From Windows Application to Portable Emulation Platform

Architecture, Correctness, and Verified Debugging in VCC Plus

Brian K. Holman
BS, Computer Science, Brigham Young University
MBA, Marriott School of Management, Brigham Young University
me@brianholman.com
VCC Plus Technical Papers · November 18, 2026

Abstract

Legacy emulators preserve detailed hardware models within applications whose operating-system dependencies can obstruct reuse and verification. In VCC Plus, I separate the Windows-oriented Virtual Color Computer engine into explicit processor, memory, machine-service, and video contracts. A serialized native macOS host composes those components. The extraction exposed defects in instruction operand timing, reset state, interrupt lifetime, raster bounds and storage transfers. Additional CMOC corrections account for effective-address dependencies and condition-code liveness that arithmetic-only optimizer reasoning had discarded. The resulting development platform binds debugging interpretations to executable bytes, compiler metadata, physical mappings and immutable machine observations. It supports conservative stack and variable inspection, bank-aware source debugging, interpreter-aware BASIC stepping, historical tracing and bounded call-context inspection. Process-isolated workers add field-counted input, retry-safe control and graphics/audio evidence without sharing the user's running machine. Guest-library ownership contracts and transactional disk editing extend the same discipline to mappings, interrupts and writable media. I evaluate the design through mechanism-specific regressions, observation-invariance checks, a bounded XRoar comparison, and target-executed compiler cases. A correctness-gated optimization study connects measured code size and guest cycles to the tradeoff between frame omission and inspectable storage. The results show how host decoupling and evidence-backed runtime interpretation can reinforce one another while retaining a mature emulator's useful implementation.

Keywords: emulator architecture; legacy software modernization; Color Computer; MC6809; HD6309; source-level debugging; runtime observability; differential testing.

1. Introduction

An emulator is simultaneously a software application and a model of another computer. Its menus, windows, audio devices, and persistent files belong to the host environment; its instruction execution, memory mapping, interrupts, raster generation, and peripheral protocols describe the guest. In a long-lived implementation, those responsibilities often grow together. Hardware models call application utilities, register code reaches debugger windows, and a display surface carries both pixel storage and host-window state. Such integration can be practical within the original application while making later reuse and independent verification difficult.

VCC, the Virtual Color Computer, embodies substantial prior work on the Tandy Color Computer 3 and its MC6809 and HD6309 processor configurations. VCC Plus is an independent evolution of that codebase. My engineering objective was to retain the useful emulation code while supplying a native macOS application, selected expansion-device support, and development workflows appropriate to that host. The work did not begin as a processor rewrite or as a general Win32 compatibility layer. Its architecture instead makes previously implicit relationships explicit and places host responsibilities behind interfaces or native adapters. The retained Windows application and upstream authorship remain part of the project's provenance. [1, 2, 6]

Platform extraction also created an opportunity to examine behavior independently of a running graphical application. Authored memory buses, recording host implementations, headless machine sessions, framebuffer guards, and bounded execution made defects reproducible at smaller scopes. Some defects concerned conventional software safety, such as initialization order or buffer bounds. Others affected guest-visible behavior: the selected processor used for floating-bus reads, instruction timing after HD6309 reset, actual advancement of a multi-sector floppy transfer, or the position of an operand read within a nominally correct instruction-

cycle total. Compiler regressions likewise require attributing wrong program results to generated instructions rather than to input devices or the emulator itself. These differences matter because a successful boot is a weak substitute for a precise account of what has been preserved or changed. [49, 50, 51, 52, 53, 54, 55, 56, 57, 58]

The resulting system also extends substantially beyond host replacement. A debugger needs access to registers and memory, but access alone does not establish the meaning of the observed bytes. A logical address can refer to several physical banks over time; a function symbol may identify an assembly prologue rather than an executing C statement; a stack word can be data rather than a return address; a readable pointer can outlive its object. VCC Plus therefore adds services that bind interpretation to an identified machine, build, mapping, and observation. Their design favors a supported partial answer over an apparently complete but unjustified one. [16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]

I make four connected contributions. First, I document an incremental architecture for separating a retained emulator from its original Windows host. Second, I analyze substantive correctness changes with causal explanations and mechanism-specific regressions. Third, I describe a debugging and development service layer spanning the core, native bridge, compiler integration, guest SDK, and scoped external tools. Its contracts govern both interpretation of state and temporary ownership of guest hardware. Fourth, I establish execution and resource-ownership contracts for isolated programmable investigation, guest hardware services and transactional media editing. I test these contracts and corrective mechanisms against explicit observable expectations.

I organize the report around four questions. **RQ1:** Which boundaries permit host replacement while preserving the useful machine model? **RQ2:** Which correctness problems become observable through extracted subsystem and complete-machine tests? **RQ3:** What evidence is required before a debugger can associate runtime state with source locations, frames, or typed values? **RQ4:** How can the same engine support ordinary interactive use and isolated programmable investigation without conflating their state or authority? Sections 2–4 address the first question and method. Sections 5–7 develop the observation and development services. Section 8 analyzes corrective cases, transactional editing, and an integrated guest application. Section 9 reports the evaluation, and Section 10 discusses what its results establish.

I use the following terms with specific meanings. “Portable” refers to the extracted contracts and implementations exercised by the stated hosts; it does not mean that every remaining source file is independent of its original platform. “Verified” refers to implemented identity, mapping, metadata, and byte-consistency checks; it does not denote a formal proof of emulator or debugger soundness. “Compatible” refers to an identified workload and observed path unless broader evidence is supplied. These distinctions allow a technical account of substantial functionality without converting a successful development checkpoint into an unsupported universal claim.

2. Background and Baseline

2.1. The emulated machine and the preservation problem

VCC models the Tandy Color Computer 3 and selected expansion hardware. Its principal execution engines implement the Motorola MC6809 and the Hitachi HD6309, while the machine model supplies memory mapping, GIME/SAM video and timing behavior, peripheral interface adapters, cartridge routing, and storage devices. These components are interdependent. A processor instruction can address a peripheral rather than ordinary RAM; a peripheral read can acknowledge an interrupt; a memory-map change can alter the executable bytes at an unchanged logical program counter; and an instruction's event scheduling can affect when a video-status bit becomes observable. Consequently, preserving an emulator requires preserving relationships among components as well as their individual algorithms. [1, 2]

The inherited implementation already embodied substantial engineering knowledge about these relationships. The purpose of VCC Plus was therefore to retain and qualify useful implementation rather than reconstruct the machine through an unrelated rewrite. The project introduced a native Mac application around the existing C/C++ engines and selected device models. SwiftUI and AppKit provide the application interface; an Objective-C++ boundary connects that interface to the machine owner; Metal presents completed images; and native audio facilities consume samples generated in emulated time. The architectural problem was to make those substitutions without allowing the replacement host facilities to redefine the emulated hardware. [1, 3]

I distinguish *host portability*, which concerns where the emulator can run, from *guest compatibility*, which concerns how Color Computer programs behave inside it. The host application and the emulated processor have separate responsibilities in the design. [1, 2]

2.2. Source baseline and inherited Windows integration

I examine the VCC Plus implementation and engineering records reviewed on November 12, 2026. Source citations identify files in the VCCPlusOSS distribution; named product versions and validation dates distinguish the evaluated configurations. Historical development records describe earlier extraction and integration checkpoints; their measurements remain associated with those checkpoints. I use the current source to establish the implemented interfaces and ownership relationships, and dated records to establish the corresponding historical observations. This separation avoids treating an early diagnostic configuration or an earlier suite's test count as a description of every later version. [1, 2, 3, 4, 5]

The original application integrated hardware emulation with Windows application state and services. Processor code depended on debugger state maintained by the application; video code accessed display state; ROM selection and loading were associated with Windows configuration and file handling; and expansion devices were integrated through dynamically loaded libraries and native menus. These dependencies crossed the intuitive boundary between a hardware model and its user interface. Replacing only the main window would therefore have left application assumptions embedded in the execution path. [1, 2, 6]

Windows reference binaries and retained source require separate qualification: a pre-existing release binary cannot be assumed to represent every later source change. Selected Windows workload observations are therefore distinguished from authored architectural expectations and Mac integration tests. Upstream authorship, notices and the retained Windows host establish provenance; complete runtime parity requires additional comparison.

The revised repository separates shared implementation under `src/core/` and `src/devices/` from the native Mac host under `platforms/macos/` and the retained Windows implementation under `platforms/windows/`. This organization makes responsibility more visible, but directory placement alone does not prove platform independence. Shared host-adapter files still contain conditional Windows implementations, and some public C++ names retain the `VCC::Mac` namespace to avoid an unrelated naming refactor. Portability is established at the selected build and interface boundaries rather than inferred from directory or namespace labels. [6, 7, 8, 9]

2.3. Scope of the architectural transformation

I set out to build a native macOS application around the retained emulator. File panels, application settings, keyboard delivery, display submission, audio output, and asynchronous user-interface updates were implemented using host-appropriate facilities. Existing machine algorithms were retained where practical and connected through explicit operations. The project did not require Windows DLL binary compatibility, a third-party Mac plug-in ecosystem, physical floppy passthrough, or automatic parity with every historical expansion module. Device support was chosen according to the product scope and qualified against documented workloads. [1]

This choice separates hardware compatibility from implementation packaging. A Multi-Pak Interface is an emulated arrangement of devices, signals, and routing registers; it need not be implemented by loading a Windows DLL. Conversely, replacing a DLL with a native menu is insufficient if the device's register behavior, interrupt participation, or independent routing is lost. The Mac architecture therefore preserves the relevant device and bus contracts while replacing operating-system-specific discovery, configuration, and presentation. [1, 10]

