B-TRON Virtualization and Formal Foundations: A Cleanroom Architecture for Real-Time Operating Systems from Embedded MCUs to Modern Hypervisors

Maxim Sokhatsky (Namdak Tonpa)

Synrc Research Center / Groupoid Infinity Cleanroom Working Group

Kyiv, Ukraine

maxim@synrc.com

August 2026

Abstract

Real-Time Operating Systems (RTOS) represent the computational bedrock of autonomous cyber-physical systems, aerospace avionics, and critical edge infrastructure. However, modern deployment environments demand both rigorous mathematical safety guarantees and seamless integration with contemporary hypervisor ecosystems. This paper presents a comprehensive cleanroom implementation and virtualization architecture for the **BTRON3 SPEC 3.20** specification, situated within a modern Type-Theoretic formal verification framework and a unified VirtIO 1.4 hypervision pipeline. We analyze the tri-tier operating system taxonomy (Nano, MCU, and Datacenter Compute) and investigate the triadic landscape of modern lightweight RTOS: Apache NuttX, Little Kernel (LK), and TRON (μITRON/BTRON). We present the design of a modular, multi-target execution matrix spanning four distinct backends: POSIX user-space emulation (*Takada*), QEMU VirtIO virtualization (*Izumina*), QEMU T-Kernel 2.0 compliance (*Sakamura*), and direct baremetal ARM execution on Raspberry Pi 2/3 (*Yokobayashi*). Furthermore, we formalize the BTRON Real Object / Virtual Object (実身/仮身) hyper-data model, the Display Primitives (DP) graphics engine, and demonstrate how Martin-Löf Type Theory (MLTT) and constructive proof assistants provide end-to-end verification isomorphism for safety-critical kernel services.

Keywords—BTRON, Real-Time Operating Systems, VirtIO, Formal Verification, Martin-Löf Type Theory, Cleanroom Implementation, Microkernels, T-Kernel, seL4.

1 Introduction and Historical Context

The architecture of modern systems software is shaped by decades of evolutionary trade-offs between monolithic general-purpose operating systems (GPOS) and deterministic real-time kernels. From early developments in low-level systems programming in the late 1980s through the emergence of open-source Unix distributions in the late 1990s, kernel engineering has continually grappled with balancing performance, fault isolation, and developer ergonomic models.

Concurrently, the microkernel movement—embodied historically by pioneering systems such as Mach, QNX, and Travis Geiselbrecht's NewOS (the architectural precursor to Haiku OS)—demonstrated that decoupling OS subsystems into distinct, message-passing server tasks yields significant resilience benefits. In high-concurrency enterprise and telecom systems, the actor-model paradigms of Erlang/OTP demonstrated that software fault tolerance could be elevated to a first-class language semantic. Later experiments with unicore unikernels on Type-1 hypervisors (such as LING on Xen) demonstrated the viability of running minimal language runtimes directly on virtualized hardware without the overhead of conventional POSIX userlands.

In 2026, the convergence of autonomous robotics, unmanned aerial vehicles (UAVs), and distributed cloud infrastructure has renewed the demand for ultra-lightweight, formally verifiable real-time foundations. While general-purpose operating systems such as Linux dominate enterprise compute, their non-deterministic scheduling, massive codebase footprint, and complex memory-management subsystems render them ill-suited for real-time hard-deadline tasks.

To address this challenge, we revisit the seminal **TRON** (*The Real-time Operating system Nucleus*) architecture conceived by Ken Sakamura in 1984, specifically focusing on the **BTRON** (*Business TRON*) workstation standard. Unlike conventional Unix or Windows models, BTRON was designed from first principles around a document-centric hyper-data model, lightweight preemptive multi-tasking, rich multilingual character encodings (TRON Code), and a clean separation between real-time kernel primitives (μITRON) and user-facing shell subsystems.

