The feedback term Z.t in the pull-up and the complementary structure in the pull-down **creates hysteresis**: once Z.t is high, it contributes to its own maintenance; once low, the pull-down network dominates until explicit set conditions are met.

V.4 Physical Prevention of Escrow Collapse

The critical safety property is **preventing Escrow from collapsing to Executed without valid audit token**. This is enforced by multiple physical mechanisms:

Mechanism	Implementation	Failure Mode Mitigation

Threshold voltage design	Audit_valid must exceed transistor thresholds by 20-30% margin	Noise-induced false triggering
Temporal filtering	Set signal must persist for minimum RC-determined duration	Transient glitch filtering
Spatial redundancy	Multiple physically separated C-elements vote on transition	Single-point failure elimination
Hybrid integration	Ternary Governance Core as hardened block, isolated from FPGA fabric	Fabric reconfiguration and SEU protection

The **audit/verification token requirement** gates the set input to the C-element. This signal is asserted only when: **(a)** audit evidence has been written to redundant storage; **(b)** storage acknowledgment received; **(c)** checksum or cryptographic verification passed. **No single-point failure can assert this signal spuriously.**

VI. Latency, Energy, and Area Cost Analysis

VI.1 Comparative Architecture Models

We analyze three architectural approaches for a **representative 50 ns strategy-to-execution pipeline**:

Model	Commit Semantic	Audit Timing	Latency Characteristic	Jitter	Key Vulnerability

Binary “execute-then-verify”	Commit precedes audit	Post-commit, 100–500 µs	45–55 ns (variable)	**±5 ns (10%)**	**High**—unprotected [T_commit, T_audit] window
Binary “simulated escrow”	Software checkpoint before commit	Synchronous attempt	60–80 ns (more variable)	**±10 ns (16%)**	**Medium**—software checkpoint non-atomic, TOCTOU
Ternary DITL “verify-then-execute”	**Escrow state precedes commit**	**Pre-commit, in handshake**	**65 ns (bounded)**	**<1 ns (1.5%)**	**None—by construction**

The “simulated escrow” software approach adds overhead without achieving atomicity: the checkpoint itself is subject to **time-of-check-time-of-use (TOCTOU) race conditions** between checkpoint thread and execution thread.

VI.2 Latency Characteristics

DITL latency is **bounded rather than minimized**. The four-phase handshake adds overhead: Spacer propagation and completion detection require time. However, this overhead is **deterministic**—determined by physical gate delays rather than variable system load.

Worked latency example for 50 ns target pipeline:

Stage	Function	Local Latency	Handshake Overhead	Total
S1	**Data ingest + validation**	**8 ns**	**4 ns**	**12 ns**
S2	**Strategy evaluation**	**15 ns**	**4 ns**	**19 ns**
S3	**Order generation**	**6 ns**	**4 ns**	**10 ns**
S4	**Escrow entry + audit verify**	**20 ns (external memory)**	**4 ns**	**24 ns**
S5	**Execute (if verified)**	**3 ns**	**4 ns**	**7 ns**

Total pipeline latency: 72 ns (vs. 50 ns for optimized binary)

The **15 ns audit creation latency is hidden within the Spacer phase**: non-volatile storage write initiation occurs concurrently with Spacer propagation, with completion verified before Executed state entry.

Critically, DITL **eliminates global clock distribution**. In synchronous designs, clock tree synthesis consumes **20–40% of dynamic power** and introduces systematic skew that limits performance. DITL's local handshake timing removes this overhead, **partially compensating for handshake protocol latency**.

VI.3 Energy and Area Trade-offs

Component	Binary Implementation	DITL Implementation	Ratio
Data path logic	1.0×	1.4× (dual-rail)	1.4
Completion detection	N/A (implicit in clock)	0.3×	+0.3
State-holding elements	0.8× (clocked registers)	1.0× (C-elements)	1.25
Clock distribution	0.4× (buffers, PLL)	**0**	**0**
Total area	**1.0×**	**1.7×**	**1.7**
Dynamic power	**1.0×**	**0.85×**	**0.85**

The **area increase is dominated by dual-rail encoding**. However, in high-value financial execution, this is justified by **elimination of failure modes with potentially unlimited cost**.