3. Engineering Objectives and Method

3.1. Preservation through explicit boundaries

My first objective was to retain the inherited execution and device logic while making dependencies explicit enough to compile, execute, and test the selected machine without a Windows application. My second objective was to integrate that machine into a responsive native interface without introducing unsynchronized access to legacy global state. My third objective was to create inspection and development facilities that could make justified claims about the executing program. These objectives are related: a debugger cannot produce consis-

tent observations if the host can read changing core state directly, and independent subsystem tests cannot isolate a defect if application services remain implicit. [2, 3, 4, 5]

I organized the extraction in stages of increasing integration. Processor checks initially used an explicit flat-memory bus to exercise instruction behavior. Machine checks then used the real MMU, hardware registers, and interrupt paths while replacing external connections with recording hosts. Graphics checks added the real renderer and event scheduler. A synchronous session owner connected the resulting components, and the asynchronous native bridge exposed controlled commands and owned observations to the application. Real-ROM boot and selected software workloads followed these authored checks. For each layer, I record the replacement components and the limits of the evidence. [2, 3, 4, 5, 11]

The flat test bus is especially important to this method. It is an explicitly installed override rather than the application's normal memory implementation. Ordinary CPU accesses dispatch through the real MMU when the override is absent. Isolated checks detach the override before a machine integration runs, and the synthetic bus is excluded from the application target. Thus, a test harness can isolate instruction behavior without silently changing the machine shipped to users. [2, 7]

3.2. Change classification and observable contracts

I distinguish three classes of change. A behavior-preserving extraction changes where a dependency is expressed while retaining the intended machine behavior. A corrective change alters an implementation shown to violate a stated expectation, such as reset state or an observable instruction-sampling point. A host or device adaptation introduces a deliberate policy, such as bounded work or deterministic media handling, whose correspondence to the inherited implementation must be stated rather than assumed. New native integration can also contain defects of its own; those are not attributed to upstream VCC merely because the integration uses its machine engines.

I define the contracts through observable effects. For example, the retained CPU execution functions return unused cycles, and a final instruction can exceed a requested budget. A caller must therefore interpret a negative remainder as an overshoot, rather than assuming the request is an exact instruction boundary. A soft reset preserves RAM at the reset boundary but does not guarantee that guest warm-start code will leave every byte unchanged after execution resumes. A frame snapshot owns a completed image but is not a serialized machine checkpoint. Defining these distinctions constrains both implementation and evaluation. [2, 5, 12]

I combine authored fixtures with integration workloads in the validation. Authored fixtures provide known memory contents, instruction sequences, register transitions, and expected timing relationships. They permit source-controlled reproduction without redistributing private ROMs or applications. Integration workloads demonstrate that the composed implementation performs a useful sequence, such as booting an interpreter and executing a program through emulated input. Neither layer replaces the other: a booting application may miss a rare arithmetic defect, while a collection of isolated register checks may miss an incorrect connection between correct components. I report the narrowest conclusion supported by each result. [2, 3, 4, 5, 11]

3.3. Evidence and reproducibility boundaries

I distinguish source inspection, historical execution records, and new demonstrations prepared for the paper. Source inspection establishes that a mechanism exists at the recorded revision. A dated validation record establishes that a particular check was reported as executed in its stated environment. An illustrative harness can explain or exercise a selected algorithm without establishing complete-machine accuracy. The architecture described here relies on the first two forms; it does not imply that all historical experiments were rerun during preparation of the manuscript.

Some integration fixtures require separately supplied ROMs or software. Their absence is a reproduction dependency, not a reason to substitute synthetic evidence while retaining the stronger compatibility claim. The repository therefore keeps public authored checks separate from optional-input probes and preserves notices and provenance for imported implementation. Broader source publication and distribution of third-party payloads remain distinct matters. [1, 2, 11]

4. Architectural Refactoring and Host-Platform Decoupling

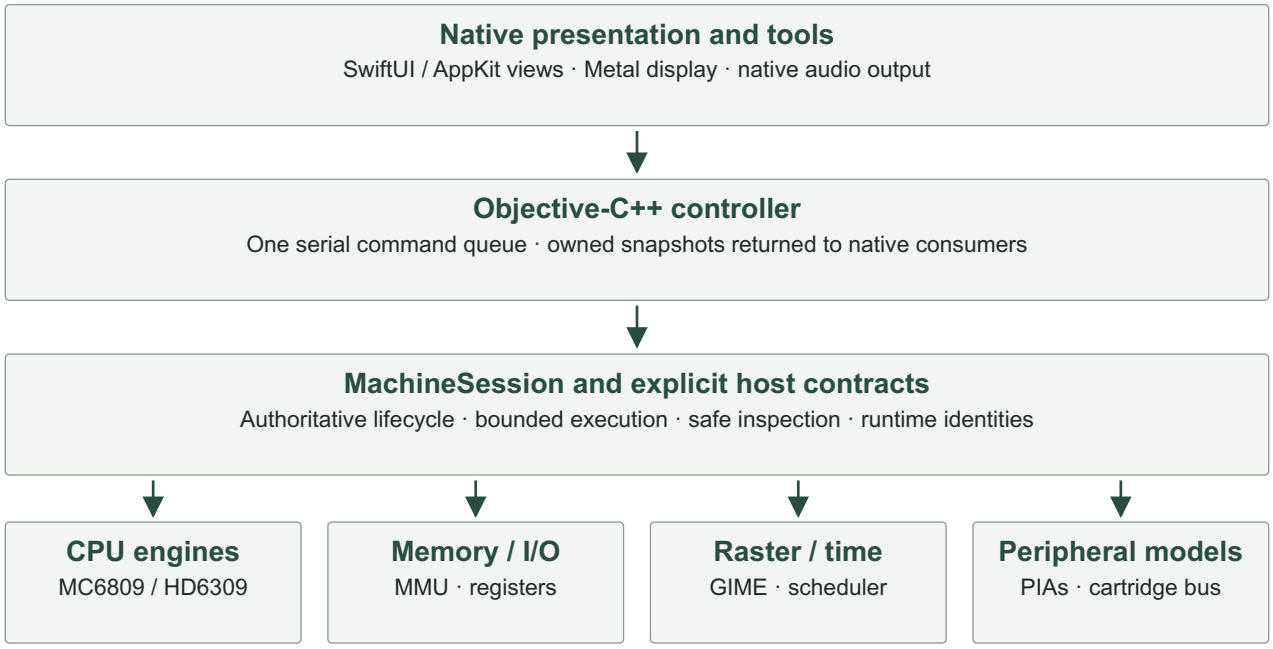


Figure 1. Ownership boundaries in the native interactive process. The disposable worker composes a separate machine in its own process.

4.1. Processor memory and debugger interfaces

`CpuMemory.h` states the byte, word, and map-selection operations used by the processor engines. `CpuMemory.cpp` dispatches to `MmuRead8`, `MmuWrite8`, and the real map-selection implementation unless a `CpuMemoryBus` override is installed. Word accesses compose byte operations in guest order, including 16-bit address wrapping. The separation removes the need for a processor engine to know how a host window, configuration object, or test executable obtains memory. It also preserves I/O routing through the machine's actual address decoder during normal execution. [7]

`CpuDebugHooks` performs an analogous extraction for execution observation and control. Its synchronous interface includes halt state, stepping, breakpoint eligibility, source-step notifications, trace capture before and after execution, interrupt observations, video-event notifications, and diagnostics. The CPU engines can invoke these hooks without including debugger-window definitions. On Windows, the default implementation forwards to the retained application debugger. On non-Windows builds, the default supplies a minimal halt latch, while the native machine owner installs the hooks required by its debugging facilities. [8]

The hook interface is an integration contract, not a synchronization mechanism. The host owns the installed object and must retain it through execution. Installation and detachment occur outside active execution on the owning thread. This requirement prevents an abstract interface from being mistaken for thread safety: an interface can remove platform includes while leaving the original processor state global and nonreentrant. The same distinction applies to the isolated memory bus. [2, 7, 8]

4.2. Machine-service dependencies and ROM provisioning

`MachineHost` makes the remaining connections from the MMU, PIAs, and GIME/SAM registers explicit. Its required methods cover internal-ROM provisioning; cartridge memory and port access; cartridge audio; video and timer register delivery; CPU speed selection; selected-CPU state; interrupt assertion and clearing; keyboard scans; joystick strobes; tape input and motor signals; and printer bits. A recording implementation can supply known inputs and observe outgoing requests, while a complete machine implementation connects them to actual models. Required methods prevent a missing device connection from being silently interpreted as a successful hardware operation. [3, 9]

The native path explicitly fails if a machine host has not been installed. The Windows adapter instead forwards the contract to the existing implementation. Windows-specific printer handling was moved out of the PIA source into the Windows host. This is a targeted dependency inversion: the hardware model expresses the operation it requires, and the host supplies a platform-appropriate connection. It does not emulate Win32 functions through a universal substitute. [3, 9]

ROM provisioning illustrates the allocation and ownership implications of the boundary. `MmuInit` validates the RAM selection and prepares replacement RAM and ROM before replacing the active allocation. The ROM provider receives a fixed-size destination prefilled with `$FF`; file selection and validation belong to the host. A failed allocation or provider operation preserves the previously active memory, and repeated shutdown is supported. A successful reinitialization or shutdown invalidates borrowed RAM and ROM pointers, so consumers must not cache them beyond the corresponding machine lifetime. The resulting contract makes failure behavior explicit rather than allowing a file operation to partially replace machine state. [3]

4.3. Rendering independent of window systems

The retained renderer computes pixels from emulated RAM and video-register state. `VideoState` separates those data and host-owned surface pointers from window-system handles, and `VideoHost` supplies surface access, clearing, artifact-color policy, active-area reporting, and frame-completion notification. The Windows `SystemState` retains access to the extracted fields through its adapter. The Mac path uses a 640 × 480 array of packed 32-bit `0x00RRGGBB` values, with pitch measured in pixels. The zero high byte is not transparency: the native display treats the pixels as opaque. [4, 13]