This paper presents the cleanroom reconstruction of BTRON3 SPEC 3.20, designated **B-System** or **B-TRON**, designed for operation within the **Synrc VE OS.1** hypervision environment. By integrating modern VirtIO 1.4 virtualization, a four-tier compilation pipeline, and a constructive Type-Theoretic formal verification methodology, we demonstrate that the classic TRON architectural principles offer an extraordinarily effective blueprint for next-generation safety-critical computing.

2 Operating System Taxonomy and Virtualization

2.1 Tri-Tier Hardware Classification

Modern embedded and industrial computing spans a vast continuum of hardware capabilities. To structure kernel deployment and runtime requirements, we categorize target systems into three architectural tiers:

1. **Tier 1: Nano / Microcontroller (8/16-bit):** Resource-constrained sensors, actuators, and power-harvesting nodes with memory envelopes measured in tens of kilobytes (e.g., AVR, MSP430, small ARM Cortex-M0+). Execution relies on static memory allocation, cooperative or simple priority scheduling, and zero virtualization.

2. **Tier 2: MCU & Edge Embedded (16/32-bit, AArch32):** High-performance microcontrollers and application processors (e.g., ARM Cortex-M4/M7, Cortex-A7, AArch32) driving flight controllers, robotic kinematics, and automotive electronic control units (ECUs). Demands deterministic preemptive scheduling, low interrupt latency, and memory protection (MPU/MMU).

3. **Tier 3: Datacenter & Compute (32/64-bit, AArch64 / x86_64):** Multi-core server silicon, edge hypervisors, and cloud orchestration nodes running heavy workloads, distributed message fabrics, and virtualized guest fleets.

Our work focuses primarily on unifying Tier 2 and Tier 3 using modern 32-bit and 64-bit ARM architectures (AArch32 and AArch64), establishing an isomorphic operational model across embedded flight hardware and cloud-based virtualization testbeds.

2.2 The VirtIO 1.4 Virtualization Standard

Historically, virtualization in enterprise environments was bifurcated between proprietary hypervisors (VMware ESXi) and specialized para-virtualization architectures (Xen). However, due to licensing overhead and ecosystem consolidation, the industry has standardized overwhelmingly on KVM-based hypervisors (such as Proxmox VE) leveraging the open OASIS **VirtIO** standard.

VirtIO defines a standardized, zero-copy, ring-buffer-based transport interface between virtual machine guests and the underlying hypervisor. In our architecture, implementing VirtIO 1.4 drivers for block devices (`virtio-blk`), network interfaces (`virtio-net`), and graphics consoles (`virtio-gpu` / framebuffers) provides two distinct advantages:

- **Universal Portability:** The identical guest OS binary can execute interchangeably on developer workstations, continuous integration (CI) simulation pipelines, and cloud-hosted Proxmox VE clusters.

- **Unified Staging Across Heterogeneous OS:** Embedded RTOS guests (BTRON, NuttX, LK) share an identical I/O driver abstraction with industrial Linux/BSD appliances (Alpine Linux, NetBSD, FreeBSD).

3 The Real-Time Operating System Triad

When architecting avionics and drone platforms, three open RTOS architectures represent distinct points in the design space:

1. **Apache NuttX:** Engineered for maximum standards conformance, providing a rich POSIX API. While facilitating easy porting of Unix software, POSIX abstractions introduce non-trivial overhead in interrupt handling and memory footprints.

2. **Little Kernel (LK):** Designed by Travis Geiselbrecht for bootloaders, Android pre-boot environments, and low-latency microcontrollers. LK prioritizes rapid synchronization, minimal context-switch latency, and an uncluttered threading model.

3. **TRON (μITRON / BTRON):** Designed from inception for real-time determinism (Sakamura's μITRON) coupled with a radical hypertext document paradigm (BTRON). Unlike POSIX, BTRON eliminates the distinction between files and applications, structuring all state as interconnected Real Objects (実身) and Virtual Objects (仮身).