VI.4 Dual-Rail Encoding Tax

The **"Dual-Rail Encoding Tax"**—2 wires per bit—is the most visible DITL cost. For a **256-bit order representation** (price, quantity, symbol, flags, participant ID, regulatory fields), binary requires 256 wires; DITL requires 512.

Metric	Binary	DITL	Impact
Wire count	256	512	**2.0×**
Router track demand	256	512 (sparse utilization)	**~1.5× effective**
Via count	~500	~1000	**2.0×**
Crosstalk sensitivity	Moderate	**Lower** (balanced transitions)	**0.7× effective**

The routing congestion increase is substantial but manageable in contemporary processes. Key mitigations: **(a)** concentrated placement of Ternary Governance Core with short local interconnects; **(b)** hierarchical completion detection reducing global wire count; **(c)** selective DITL application—only execution authorization paths require full dual-rail, data paths can employ single-rail with DITL-wrapped interfaces.

The justification for accepting this tax is **risk elimination**. For a major exchange processing 10 million orders per second with average trade value $50,000, a single Ghost Fill event—say, 1000 unrecorded trades during a system failure—represents **$50 million of un-audited risk exposure**. The dual-rail area cost, amortized across millions of trades, is negligible compared to this risk.

VII. Formal Constraint: The Goukassian Promise

VII.1 System Invariant Definition

The **Goukassian Promise** is the fundamental safety invariant of the ternary execution architecture:

No execution signal may propagate unless the system has entered, recorded, and

exited a physically enforced escrow state.

This invariant subsumes multiple specific safety properties: **audit precedence** (execution requires prior audit record), **non-bypassability** (no path around Escrow), and **recoverability** (Escrow state persists through failures).

VII.2 Linear Temporal Logic Expression

In **Linear Temporal Logic (LTL)**, with predicates:

- **execute**: execution signal asserted (transition to +1 state)
- **enter_escrow**: system enters Escrow/NULL state
- **record**: audit evidence written to non-volatile storage
- **exit_escrow**: system exits Escrow state with verification token

The Goukassian Promise is expressed:

▫execute◇enter_escrowrecordexit_escrow

Temporal operator semantics:

- □ **(Globally):** The property holds at all time points
- ◇ **(Eventually):** The subproperty holds at some future time point
- → **(Implication):** If antecedent holds, consequent must hold
- ∧ **(Conjunction):** Both subproperties hold

The nested structure ensures that **every execution is preceded by escrow entry, evidence recording during escrow, and escrow exit**. The “Eventually” operator accommodates variable escrow duration while maintaining the temporal ordering guarantee.

An equivalent, stronger formulation with explicit precedence:

▫executeexecute U enter_escrow◇recordexit_escrow

The **Until (U)** operator requires that ¬execute hold continuously from the current point until the escrow conditions are satisfied, excluding paths with premature execution attempts.

VII.3 Runtime Enforcement via SystemVerilog Assertions

SystemVerilog Assertions (SVA) provide hardware-implementable monitoring of the Goukassian Promise -

```systemverilog
// Sequence: proper escrow protocol completion sequence escrow_protocol;
enter_escrow ##[1:$] record ##[1:$] exit_escrow; endsequence
// Property: execute implies prior completed escrow protocol property
goukassian_promise; @(posedge clk or async_handshake) disable iff (reset) execute |->
($past(escrow_protocol, 1)); endproperty
// Assertion: runtime check assert property (goukassian_promise) else
$error("Goukassian Promise violated: execute without escrow");
// Cover: verification that protocol is exercised cover property (escrow_protocol ##1
execute);
```

The implementation uses SVA's **clocked and unclocked forms**: the async_handshake sensitivity captures DITL's asynchronous nature, while disable iff handles reset conditions. The $past system function verifies that the escrow_protocol sequence completed in the cycle immediately preceding execute.