The bridge publishes an owned image so that later execution, reset, or shutdown cannot mutate an already delivered frame. Metal presentation then handles drawable dimensions and display scaling. In the native boot implementation, each submitted GPU operation owns its upload texture; presentation may skip when the bounded in-flight submission capacity is occupied rather than blocking the main thread or overwriting an upload still in use. Presentation backpressure therefore affects which completed frames are displayed, not the machine's CPU-to-raster clock relationship. [5, 11]

Event lifetime is part of this separation. A scheduled renderer owns raster and timer event tags, cancels them when stopped, and uses lifetime guards so that abandoned callbacks cannot dereference a destroyed renderer. Reset must replace the old event set rather than layering a second raster schedule on top of it. Immediate debugger preview renderers do not acquire the live raster schedule. These rules support observation and reconfiguration without allowing an auxiliary image to become another authoritative machine clock. [4]

4.4. Bounded execution and authoritative emulated time

`RunFrame` is a synchronous core operation with four relevant completion outcomes: completed frame, debugger halt, exhausted work budget, or stalled processor. It repeatedly requests processor execution, converts the returned unused-cycle count to actual progress, and returns if that progress is nonpositive. A budget-limited partial frame can be resumed; the function does not reset the raster, wait for a window, flush audio, or pace wall-clock time. The final instruction may overshoot the requested cycle budget according to the retained CPU contract. [4, 12]

Separating bounded execution from pacing resolves an important ambiguity in graphical emulators. A host presentation callback is not an instruction clock, and a window resize is not a reset request. The native host decides when it is appropriate to request another bounded unit of work, while the core's event scheduler maintains the relationships among CPU execution, raster events, timers, and scheduled sample generation. Frame presentation intervals can omit host updates while guest devices continue to run. [1, 4, 5]

The audio path follows the same principle through a different host constraint. Scheduled emulated events produce samples; an `AVAudioSourceNode` callback drains a fixed-capacity single-producer/single-consumer buffer. The callback does not enter the emulator, allocate memory, acquire locks, or wait for the interface. Device consumption supplies pacing feedback to the machine queue, which refills a bounded reserve. Silent pacing is available when native output is disabled or unavailable. This design accommodates the output device without allowing its callback timing to rewrite guest clock ratios. The reserve and missing-sample policy address host scheduling variability; they are not claims of unlimited immunity to operating-system stalls. [14]

4.5. Device objects and cartridge topology

Portable device objects replace Windows DLL loading in the native application. The FD-502 adaptation, for example, owns registers, transfer state, drives, images, and head positions while leaving host file dialogs and persistent-file operations outside the controller model. Device ownership is separate from an active slot assignment: disconnecting or moving a controller need not destroy its mounted image or discard modified bytes. Safe configuration changes still require the appropriate paused boundary and media checks. [1, 10]

The cartridge bus preserves topology rather than reducing the Multi-Pak to a menu of mutually exclusive devices. Its routing distinguishes the ROM slot from the disk-port slot, permits eligible active devices to receive the appropriate other-port traffic, advances devices on actual horizontal-sync callbacks, and retains the relevant interrupt behavior. The physical startup selector establishes reset routing; software register writes establish current routing. Persisting the startup selection does not serialize the current running program or turn runtime routing into a separate machine. [10]

This organization supports independent device testing, but it does not establish arbitrary multi-machine execution. Some adapted device models have isolated instance state while the CPU engines, scheduler, and host bindings remain process-wide. The architecture therefore makes a narrower promise: selected device implementations are independently testable, and one authoritative loaded machine coordinates their use within each process. [2, 5, 10]

4.6. Session ownership and the native bridge

`MachineSession` is the synchronous C++ owner of the composed machine. An atomic occupancy claim permits one loaded session and prevents a second session from replacing the first session's host bindings. It is a lifecycle guard, not a lock around all emulator state. Initialization, command execution, inspection, host replacement, and destruction must still be serialized. Low-level core tests cannot safely run concurrently with a loaded session in the same process. [5]

The native `VCCMachineController` enforces the corresponding application rule through a private serial dispatch queue. User-interface commands and execution callbacks enter that queue, and completion handlers carry owned snapshots to the main queue. The interface does not dereference live core pointers. A snapshot can therefore include CPU and run state, configuration, frame and cycle counters, generation and program identities, and an immutable completed frame without requiring a view to understand the internal emulator locks or global variables. [5, 15]

Lifecycle operations have distinct semantics. Pause stops further execution without replacing the loaded machine. Cold reset reconstructs the machine from retained configuration and ROM, while soft reset follows the retained warm-reset sequence and preserves RAM at its boundary. Both reset paths advance identities used to distinguish old observations from the new timeline. Synchronous session shutdown is idempotent and permits a later load, whereas bridge shutdown is terminal: delayed queued work cannot restart a bridge after application teardown. This distinction makes ownership visible across the language boundary. [5]

Compile-time checks make the retained register layout and integer-width assumptions explicit at the host boundary. Complete reentrancy, big-endian support, exhaustive aliasing review, all historical Windows devices, and complete Windows/Mac trace parity are outside the demonstrated boundary. The result is a selected shared engine with explicit host contracts and a native integration, providing a basis for the richer observation and development services examined in the following sections. [1, 2]

5. Runtime Observability and Debugging Services

I used the extracted core as the foundation for the new debugging tools. Consistent observation also required an explicit state boundary. A debugger must also determine which machine state it observed, whether a memory read changes hardware behavior, and whether its interpretation remains applicable after execution or mapping changes. VCC Plus addresses these requirements through a set of services distributed across the shared C++ session, native bridge, and development models. Their common design principle is that a visible value or source association must be supported by a specific captured state and an identified program. [16, 17]

5.1. Consistent observations at a serialized execution boundary

The session owns execution and inspection on one serial queue. A debugger request captures registers, execution counters, run state, effective mapping, and selected memory before another machine command is

allowed to interleave. The resulting buffers are owned by the observation rather than borrowed from mutable emulator allocations. Native views consume these observations through the Objective-C++ bridge; they do not dereference the running core. This discipline prevents a display of registers from one instruction boundary alongside memory from a later boundary, and allows an inspector to retain an observation across subsequent execution or reset. It does not make the legacy global state reentrant; its validity depends on the serialized owner described in Section 4. [16]

The distinction between a consistent observation and a complete history is important. A snapshot describes one boundary. Selecting an older caller in a call-stack view changes how the same captured state is interpreted; it does not recover the memory that existed when that caller originally invoked its callee. A pointed-to object or global may therefore contain modifications made by the current callee. Historical execution is provided separately through retained trace records, not inferred from current memory. [17]

The main application and auxiliary tools share the same machine authority. Polling transports observations without giving views ownership of execution or media operations. A delayed refresh cannot redefine guest time, and closing an inspector does not undo an armed machine operation. [16, 18]

5.2. Side-effect-free logical and physical memory inspection

Memory-mapped I/O makes an ordinary emulated read unsuitable as a universal debugger primitive. A read can acknowledge a PIA interrupt, consume a device datum, or otherwise alter the state being examined. The inspection services consequently distinguish accessible memory from locations whose value cannot be obtained safely. `MachineSession::debuggerSnapshot` associates each logical byte with an availability flag. Its safe path includes ordinary memory, a non-destructive GIME register view, and system-vector reads; other I/O locations remain unavailable. A placeholder byte in the transport is not a claimed device value. [16]

Physical inspection is a separate operation. It reads installed RAM without mapping it into the CPU address space and without substituting ROM or device values. This supports examination of banks hidden by overlays or inactive MMU tasks. The debugger reports the effective backing selected by the core, including special vector windows and mirroring, rather than attempting to derive every result from the displayed register values. Logical and physical addresses therefore remain distinct even when their numeric values happen to agree. [18, 19]

The APIs deliberately impose different range contracts. The snapshot service can capture a bounded logical view with 16-bit wrapping semantics, whereas explicit memory-range capture rejects wrapping or overflowing ranges. Physical snapshot requests are limited and truncate at installed-RAM bounds; explicit range exports require the complete selected range to be valid. These contracts are stated separately because silently applying one address-space convention to another would produce misleading dumps. A mapped export containing unavailable I/O fails rather than replacing unknown values with zeros. [16, 19]

Writes remain explicit debugger operations with their own preflight. The conservative RAM-edit path validates an entire bounded span before modifying it and excludes I/O, ROM, and unsupported high-memory regions. Observation services never acquire write authority merely because an inspector can display a corresponding address. This separation also makes it possible to expose a narrower external inspection interface without exposing native memory editing. [16, 20]

5.3. Program identity and rejection of stale commands

A logical address is not a durable identity for source code. Reset can replace the machine timeline, a load can replace executable bytes, a debugger edit can invalidate the retained build, and MMU changes can select different RAM at the same address. VCC Plus carries several distinct identities to represent these changes. Machine generation identifies lifecycle changes; program revision identifies explicit code replacement or editing; the mapping fingerprint summarizes the effective CPU address space; expected instruction bytes connect source associations to actual code. The program counter identifies the execution point within that combination. [16, 17]

A source-step request supplies the expected generation, program revision, mapping fingerprint, and PC. The session rechecks those facts on the execution queue before accepting the request. The instruction set supplied for a step must also be bounded, nonoverlapping within its declared module, and consistent with verified executable evidence. Failure of any precondition results in an explained refusal instead of execution against

a plausible but obsolete source map. For flat programs, unexpected effective-mapping changes invalidate the step; declared banked programs use module-aware rules described in Section 6. [16, 17]