4 B-TRON Cleanroom Architecture

4.1 Specification Compliance and Lineage

Because BTRON3 SPEC 3.20 is an open, formally documented standard, a complete cleanroom implementation can be authored without proprietary code contamination. Our cleanroom project, **B-System**, strictly adheres to the

Table 1: Architectural Comparison of the RTOS Triad

Feature	**Apache NuttX**	**Little Kernel (LK)**	**BTRON / μITRON**
Primary Author	Gregory Nutt	Travis Geiselbrecht	Ken Sakamura
API Standard	POSIX (IEEE 1003.1)	Custom Event-Driven	ITRON / BTRON3 SPEC 3.20
Design Focus	POSIX Compliance	Ultra-low Latency Boot	Document-Centric Real-Time
Data Model	Hierarchical VFS	Flat / Memory Mapped	Hyper-Data (Jisshin / Kashin)
GUI Subsystem	NxWidgets	None (Headless/Display)	Display Primitives & Window Mgr
Multilingual	UTF-8 POSIX	UTF-8 / ASCII	TRON Multilingual Code

BTRON3 SPEC 3.20 hierarchy, structured into standard C99 modules with zero external dependencies beyond a standard C library and display canvas.

4.2 Fundamental TRON Types and Primitives

The BTRON specification establishes a clean, fixed-width type taxonomy:

- `B`, `H`, `W`: Signed 8-bit byte, 16-bit half-word, and 32-bit word.
- `UB`, `UH`, `UW`: Unsigned 8-bit, 16-bit, and 32-bit integer representations.
- `VW`: Generic pointer type (32-bit/64-bit void pointer).
- `ID`, `ER`: Kernel object identifier and signed error code.
- Geometric primitives: `PNT` (x, y), `RECT` (l, t, r, b), `PAT`, and `COLOR`.

Real-time task synchronization is governed by μITRON system calls:

$$\text{Task Lifecycle:} \quad \texttt{cre\_tsk}, \texttt{sta\_tsk}, \texttt{ter\_tsk} \tag{1}$$

$$\text{Synchronization:} \quad \texttt{slp\_tsk}, \texttt{wup\_tsk}, \texttt{can\_wup} \tag{2}$$

$$\text{Semaphores:} \quad \texttt{cre\_sem}, \texttt{wai\_sem}, \texttt{sig\_sem} \tag{3}$$

4.3 The Real Object / Virtual Object Hyper-Data Model

The central architectural innovation of BTRON is the abandonment of the hierarchical tree directory system (e.g., POSIX `/usr/bin/`) in favor of a networked hyper-data model:

- **Real Object (実身 – *Jisshin*):** The actual persistent record containing document content, drawings, or executable structures stored within persistent storage cabinets.
- **Virtual Object (仮身 – *Kashin*):** A visual, interactive pointer embedded within any Real Object that references another Real Object. Clicking or activating a Virtual Object opens the referenced Real Object in-place.

Mathematically, the BTRON storage system forms a directed graph $G = (V, E)$, where vertices V represent Real Objects (*Jisshin*) and edges E represent embedded Virtual Objects (*Kashin*). This allows many-to-many non-hierarchical links, instantaneous transclusion, and multi-parent categorization without symlink anomalies.

5 Multi-Target Execution Matrix

To validate B-System across different deployment scenarios, we implemented a four-backend target matrix controlled via a unified BSD Makefile:

Table 2: B-System Execution Backends

Target	**Environment**	**Primary Purpose**
`Takada`	POSIX (SDL2)	Host CI / Debug
`Izumina`	QEMU VirtIO	VirtIO Driver Testing
`Sakamura`	QEMU TK 2.0	Spec Conformance
`Yokobayashi`	Baremetal RPi 2/3	Embedded Production

5.1 POSIX Light Backend (*Takada*)

The *Takada* backend maps μITRON tasks to POSIX threads (`pthreads`) and Display Primitives to an SDL2 software rendering buffer. This allows instant compilation and sub-second unit test execution directly on developer workstations running macOS, Linux, or NetBSD.

5.2 QEMU VirtIO Backend (*Izumina*)

The *Izumina* backend compiles B-System into a standalone bootable image communicating directly with QEMU's virtualized hardware using VirtIO ring buffers. This verifies that all memory-mapped I/O and interrupt service routines behave deterministically under hypervisor scheduling.