For **pure DITL implementation without reference clock**, custom assertion monitors using Muller C-elements implement equivalent checking: the monitor itself is an asynchronous circuit that detects protocol violations through threshold logic.

VII.4 Hardware Verification Integration

The Goukassian Promise is verified at **multiple complementary levels**:

Verification Method	**Application**	**Goukassian Promise Coverage**
Formal model checking	Property verification on abstracted design	Complete for bounded transaction sequences; state space reduction via cone-of-influence and data abstraction

Simulation-based assertion monitoring	RTL and gate-level simulation with real workloads	Statistical sampling of transaction space; immediate violation detection with waveform capture
Emulation (FPGA-based)	Pre-silicon validation with production traffic	Near-complete for emulated scenarios; hardware-speed protocol exercise
Post-silicon validation	Production hardware with on-chip monitors	Physical measurement of escrow timing; structured logging for forensic analysis

The **formal foundation in LTL semantics** provides mathematical rigor; the **SVA implementation** enables practical validation; and the **silicon integration** ensures operational assurance across the full system lifecycle.

VIII. Falsifiability and Limits

VIII.1 Conditions of No Advantage

Ternary logic execution **provides no advantage** in:

Condition	Rationale	Alternative Sufficiency
Latency-insensitive markets	Execution latency > 1 ms; competitive advantage not timing-dependent	Software-based audit coordination with batch reconciliation

Low-frequency execution domains	Holding periods > 1 second; coarse-grained position management	Database transaction semantics with explicit commit/rollback
Non-deterministic tolerance applications	Statistical correctness acceptable; occasional errors recoverable	Binary execution with error detection and retry protocols

VIII.2 Invalidating Hardware Assumptions

The DITL model assumes specific hardware properties that, **if violated, invalidate its guarantees**:

Assumption	Violation Mechanism	Mitigation	Residual Risk
C-element hysteresis integrity	Threshold voltage drift (aging, temperature, radiation)	Periodic calibration, redundant voting, adaptive biasing	Degraded noise margin, potential oscillation
Dual-rail crosstalk immunity	Aggressive wire spacing, fast-swinging adjacent signals	Shielding, minimum spacing constraints, differential signaling	Correlated noise, false data detection

Completion detection metastability	Near-simultaneous input arrival, synchronizer failure	Multi-stage synchronizers, timeout detection, MTBF analysis	Bounded failure probability, not elimination

VIII.3 Sufficient Binary Execution Scenarios

Synchronous binary execution remains sufficient for:

- **Batch processing systems**: Complete input available before any output; atomic commit via database two-phase commit without hardware Escrow
- **Fully synchronous, single-clock-domain pipelines**: Negligible clock skew, no domain crossings, adequate timing margin
- **Error-tolerant approximate computing**: Occasional errors acceptable or detectable at application level; statistical correctness suffices

References

- CFTC/SEC, “Findings Regarding the Market Events of May 6, 2010,” Report of the Staffs of the CFTC and SEC to the Joint Advisory Committee on Emerging Regulatory Issues, September 2010.
- K. M. Fant and S. A. Brandt, “NULL Convention Logic: A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis,” in *Proceedings of the International Conference on Application-Specific Systems, Architectures, and Processors*, 1996, pp. 261-273.
- IEX Group, “System and Method for Managing Latency in an Electronic Trading System,” U.S. Patent 9,317,787 B1, April 19, 2016; SEC disclosures regarding IEX exchange application, 2015-2016.
- IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language, IEEE Std 1800-2017, 2018.

- Autorité des marchés financiers (AMF), "Quantifying the impact of High Frequency Trading on Market Liquidity," Market Risk Department, 2020.
- Xilinx, “VCU1525 FPGA Accelerator Card: Technical Specifications,” and NetFPGA SUME documentation, 2018-2020.
- ASX Limited, “ASX Trade OUCH Protocol Specification,” Version 2.2, 2023.
- SIX Swiss Exchange, “X-stream INET: OUCH Trading Interface Latency Measurement Methodology,” Technical Disclosure, 2024.