These identities establish a conditional association between captured state and retained code. They do not authenticate third-party programs or formally prove hardware state. A previously successful compilation cannot by itself authorize interpretation of later RAM. [17, 20]

5.4. Controlled execution with finite work

The bounded frame runner returns on a completed frame, debugger halt, exhausted cycle budget, or lack of CPU progress. A partial frame can resume without restarting the raster. This makes a debugger stop an ordinary execution result, rather than a condition that can strand a host loop waiting for a frame that will never complete. The final instruction may exceed a nominal cycle budget because the engine preserves its instruction-execution contract; the budget limits further work, not the indivisibility of an already started instruction. [16]

Source stepping builds on normal guest execution instead of simulating source-language statements in the host. Step Into seeks the next mapped line, Over follows the current call without suppressing explicit breakpoints, and Out follows the supported return path. Interrupt handling continues through the ordinary engine. Independent instruction, frame, and depth limits prevent an input wait, nonreturning subroutine, or malformed control flow from becoming an unbounded UI command. A user pause cancels the pending step. A limit-induced pause reports its actual reason rather than inventing a source statement at an arbitrary mid-operation location. [17]

Instruction stepping retains meaningful processor-specific behavior. An HD6309 TFM block transfer can span many internal transfer iterations and raster boundaries. The explicit instruction-step operation completes the instruction while device time continues to advance, and frame-completion accounting records actual boundaries. Normal continuation and tracing retain the engine's transfer-byte behavior. The different granularity is reported rather than flattened into a misleading universal definition of a step. [18]

5.5. Historical instruction and hardware-event tracing

The trace service retains pre-execution instruction bytes, before/after CPU registers, instruction-cycle information, and optional hardware or scheduler events. It does not reconstruct past instructions by reading current RAM. Consequently, self-modifying code, later MMU changes, or debugger edits cannot rewrite an already captured row. A trace that reaches its configured bound stops recording while leaving guest execution in its current run state. Triggering a trace and setting a breakpoint are separate operations: a stop-before trace trigger ends recording before the identified instruction but does not itself prevent that instruction from executing. [21]

Interrupt requests, servicing, and vector-entry events come from the corresponding CPU hooks. These are distinct observations: an asserted request does not prove that the processor accepted it. HSYNC, VSYNC, and video-register rows likewise originate in the renderer and register hooks. The implementation does not infer missing events from plausible register values, and it does not recreate the old Windows scheduler's private variables as if those variables were native events. [21]

Rows use an emulated master-clock timeline, with instruction cycles retained as a separate quantity. A 263-line raster with 912 master clocks per line yields a nominal frame period of 239,856 master clocks in the retained model. This permits comparisons across ordinary guest CPU speed changes without confusing CPU cycles, video fields, and wall-clock time. It is not a measurement of display latency or host scheduling performance. [21]

Capture is finite, up to 100,000 combined instruction and event rows in the documented interface. The viewer retrieves immutable pages, and export copies a selected range before performing formatting away from the machine queue. Reset stops recording while retaining collected samples; replacement of the machine clears the old capture. These rules make lifecycle boundaries visible and prevent a growing trace or a slow save operation from becoming an implicit owner of the emulator. [21]

5.6. Graphics inspection as a memory service

The displayed framebuffer is useful evidence of what the renderer completed, but it is not sufficient to inspect a drawing buffer that the guest has not yet displayed. VCC Plus therefore provides a graphics-memory interpretation service in addition to normal frame delivery. It can inspect the current video page or render an explicitly selected logical or physical memory range using the supported text, semigraphics, PMODE, and

HSCREEN interpretations. Pixel inspection connects a visible coordinate to its stored byte, address, and bit mask. Attributed text also exposes the attribute byte. [19]

The preview does not advance the raster or change the guest's displayed page. Logical and physical origins remain explicit, unavailable I/O is marked, and declared custom GIME canvases preserve relevant scrolling and row-stride information. A paused paired capture can describe the displayed buffer and one manifest-declared buffer from the same state, which is useful for diagnosing double buffering without changing the program to expose its hidden page. A decoded pixel color is an interpretation of captured memory and video state; composite artifact colors may depend on neighboring bits. [19, 22]

The related disassembler also operates on captured bytes. Physical disassembly accepts an assumed logical origin for relative branches without actually mapping the selected RAM. Optional OS-9 interpretation annotates recognized calls and module headers, but it does not assert that every numeric service exists in the running OS or that a header has a valid complete module CRC. These tools preserve the distinction between a useful annotation and a validated guest object. [19]

6. Language-Aware Debugging and Toolchain Extensions

I start from the bytes and registers obtained through machine inspection. Source debugging requires a further chain of interpretation: source text must correspond to compiler output, compiler output must correspond to the linked artifact, and the artifact must correspond to the code currently mapped in RAM. VCC Plus implements this chain in native development services using core observation and execution primitives. These services are part of the overall platform, but their source parser, metadata interpreter, and UI are not represented as features of the CPU engines themselves. [17, 20]

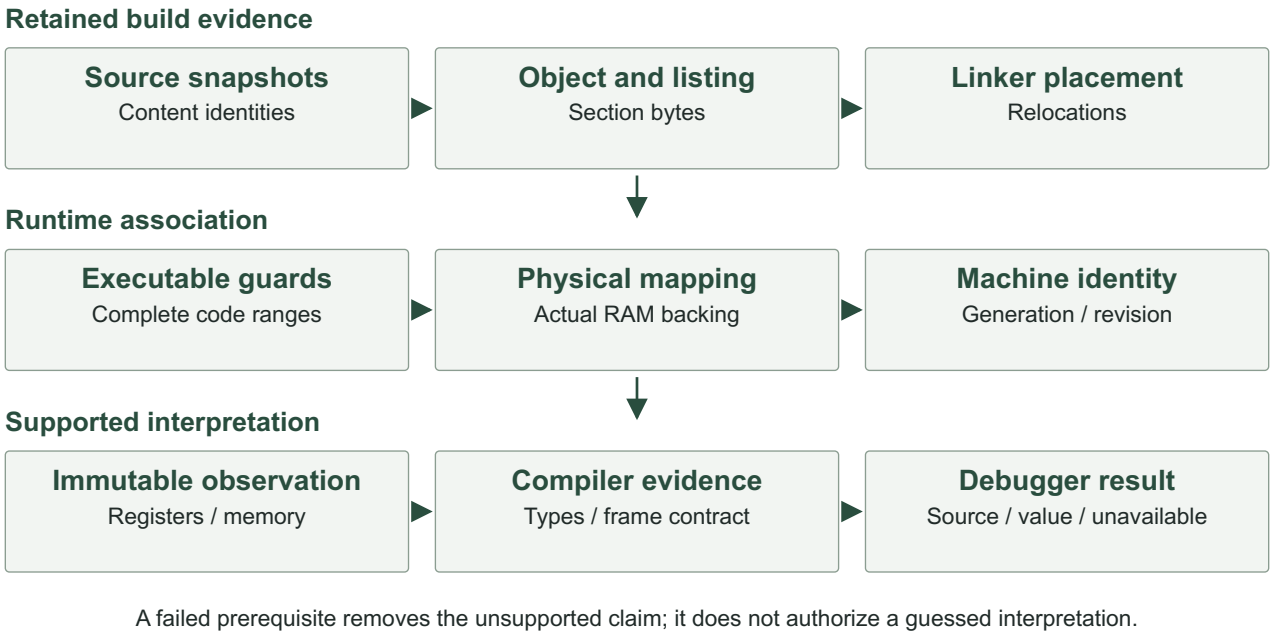


Figure 2. Evidence categories used to associate retained build artifacts, actual runtime code, and supported debugger interpretations.

6.1. Source maps anchored to executable evidence

`CMOCSourceMap` and its project-level companions associate source lines with final instruction bytes. The verifier checks source snapshots, listings, objects, linker placements, and executable artifacts rather than accepting a listing address in isolation. Supported object relocations are replayed against the selected section bases and compared with the delivered initialized bytes. Missing or ambiguous symbols, invalid relocation expressions, overlap, and incompatible listing bytes fail verification. The resulting source map therefore identifies the bytes after linking, including operands whose values differ from the unresolved object listing. [23]

The build receipt exposes this distinction directly. A successful compiler or Make process does not necessarily produce usable debugging evidence. The documented statuses distinguish an executable/source association with usable typed metadata, an executable/source association without that metadata, and unavailable debug-

ging with an explanation. A source declaration, a mapped C statement, and a verified executable entry are also separate catalog entries. This avoids presenting an assembly entry or a function declaration line as if it were a currently executing C statement. [20]

For optimized programs, executable symbols can remain useful even when local variables or statement mappings are unavailable. Named symbol-entry stops permit inspection of verified code without manufacturing a C frame or source location. The service selects advertised unique identities rather than accepting an arbitrary address through the external interface. Native manual address breakpoints retain their separate role. [20, 22]

6.2. Compiler metadata for types, scopes, and storage

The CMOC extension emits versioned `VCCDBG1` records from the compiler's resolved declarations and types. The records describe function entry, frame-ready body, and epilogue boundaries; lexical scopes; signedness and storage widths; arrays, members, pointers, and floating encodings; linker-resolved global storage; and U-relative parameter/local locations. Zero-byte assembler labels anchor ranges without adding instructions or data. [24]

The metadata is an additional evidence source, not authority by itself. The application resolves its symbols through objects and matching linker placement and accepts it only within the retained build identity. Ordinary release compiler invocations omit these debug labels and records. Debug labels can affect optimization opportunities even though they occupy no bytes, so the project does not claim that enabling source debugging is semantically identical to every optimized build. Supported local inspection requires an appropriate unoptimized build with the preserved frame convention. [17, 24]