5.3 Baremetal ARM Backend (*Yokobayashi*)

The *Yokobayashi* backend compiles directly for ARM Cortex-A7 (Raspberry Pi 2) and Cortex-A53 (Raspberry Pi 3) processors without any underlying operating system. The kernel directly programs the ARM generic interrupt controller (GIC), the Mailbox property interface for

```
btron/
|-- Makefile              # Plain BSD/TRON Makefile
|-- include/btron/        # Authentic Japanese Headers
|   |-- btron.h           # Master include header
|   |-- types.h           # Core TRON types (W, H, B,
    etc.)
|   |-- itron.h           # uITRON RT-Kernel Primitives
|   |-- dp.h              # Display Primitives engine
|   |-- wnd.h             # Window Manager APIs
|   |-- event.h           # System Event Queue
|   |-- vobj.h            # Real/Virtual Object Engine
|   `-- troncode.h        # TRON Character Code
|-- src/                  # Subsystem Implementations
`-- apps/                 # Accessories (gterm, t_editor
    )
```

Figure 1: Authentic BTRON subsystem and header tree structure.

```
+-------------------------------------------------+
|              BTRON Shell & Apps                 |
|   (Desktop Manager, T-Editor, gterm, vobj_mgr)  |
+-------------------------------------------------+
|         Display Primitives & Window Manager     |
+-------------------------------------------------+
|         BTRON3 SPEC 3.20 System Services        |
|    (Event Manager, Real/Virtual Object Manager) |
+-------------------------------------------------+
|                  uITRON Kernel                  |
+-----------+-------------+-----------+-----------+
|   POSIX   | QEMU VirtIO | QEMU TK2  | Baremetal |
|  (Takada) |  (Izumina)  |(Sakamura) |(Yokobayashi)|
+-----------+-------------+-----------+-----------+
```

Figure 2: Layered architecture and target execution matrix.

framebuffer configuration, and the Mini-UART for serial diagnostics.

6 Formal Verification and Type-Theoretic Hypervision

6.1 Constructive Theorem Proving vs. Ad-Hoc Testing

Traditional software engineering relies on automated unit testing and integration testing. However, Dijkstra's fundamental maxim remains immutable: *testing shows the presence of bugs, not their absence.* In mission-critical aerospace avionics, flight-control software, and hypervisors, runtime kernel panics or concurrency deadlocks can result in catastrophic mission failure.

As demonstrated by the verified **seL4** microkernel, absolute correctness guarantees require constructing a formal mathematical proof that the executable binary is semantically isomorphic to an abstract high-level specification.

6.2 Martin-Löf Type Theory and the Curry-Howard Isomorphism

Our formal verification methodology grounds all kernel abstractions in **Martin-Löf Type Theory (MLTT)** and **Homotopy Type Theory (HoTT)**. Under the Curry-Howard isomorphism, propositions are interpreted as types, and mathematical proofs are interpreted as valid, terminating programs:

$$\text{Proposition} \iff \text{Type } A \tag{4}$$

$$\text{Proof} \iff \text{Term } a : A \tag{5}$$

$$\text{Verification} \iff \text{Typechecking } \vdash a : A \tag{6}$$

In the Groupoid Infinity research ecosystem, we develop verified formal models using dedicated Type Theory engines:

- **AXIO Z.900:** Formal axiomatic base for categorical semantics.
- **HENK Z.911:** Pure Type System (PTS) implementation of Barendregt's λ-cube and System F_ω.
- **FRANK Z.918 / PER Z.928:** Partial Equivalence Relations and verified computational equality.
- **CHRISTINE Z.944 / ANDERS Z.959:** Higher-order dimensional cubical type checkers supporting constructive univalence and higher inductive types.