6.3. Conservative call stacks and typed values

The native C unwinder supports ordinary CMOC calls with a preserved U frame. It verifies the relevant prologue/epilogue convention, readable frame links, source scope, and the direct call instruction associated with a saved return address. It does not scan arbitrary stack words for addresses that happen to resemble code. An unsupported interrupt, assembly, optimized, indirect-call, or unmapped boundary terminates reconstruction with a reason. Thus a partial stack denotes the verified suffix of a call chain, not proof that its oldest displayed function is the program root. [17, 25]

Caller selection is an observation operation. It changes the source location and scope used by Variables and Watches, while stepping continues from the real leaf execution point. Registers, cycles, PC, and memory are not restored to the selected caller. This rule avoids a common ambiguity in source debuggers: the selected frame is a view context, not a second execution cursor. [17]

Typed values are decoded from one paused memory capture using compiler-provided storage and scope. Arrays and aggregates are paged and bounded; pointer expansion is explicit, pointer cycles terminate, and unsupported children do not invalidate readable siblings. A pointer value does not establish an allocation length or the continued lifetime of an object. The watch evaluator supports read-only integer expressions and selected address/member/index operations without running guest functions, assignments, getters, or hardware reads. Its calculated arithmetic uses checked 64-bit host integers, a documented inspection language rather than a claim to reproduce CMOC's complete target promotion and overflow rules. [17]

Stored floating-point inspection depends on explicit encoding metadata. The implementation supports the CoCo BASIC five-byte representation and big-endian IEEE binary32 selected by the applicable compiler option. It does not infer a format merely from a type name, current ROM, or byte count. Missing and inconsistent metadata leave the value unavailable. Decimal formatting exposes the stored approximation; displaying more digits than a familiar rounded literal can therefore reveal representation rather than an arithmetic failure. Floating-point expressions remain outside the integer watch evaluator, and decoding does not invoke a guest math runtime or imply that an optional runtime has been executed. [17, 24]

6.4. Banked and packed executable identity

Banked programs require a stronger identity than the pair of source file and logical address. A loader, resident function, or alternate overlay can present the same opcode at the same CPU location while referring to different physical storage. The banked manifest binds a delivered binary and disk to source hashes, object/listing evidence, linker sections, unpacked module bytes, and explicit logical-to-physical page assignments. Actual physical backing is checked after hardware mirroring; a written MMU register value alone is not the final identity. [22, 26]

Executable sections have complete byte guards, including assembly outside mapped C lines. Mutable data and BSS retain section ownership but are not falsely required to remain equal to their load-time values. Modules that reuse logical addresses must have distinguishable physical ownership, and ambiguous simultaneous claims are rejected. The application independently replays supported object relocations and checks live code instead of treating the manifest's hashes as self-proving declarations. [22, 26]

A disk-source attachment does not install the program by altering RAM, PC, or banks. Normal guest loading establishes the program; a source stop becomes eligible when its own module is mapped and its complete code matches. Expected declared bank transitions can preserve the attachment, while wrong-page selection or changed code makes the corresponding source interpretation unavailable. The debugger does not remap hidden storage to retrieve a global. This preserves the program's execution context while permitting useful inspection of overlays as they become active. [17, 22]

6.5. Interpreter-aware BASIC services

BASIC debugging follows a different contract from compiled C. `BasicProgram` and the session services recognize a qualified interpreter and memory layout, tokenize editor source, and retain source associations for the tokenized program. Load and Run validate the entire replacement against interpreter state, mapping, program space, and source constraints before writing program memory. They operate through the existing ROM interpreter and retain mounted media and machine configuration. Continue is explicitly distinct from Run: it resumes the existing program without clearing variables or GOSUB/FOR state. [27]

Line-entry breakpoints use a validated interpreter dispatch point and source/program correspondence. Statement stepping can identify separate statements on a colon-separated line and distinguish dispatched THEN/ELSE commands. Over and Out track supported GOSUB behavior, including recursion, without mistaking FOR/NEXT records for subroutine frames. Explicit breakpoints take precedence, and a finite emulated-frame limit prevents INPUT waits or nonreturning subroutines from monopolizing a command. Runtime errors and normal returns to the prompt terminate the corresponding operation with an appropriate location or status. [27]

Tokenization records positions in the editor's original source so a highlight can preserve spelling, spacing, and line-ending conventions. Reading a listing back from guest memory is a different operation and produces a canonical representation. Source edits detach obsolete execution associations, and runtime changes to line links, dispatcher identity, or relevant mapping likewise invalidate the attachment. Unknown ROMs, arbitrary cartridges, and OS-9 are not accepted merely because an address resembles a known BASIC workspace. These restrictions make the interpreter-specific facility useful without claiming a universal BASIC debugger. [27]

The historical interpreter documentation supplies an independent description of the underlying mechanism. *Color BASIC Unravelled II* identifies the linked tokenized-line format and the dispatcher at `$ADC6`, including `CURLIN` at `$0068` and the `CHARAD` word at `$00A6`; its GOSUB/RETURN listing documents the interpreter state retained across a subroutine. The Extended and Disk BASIC disassemblies show their separate entry paths calling the common dispatcher. These edition-specific references explain the hook design; runtime qualification of a particular ROM remains a project responsibility. [74, 75, 76]

6.6. Assembly workflows and guest-facing libraries

The assembly workspace reuses the machine's instruction debugging and executable guards while providing native source documents, assembler output, source associations, processor checks, and binary or Disk BASIC export. It does not require a separate emulation state. A changed source project is rechecked before continuing with its retained associations, and entry debugging operates on the loaded program identity. This establishes a direct workflow for code whose meaning cannot be represented honestly as C statements. [28]

The guest SDK extends the development platform in another direction. Libraries expose reviewed CoCo operations for GIME graphics, banked transfers, joystick input, Speech/Sound, Orchestra-90, SDC, printer, and cassette, together with a bounded C compatibility subset. These execute as guest software using the modeled hardware and supported ROM services; they are not general privileged shortcuts from a guest program into native host memory. Original VCC Plus libraries and unchanged third-party libraries retain distinct provenance and terms. Their examples also provide integration workloads that connect compiler output with real device paths, but successful SDK use does not establish exhaustive emulation fidelity. [29]

6.6.1. Banked assets and restoration of machine context

The bank library provides a copy-oriented interface to an explicitly reserved 32 KiB asset region, physical pages $28–$2B ($50000–$57FFF). A caller supplies an ordinary C buffer and an offset within this region rather than retaining a pointer into a temporarily mapped bank. This distinction matters because a 16-bit guest address identifies the currently selected CPU mapping, not a permanent location in expanded physical RAM. The library accepts only a supported stock CoCo 3 BASIC task-0 environment, at least 512 KiB of RAM, and a normal CMOC code, data, and stack layout below $8000. Opening and closing reserve and release the library's use of the region without clearing its contents. They do not constitute a general physical-memory allocator. [41]

Before a transfer, the implementation validates the complete bank range and the complete caller buffer. Subtraction-based bounds checks avoid accepting a wrapped offset-plus-length expression. Read destinations must be writable data; write sources may include constant program data. A zero-length operation needs no buffer and can name the end of the reserved region. Unsupported mappings and overlapping use of the reserved physical pages are refused before copying. The opening probe also checks that candidate high-memory pages are independent of ordinary low memory, so a configuration in which nominal extended pages alias a smaller RAM installation is not accepted merely because an MMU register can be written. On expanded machines the documented BASIC bank and prefix settings must remain in force: write-only state cannot be recovered reliably after arbitrary application paging. [41]

Transfers are divided at physical page boundaries and into chunks of at most 32 bytes. Each chunk saves the condition-code register and affected MMU mapping, masks IRQ and FIRQ, maps the selected page into the copy window, performs the short copy, restores the mapping, and finally restores the original condition codes. Restoration therefore preserves a caller's existing interrupt mask rather than unconditionally enabling interrupts. The normal mapping is available between chunks. This provides a concrete upper bound on the copy work inside one masked interval, although it is not a complete bound on application interrupt latency. The authored qualification sources examine physical bytes, page-boundary transfers, rejected ranges and configurations, and mapping restoration for both emulated processors. Inspection of these fixtures identifies the intended regression coverage; dated execution is reported separately. [41, 48]

6.6.2. GIME drawing and bounded presentation

The graphics library implements a specific double-buffered GIME mode: 320 × 192 pixels with 16 colors, packed two pixels per byte. Each image uses 30,720 bytes within a 32 KiB region; the two regions occupy physical $40000–$4FFFF. These pages are disjoint from the bank library's asset region. The public interface requires an ordinary IRQ-enabled BASIC text environment, stock MMU task 0 and video-RAM grouping, sufficient RAM, and the program and stack below $8000. Opening rejects unsupported configurations and clears both image pages. It is not a facility for adopting arbitrary pre-existing graphics-register state. [42]

Drawing changes the hidden image. Rectangle clipping uses the remaining width and height, and packed-pixel edge writes retain the adjacent pixel's nibble. The implementation restores MMU and interrupt state around mapped drawing spans and permits IRQ service between rows. A page remains intact when it later becomes hidden, allowing incremental redraws if the caller accounts for its previous contents. Palette changes are immediate and shared by the displayed and hidden images; the term double buffering applies to pixel storage, not to a second independently staged palette. [42]

Presentation waits for a change in BASIC's complete 16-bit field timer while leaving IRQ service enabled. The polling loop has a finite $2000-iteration limit. Once a change is observed, one high-offset register write selects the aligned finished page, and only a successful presentation exchanges the visible and drawing identities. Failure to observe timer progress returns failure without exchanging them. Comparing the timer for inequality also permits its ordinary wraparound. This is a bounded synchronization protocol dependent on the supported BASIC vertical-interrupt service; it is not evidence of universally tear-free output under arbitrary interrupt replacement or delay. Closing restores the saved palette and the supported BASIC text-register image, including when the field wait fails. Because several video registers are write-only, the restoration contract is deliberately limited to known BASIC text states. The public fixtures include page-content and raster observations, clipping and guard-pixel checks, timeout behavior, and restoration checks. [42, 48]

6.6.3. Background Orchestra-90 and shared interrupt resources

Orchestra-90 support separates direct DAC writes, blocking foreground playback, and an asynchronous player. Each stereo frame contains one unsigned byte per DAC. The foreground player leaves the declared guest clock, timers, and interrupt masks unchanged; interrupt handling can delay its approximately 4,000-frame-per-second output. The background player instead borrows a specific set of resources in a stock CoCo 3 BASIC environment: the BASIC IRQ trampoline, an otherwise unused PIA horizontal-interrupt source, the two DACs, and the guest clock setting. It refuses an unsupported trampoline, a busy horizontal-interrupt source, disabled required interrupts, or another active playback session. Since the DACs are write-only and have no reliable device identifier, a successful start establishes that the supported setup was accepted, not that a physical cartridge has been detected. [43]

The background routine returns to foreground code and services one stereo frame per four serviced scanlines, approximately 3,930 frames per second at NTSC timing. It selects the fast guest clock for playback and delays sample output until the second genuine BASIC vertical interrupt, allowing that transition to settle. The normal vertical-interrupt handler continues to run, preserving BASIC timer and keyboard activity. The GIME timer and FIRQ vector are left available. The handler uses resident private state without assuming the foreground direct-page register, but the application must keep the handler, sample array, stack, and their mappings valid until playback ends. Completion of a one-shot, or an explicit stop, releases the borrowed interrupt resources, restores the declared prior clock setting, and rests the DACs at midpoint 128. Cleanup preserves the current unrelated PIA multiplexer selection rather than restoring an obsolete whole-register image. [43]

This resource discipline supports cooperation with foreground work; it does not implement a general interrupt-arbitration system. Long interrupt masks and PIA reads can delay service. The bank library's short masked copies and the graphics library's row-wise mapping restoration reduce particular sources of interference, but they do not establish a zero-jitter composition theorem for all three libraries. A combined workload must be measured with its actual drawing, input, and asset-transfer schedule. The authored background-audio fixtures exercise refusal without disturbing caller state, repeated and one-shot playback, release of resources, preservation of foreground PIA changes, and DAC sequences under both emulated processors. Such tests support the stated protocol, not a claim of exact sample timing on every host or physical machine. [41, 42, 43, 48]

6.6.4. Sampled input and observable state changes

The input adapter exposes one caller-owned state record per joystick port. A sample contains six-bit X and Y coordinates, thresholded directions with an explicit neutral band, and held, newly pressed, and newly released button bits. Initialization samples a baseline: a button already held is reported as held, without inventing a press transition. Subsequent transitions compare successive samples. A complete press and release between two samples is consequently unobservable. The API is a foreground sampling interface, not a lossless host-event queue, and a game's update cadence determines its attainable input responsiveness. [44]

Axis acquisition uses successive approximation against the six-bit DAC and the CoCo analog joystick circuit, building on BControl's multiplexer and button operations. The short PIA transaction saves and restores the sound gate, multiplexer selection, DAC output, keyboard-column output, and original interrupt mask. These restorations prevent polling from persistently changing a foreground sound selection or keyboard-column configuration. They cannot undo all effects of observation: ordinary PIA data-register reads can acknowledge interrupt flags. Thus this guest API has materially different semantics from the debugger's side-effect-free inspection services. When background horizontal-interrupt audio is active, sampling can also delay an audio service opportunity. Foreground block audio permits sampling between blocks, with block size determining an additional latency tradeoff. These limits should remain explicit when the library is used as an emulator integration workload. [43, 44]

6.6.5. A bounded C compatibility subset

The compatibility layer makes selected C idioms available within the guest's memory and Disk BASIC environment. Its headers are opt-in and enumerate the supported operations rather than promising a complete hosted ISO C implementation. Bounded formatting supports character and string conversions, signed and unsigned integer conversions, octal and hexadecimal forms, integer long modifiers, field width, alignment, padding, and string or integer precision. A positive output capacity always leaves a terminated prefix; a zero-capacity call permits a null output pointer and computes the required length. Unsupported formats and lengths exceeding

INT_MAX return failure. Floating-point formatting is outside this subset, and overlapping input and output remain unsupported. The distinction between a full required length and the number of stored bytes makes truncation observable to the caller. [45]

File streams operate on guest Disk BASIC files through the ordinary ROM-backed decbfile interface. They do not open arbitrary host paths. Two foreground streams, each with a 256-byte buffer, support documented 8.3 names, drive suffixes, read/write/append modes, binary selection, and update modes. Complete granule-chain validation rejects malformed bounds or loops before exposing file data. Text reads normalize CR, LF, and CRLF to a C newline, while text writes use the guest's CR convention; binary access retains byte values. The wrapper saves and restores decbfile's active disk-service pointers around synchronous ROM operations instead of treating each call as permission to reinitialize another user's library state. A compatible Disk BASIC ROM remains an external runtime prerequisite; its code is not supplied by this adapter. [45]

Buffering introduces a second completion boundary. Acceptance by a write function does not imply that every later disk operation has succeeded. Callers must inspect flush and close results. A successful flush publishes the current file data and exact length so a newly opened reader can observe the checkpoint while the writer remains open; an already opened reader retains its recorded length. Update streams enforce transitions between reading and writing, and append writes target the end even after a seek. Mounted media and alternative disk APIs must remain stable until streams are closed. These contracts define guest file visibility; they do not independently establish host crash durability or remove the emulator's write-protection, media-saving, and recovery obligations. [45]

Allocation is similarly explicit. The caller supplies a fixed writable guest array of at least 16 bytes whose extent does not wrap the address space. Reinitialization while allocations remain live is rejected. The wrapper uses WydeAlloc and retains its own allocation-size prefix rather than depending on that allocator's private block representation. Zero-size allocation returns null, calloc checks multiplication overflow, and failed realloc growth leaves the original allocation valid. Shrinking retains the block; realloc with zero size releases it. These are useful bounded failure semantics on a small machine, not host-backed virtual memory or protection against freeing arbitrary invalid pointers. [45]

The composition preserves upstream authorship. CMOC is Pierre Sarrazin's compiler; the input adapter uses his BControl library, ROM-backed disk services use decbfile, and allocation builds on WydeAlloc. The manifests distinguish public-domain library components from GPL-covered components, and LWTools retains William Astle's attribution and terms. VCC Plus's original wrappers and hardware libraries do not subsume those licenses or transfer authorship of the underlying work. Compatibility fixtures and guest-running stream, write, input, and heap tests supply inspectable examples of success, refusal, and recovery. Their presence is source evidence of intended coverage; dated qualification records identify which configurations were actually executed. [37, 44, 45, 48]

6.7. Bounded call context for owned paused runs

The call-context extension exposes a deliberately limited diagnostic view for a verified, paused, owned C playtest. `get_call_context` accepts the run and immutable snapshot identifiers; it has no caller-selected address, length, or expression. The new disclosure is exactly 64 bytes beginning at the actual stack register S, accompanied by byte values, big-endian word views, and the physical RAM spans backing the window. Every byte must be available RAM below the protected high-memory boundary. ROM-backed, unavailable, wrapping, or unverified locations produce an unavailable result. [30]

The raw window is explicitly uninterpreted. A stack word may represent a register, local, padding, or an argument. Availability of the raw view therefore does not imply availability of a caller or typed values. Caller interpretation requires matching compiler metadata and verified ordinary CMOC prologue/epilogue bytes. At exact entry, the candidate return address is the two bytes at S; in a mapped body, the supported saved-return slot is U+2 and must remain wholly inside the same window. The candidate must match one unique verified direct call to the current callee in approved caller source. There is no search for plausible return addresses or assumption about indirect, interrupt, tail, or unfamiliar calls. [30]

Integer arguments require additional compiler location evidence, a supported frame, a verified caller, nonoverlapping named storage, and complete containment within the same window. Unsupported types, pointers, aggregates, out-of-window values, and optimized or unknown frames remain unavailable. Even the count of available parameter records does not claim that metadata contains the complete runtime prototype. No pointer

is dereferenced to enlarge the disclosure. The window is not added to the ordinary external memory-read ranges. [30]

The service rechecks connection, project, artifact, worker, boot, processor and execution mode, registers, mapping, program revision, counters, memory, and availability against the selected immutable pause. Execution, a new snapshot, media exchange, source/build replacement, restart, or revocation invalidates the applicable observation. Its external grant is specific to the C owned-playtest capability; older grants, ordinary shared runs, BASIC, and assembly connections do not inherit it. This provides a useful raw diagnostic for optimized code while keeping evidence-dependent ABI interpretation separate from what the bytes alone establish. [30]

6.8. Compiler Correctness and Toolchain Engineering

I treat the compiler as a second boundary for executable behavior within the development platform. A correctly emulated register can still supply an incorrect application result if a compiler transformation changes an effective address or discards a condition code on which generated code depends. VCC Plus therefore maintains targeted CMOC 0.1.100 corrections in addition to its debugging metadata extension. These corrections address general optimizer rules, not recognition of a particular game, variable name, or peripheral API. Their separation from emulator changes is essential to diagnosing the layer responsible for a failure. [34, 35]

6.8.1. Indexed arithmetic and hidden register dependencies

The MC6809 register model creates dependencies that are easy to miss in an assembly peephole transformation. Register B is the low byte of D. An instruction that appears merely to replace a byte value can consequently change the effective address of a later `B,X` or `D,X` operand, including indirect addressing. Similarly, removing a stack push changes the interpretation of subsequent S-relative operands, and removing a predecrement or postincrement operation changes state beyond its loaded value.