6.3 Formal Verification of Task Scheduling and Isolation

To formalize the μITRON priority-preemptive scheduler, we model the system state as a tuple:

$$\Sigma = \langle \mathcal{T}_{\text{ready}}, \mathcal{T}_{\text{blocked}}, \tau_{\text{current}}, \mathcal{M}_{\text{spatial}} \rangle \tag{7}$$

where $\mathcal{T}_{\text{ready}}$ is the priority queue of ready task identifiers, $\mathcal{T}_{\text{blocked}}$ is the set of tasks waiting on semaphores or sleep timers, and $\mathcal{M}_{\text{spatial}}$ is the memory protection partition table.

We prove the following core invariants:

1. **Liveness and Progress:** For all states Σ where $\mathcal{T}_{\text{ready}} \neq \emptyset$, the scheduler transition function $\text{step}(\Sigma) \to \Sigma'$ selects the highest-priority runnable task in bounded time $O(1)$.

2. **Spatial Memory Isolation:** For any two distinct tasks $\tau_i, \tau_j \in \mathcal{T}$ $(i \neq j)$, their allocated memory partitions satisfy:

$$\text{AddrSpace}(\tau_i) \cap \text{AddrSpace}(\tau_j) = \emptyset \tag{8}$$

preventing unauthorized memory mutations across task boundaries.

7 Industrial Architecture and Ecosystem

The B-System cleanroom implementation forms one layer within the broader **Synrc VE OS.1** industrial operating infrastructure, organized across five functional architectural planes:

- **Security Plane:** Cryptographic primitives (LibreSSL), memory isolation, and NIST SP 800-53 security control compliance.
- **Control Plane:** Deterministic hypervision orchestration, stage management, and RTOS telemetry for autonomous drone swarms.
- **Web & Visualization Plane:** TAD-native browser, HTML5 bidirectional transcompilation, and NITRO/FORM document applications.
- **Compute Plane:** Bytecode virtual machines, including BEAM (Erlang/Elixir OTP), CLR (C#/F#), and JVM (Scala).
- **Service Plane:** Lightweight microkernel services running Alpine Linux, NetBSD, FreeBSD, and B-TRON guests in isolated Proxmox VE VirtIO slices.

8 Conclusion and Future Work

This paper demonstrated that the BTRON3 SPEC 3.20 architecture, conceived as an open standard over four decades ago, provides an extraordinarily modern and robust foundation when combined with contemporary virtualization and formal verification techniques. By implementing a cleanroom C99 BTRON kernel with authentic Japanese headers, a four-tier compilation pipeline spanning POSIX, VirtIO, and baremetal ARM, and anchoring real-time scheduler semantics in constructive Martin-Löf Type Theory, we have established a viable, high-assurance RTOS for safety-critical drone and industrial edge computing.

Future work includes completing the automated Coq/Lean extraction pipeline for the VirtIO ring-buffer driver and extending the TAD browser to support hardware-accelerated 3D vector figure rendering.

References

[1] K. Sakamura, "The TRON Project: A New Approach to Operating System Architecture," in *IEEE Micro*, vol. 7, no. 2, pp. 8–14, 1987.

[2] BTRON3 Working Group, "BTRON3 Specification Ver 3.20.00," TRON Association, Tokyo, Japan, 1994.

[3] G. Klein, K. Elphinstone, G. Heiser, et al., "seL4: Formal Verification of an OS Kernel," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP)*, pp. 207–220, 2009.

[4] R. Russell, "virtio: Towards a De-Facto Standard For Virtual I/O Devices," in *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.

[5] T. Geiselbrecht, "Little Kernel (LK) Embedded Operating System Architecture," https://github.com/travisg/lk, 2014.

[6] G. Nutt, "NuttX Real-Time Operating System User Guide," Apache Software Foundation, https://nuttx.apache.org/, 2007.

[7] P. Martin-Löf, *Intuitionistic Type Theory*, Bibliopolis, Naples, 1984.

[8] The Univalent Foundations Program, *Homotopy Type Theory: Univalent Foundations of Mathematics*, Institute for Advanced Study, Princeton, 2013.

[9] M. Sokhatsky, "SYNRC NITRO/FORM: A TAD-native X-Forms Framework for BTRON3 SPEC 3.20," Cleanroom Working Group Research Report, 2026.