The indexed-byte subtraction patch guards three transformation sites in `ASMText2.cpp`. Reordering two loads for a comparison is refused when their addresses depend on B, S, or an index-update side effect. Rewriting subtraction through negation and addition is refused when changing B before a load would change that load's address, or when removing the push changes its stack-relative operand. A further operand-swap rule rejects B-dependent addresses. Safe immediate and independent cases remain eligible for their existing transformations; the correction constrains the conditions under which a rewrite is justified. [34]

The authored regression includes `184 - heights[i]`, runtime word and byte indices, variable pointer bases, conversion-induced byte wrapping, both operand orders, and comparisons against an indexed byte. It prints raw result bytes for comparison with independently calculated host expectations. This structure matters because a guest-side assertion implemented using the same defective subtraction or comparison could mask the failure it was intended to detect. The fixture tests emitted program behavior rather than merely checking whether the optimizer recognizes a particular source spelling. [34]

6.8.2. Indexed function arguments and effective-address evaluation

A second failure arose while a joystick-driven program passed adjacent axis and calibration values to a function. The optimizer removed a load whose value appeared to be overwritten, leaving the prior argument's value to select the next array element. The relevant sequence was:

```
LDB index
LDX #centers
LDB D,X
```

Although the final instruction writes B, it first reads D to compute its effective address. The initial B load is therefore live at that instruction. Looking only at the destination register mistakes an address dependency for a dead value. The original rule also inspected the mnemonic field where it needed to examine an operand for index-update side effects. [35]

The correction uses CMOC's existing instruction-effects analysis to test B dependencies in both the intervening X load and the final B load. This includes D-indexed and indirect forms rather than a single literal `B,` prefix. It separately checks whether removal of the first load would discard a predecrement or postincrement effect. Independent overwritten loads can still be removed. The change repairs a compiler rule without changing the emulator's joystick values or the guest program to accommodate the defect. [34, 35]

The retained qualification record documents a useful before/after discriminator. The earlier staged compiler passed at `-O0` on both guest processors but failed at `-O2`. For one pair of indices, the expected bytes were `04 0C 08 10 14 1C 18 20`; the defective byte-index case instead produced `04 0C 3A 10 14 1C 3A 20`. The deliberately distinct `$3A` sentinel identifies use of an unintended array element. The corrected compiler's documented matrix passed through both host compiler slices, at both optimization levels and on both guest processors. A direct rule fixture separately checks thirteen dependency/side-effect guards and two safe removals. These historical observations are distinct from the fresh paper-specific matrix in Section 9.7. [35]

6.8.3. Logical negation and condition-code liveness

The add-zero correction addresses another implicit output of an instruction: its flags. CMOC can emit `ADDD #0` to establish a word-sized zero test. Numerically, adding zero leaves D unchanged, but removing that instruction can leave a stale Z flag for a following runtime helper. The recorded example returns one in D while Z remains set from clearing A. Removing the add before `tfrZtoB` then makes `!getReady()` produce one, although the function returned a nonzero integer. An identity transformation on the arithmetic value was not an identity transformation on the observable machine state. [34]

The patch removes unproven add-zero deletion and byte-test substitutions from the general peephole rule. More specific transformations for recognized load/compare/branch sequences remain in place. This is a conservative repair to the rule's proof obligation, not a general disablement of optimization. The corresponding C fixture prints results for zero, widened unsigned-byte values, and 16-bit values with only either half or both halves set. Expected results are supplied by the host runner. Together with the indexed cases, it demonstrates that compiler correctness requires accounting for address registers, stack effects, and condition codes in addition to explicit destination values. [34]

6.8.4. Source identity and scoped preprocessing

The dependency-path correction preserves filenames through generated Make prerequisites and preprocessor line markers, including quoted or escaped spaces, dollar signs and hashes. This prevents a valid source file from acquiring a different dependency identity merely because its host pathname requires escaping. [36, 37]

The native mcpp adapter expands ordinary C normally but processes balanced inline-assembly bodies in assembler mode. It substitutes only those bodies, preserving operands such as `#COUNT` and `-COUNT,x` without changing C token boundaries, strings or preprocessing errors elsewhere. Matching block counts and retained line markers are prerequisites; inconsistent transformations fail rather than compiling truncated input. [36]

6.8.5. Tool extensions and executable identity

Two accompanying changes extend other tools rather than CMOC's optimizer. LWASM's project-input and source-map extension bounds assembly-workspace reads and records physical source associations for verification. The `writecocofile` extension exposes a forward disk-allocation policy needed by the unmodified CoCo Letters recipe: allocation starts at granule zero, skips occupied granules and can continue beyond the directory region. Its `--allocate-before-dir` name does not impose a strict below-directory limit. These contracts identify what changed without attributing the assembler, filesystem implementation or guest libraries to the compiler corrections. [38]

Tool identity must also survive an unchanged upstream version string. Feature markers and payload hashes bind a retained artifact to the corrected compiler actually used; a source patch alone cannot establish that a running debugger is interpreting output from that compiler. A recorded source/staged-binary mismatch illustrates why the executable participates in the evidence chain. [37, 39]

The optimization study in Section 9.8 supports retaining `-O2` for ordinary Release projects while making frame-pointer omission an explicit choice. The preserved Debug frame supplies the storage and caller conventions required by the supported source debugger. The measured tradeoff concerns build settings in the corrected compiler, not a speedup attributed to these patches. [40]

7. Reproducible Development and Programmable Testing

7.1. Source-to-artifact identity

`CMOCProjectInputManifest` gives source interpretation a content-based identity spanning approved source, headers, resources, generator prerequisites, support roots and relevant control permissions. The same classification governs navigation, build snapshots and external-tool scope. Generated source can be a real input; declared products and diagnostic outputs must remain outside that identity so a build does not invalidate itself. Modification time alone cannot distinguish a source change from a scheduling change. [31]

Input scope is path-bounded. A discovered dependency cannot grant access to another root, and an approved support folder cannot recursively enlarge that scope through its own settings. Changes to included content or toolchain payload invalidate the affected artifact and owned execution. Replacing a compiler under the same version string therefore cannot silently reuse results from its predecessor. Isolated working copies preserve the user's project when cache verification fails. [29, 31]

The artifact receipt separates process completion from evidence validity. A zero Make exit status can coexist with rejected executable or debugging metadata; a preparation failure can have no process exit code. Retained diagnostics report these outcomes directly, so an external client cannot mistake a successful command for a verified program association. [20]

7.2. Refresh without erasing editor work

Editor text, filesystem contents and a retained executable can represent three different program states. Refresh updates clean inputs while preserving drafts; a conflict with a draft's saved base requires resolution. Changes to input scope or execution permissions require review and identity rechecking before adoption. Refresh alone neither builds nor starts a guest. [31]

Each state can be valid independently while their association is stale. The debugger exposes that condition, and a retry against an outdated artifact cannot quietly create another run from newly observed sources. [20, 31]

7.3. Disposable workers and explicit execution plans

Owned playtests run the installed `vcc-playtest` helper in a separate process. This is a practical response to the retained one-machine-per-process constraint: test execution reuses the engine without inserting another set of globals into the personal emulator process. A worker receives a verified artifact or retained test disk, a private writable copy, and the declared launch plan. It does not inherit the user's personal cartridge topology, tapes, mounted drives, or settings. Test disk retention preserves useful guest-produced state without modifying its verified parent image. [22]

Receipts report the actual processor and observed HD6309 MD register and mode. Selecting HD6309 does not itself enable native mode or change the compiler target; guest instructions establish the mode. Restart retains the selected processor and writable disk. [32]

The worker advances through explicit bounded actions. Keyboard chords and complete joystick states are delivered for a specified number of emulated fields, with actual input/clock receipts. A field count is not a game-update count and can be interrupted by a verified breakpoint. Framebuffer capture is nonadvancing; instruction stepping is one CPU instruction rather than a C source step. Asynchronous startup exposes progress and cancellation without claiming a boot is complete before observed worker state establishes it. [22, 32]

7.4. Evidence capture and retry-safe external control

Graphics, text, debugger state, input observations, and audio have explicit execution effects. A framebuffer capture reads the retained state without consuming guest time. Audio capture deliberately advances a bounded interval, then pauses and returns guest-produced PCM before host gain or playback. The receipt identifies run, project, source revision, artifact, boot, actual field/cycle interval, sample counts, and payload hash. It does not represent a microphone recording or a measurement of perceived sound quality. [33]

Audio capture releases held input and drains earlier samples before the requested interval. Consequently it observes the following interval rather than the sound generated by a preceding input action; an analysis of continuously held movement must respect that contract. Silence likewise requires an appropriate signal measure: a constant DAC level can contain DC while having no audible variation. Nonzero sample count alone is not a universal criterion for erroneous sound. [33]

External clients can lose a response after the worker has already executed an action. Repeating that action naively would produce a second keypress, another boot, or a different clip. Request identifiers and plan matching make supported retries return the retained receipt instead of advancing again. Reusing an identifier for a different plan produces a conflict. A retained audio response remains associated with its original boot even if the worker subsequently restarts. Capability revocation and incompatible identities still prevent stale access; retry safety does not confer indefinite authority. [22, 33]

Resource limits constrain fields, boots, requests, readable ranges, and retained audio bytes. Stop and retention remain available after ordinary execution budgets are exhausted so a client can finish safely. These limits are part of the executable interface contract, not claims that bounded workloads represent all software. They also make it possible to present evidence with explicit totals and truncation rather than accepting unbounded inspection or silently discarding results. [22, 33]

7.5. Observation privileges and evidence interpretation

The Model Context Protocol interface builds on native services rather than bypassing them. A connection advertises its granted tools and capabilities; older grants retain their schema and do not inherit newly introduced access. Build, launch, read-only observation, input, and owned call context are separately constrained. The external debugger exposes verified named stops and snapshot-advertised ranges instead of arbitrary host paths, registers, or writable guest memory. Permission to compile a project is not permission to inspect the user's normal emulator session. [20, 22, 30]

The resulting platform supports an assistant or other client in a reproducible investigation: select an identified build, start an isolated worker, deliver a bounded input plan, stop at a verified location, and inspect evidence associated with that same state. It does not make the client's conclusions automatically correct. An uninterpreted stack window remains raw bytes, a decoded graphics page remains a selected rendering interpretation, and a single successful game route remains a bounded observation. The architectural contribution is that these distinctions are represented in the service contracts rather than left to a client's guess. [19, 22, 30]

8. Correctness Improvements and Integration Cases

I group the resulting behavior changes by the kinds of problems they address. Some corrected defects in shared inherited implementation; others made the new portable device models safer or more complete; a third group corrected mistakes in native integration. These categories matter because a defect found during a port is not necessarily a defect in the original platform's execution path. The following table summarizes the classification used below. The cases were selected for explanatory value and available regression evidence, rather than to estimate a defect rate for the entire inherited codebase.

Case	Classification	Observable consequence	Principal evidence
TST operand sampling	Shared CPU correction	A VSYNC-polling editor could hang despite correct total instruction cycles	Edge-position fixtures and a selected real-cartridge probe
Scheduler construction and HD6309 reset	Shared state-consistency corrections	Undefined initial scaling or stale mode-dependent instruction timing	Scheduler and native-mode reset regressions
MMU and interrupt lifecycle	Shared correctness and robustness corrections	Wrong floating-bus source, retained interrupt inputs, or invalid memory references	Register/lifecycle checks and mapped guest execution
Raster bounds and frame progress	Shared rendering and execution corrections	Framebuffer overflow or an execution call that failed to return after a debugger halt	Guarded mode sweep, sanitizers, and frame-stop tests
FD-502 and hard-disk semantics	Portable adaptation improvements	Incomplete multi-sector behavior, misleading transfer status, or incorrect boundary data	Authored controller tests and selected disk round trips
HD6309 continuation and Multi-Pak signaling	Debugger and bus-state corrections	Stale transfer continuation or cartridge-present state	Targeted stepping and routing fixtures

Case	Classification	Observable consequence	Principal evidence
Analog audio selection	Native integration correction	Idle cartridge output became an audible game-title buzz	Before/after PCM measurements and routing regressions
Transactional image editing	Native filesystem and media integration	A host edit could invalidate guest filesystem state or damage unrelated image regions	Whole-image rollback, region-preservation, and mounted-session fixtures
Mission Recorder	Guest SDK composition and integration validation	Individually usable libraries must also coexist, release resources, and preserve output across failures	Multi-library guest source and selected whole-program flight qualification

8.1. Instruction timing and the Scripsit II polling failure

Color Scripsit II reached its menu but failed to enter its editor in the native application. The relevant guest loop polls the PIA vertical-synchronization flag using `TST`, while an interrupt handler acknowledges that flag. The failure therefore involved the ordering of three events: the flag's assertion, the operand read, and interrupt handling. It could not be diagnosed adequately from a final register value or total instruction cycle count alone. [49]

In the earlier execution path, the timing associated with the instruction was advanced after the operand had been read. Under the resulting schedule, the interrupt path could acknowledge each flag transition before the polling loop observed it. The correction moves addressing and data-cycle advancement before the read, leaving two internal cycles after it. Direct, indexed, and extended `TST` retain their total cycle counts and indexed addressing penalties. The change applies to the MC6809 engine and the HD6309 engine in emulation mode; the HD6309 native-mode path is unchanged. [49, 50]

Motorola's Figure 18 addressing and instruction diagrams provide the original processor reference used by the correction. The archived edition identifies the relevant sheets; the sampling-sequence claim is supported by the recorded implementation review and regression. The archived diagrams were not independently reverified for this paper. [72]

The regression fixture deliberately separates sampling time from total duration. For direct, zero-offset indexed, extended, and indexed-offset forms, it places a changing flag at every cycle position of the instruction. A bus callback checks the scheduler clock when the operand is read. The fixture also checks a single operand read, condition-code results, the retained carry state, and the total cycles. A second authored program recreates the conflict between polling and an IRQ handler that acknowledges the same peripheral flag. These checks are ROM-free and are reported to fail against the earlier implementations. [50]

The selected cartridge probe supplies complementary evidence. It boots the actual editor, observes polling-loop exit, enters text, checks retained memory, returns to the menu through the emulated keyboard matrix, and reenters the editor on both guest processors. The retained input record identifies the tested Scripsit II cartridge edition without redistributing the cartridge. [49]

The corrected sampling point restored editor entry by letting the guest observe the flag at the expected moment.

8.2. Scheduler initialization and processor reset consistency

Two extraction-time defects concerned derived state. The event scheduler's constructor originally computed clock scaling from `mResolution` before that member had been initialized. C++ initializes members in declaration order; initializer-list ordering does not repair a dependency on a later member. The revised constructor uses the defined default resolution when constructing the scaling fields. Full reset also clears requested clock and overclock values, so a subsequent rebase cannot restore an obsolete request. The corresponding scheduler tests exercise initialization, exact-boundary behavior, event ordering, rescheduling, rebasing, and reset. [49, 51]

The HD6309 reset path had a related logical inconsistency. It cleared native-mode bits but retained the cached native instruction timings. The apparent execution mode and the timing table could consequently disagree after reset. The corrected path invokes the existing mode-setting mechanism to restore emulation-mode timing as well as mode state. The native-mode fixture enters native mode, checks its behavior, resets, and verifies restoration; the development record identifies this as a regression that failed before the fix. [49, 50]

Both changes concern state transitions rather than new instruction functionality. They demonstrate why reset must be specified across all representations of a state variable, including cached tables and deferred requests.

8.3. MMU, floating-bus, and interrupt lifecycle corrections

The extracted machine tests exposed several assumptions that had previously been distributed across application state. GIME floating-bus reads consulted the MC6809 state even when the selected engine was the HD6309. The corrected register path obtains the active processor through the host boundary. `CheckGimeRegisters` contains an explicit expectation that the read reflects the selected HD6309 rather than the inactive MC6809. [52, 53]

PIA and GIME reset also require both internal and externally asserted state to be cleared. Clearing a peripheral latch without releasing its asserted CPU input can leave the processor observing a condition that the peripheral no longer records. The revised reset behavior releases those inputs, and tests separately exercise acknowledgement, independently asserted sources, and reset. Inspection tests ensure that a debugger read does not perform the acknowledgement that a normal guest read would perform. [52, 53]

Memory lifecycle changes address host safety. `MmuInit` validates the RAM configuration and prepares replacement RAM and ROM before replacing active allocations. Allocation or ROM-provider failure preserves the current memory. Repeated teardown is safe, debugger page access is bounded, and post-teardown inspection cannot dereference released storage. SAM vector reads consult the current internal-ROM allocation rather than retaining a stale cached pointer after reinitialization. These changes protect the host process; they should not be presented as increased historical-hardware fidelity. [52]

The integration fixture exercises the real MMU and register decoder with both engines. Its authored guest switches tasks and pages, isolates different byte values in physical pages, receives a GIME interrupt through the SAM vector, stacks into mapped RAM, acknowledges the interrupt, and returns with `RTI`. This establishes interaction among selected components, not merely successful invocation of a recording callback. It remains an authored source-contract test rather than a captured Windows differential trace. [52, 53]

8.4. Raster memory safety, event ownership, and bounded progress

AddressSanitizer exposed an inherited framebuffer-overflow path in an oversized register combination. A 128-byte row interpreted at one bit per pixel derives 1,024 pixels. The renderer's unsigned stretch calculation could underflow when that width exceeded the supported surface, after which a border write reached beyond the framebuffer. The revised path leaves unsupported oversized layouts untouched and bounds duplicated border-row writes. [54]

`CheckVideoPixels` includes the precise oversized-layout regression and surrounding guards. `CheckVideoModes` sweeps 2,304 CoCo 3 and compatibility-mode register combinations over their active rows, checking framebuffer guards after rendering. The sweep is useful evidence of exercised bounds, but it is not a claim that every combination is a historically valid mode or that every rendered pixel matches physical hardware. Separate authored expectations check known colors, nibble order, scaling, scanlines, border placement, and a CPU-driven color-bar image. [54, 55]

The extraction also makes GPU and callback ownership explicit. Raster operations use their own renderer rather than accidentally invoking a global Windows renderer. Reset establishes geometry and clears bookkeeping, while stop cancels owned raster and timer events. Weak lifetime guards prevent abandoned callbacks from dereferencing a destroyed renderer. Timer checks cover clock selection, restart and issue delays, overlapping delayed IRQs, zero-count stopping, and scanline countdown. The correction preserves multiple pending delayed IRQs when a timer period is shorter than the issue delay. [54]

`RunFrame` addresses a separate progress defect. It returns on completed frame, debugger halt, exhausted budget, or a CPU that makes no progress. A budget-limited partial frame can resume without restarting the raster. The retained CPU API returns unused cycles, and a final whole instruction can exceed the requested budget; the wrapper preserves that distinction rather than assuming every call consumes exactly its request. The Windows event-based frame loop also uses this bounded operation, replacing a path that could wait indefinitely after a breakpoint. Tests check halt return, limited work, partial-frame resume, and stalled execution. [54, 55, 56]

8.5. Storage transfer semantics and image integrity

The portable FD-502 adaptation retains the inherited WD1793 command model while separating it from Windows handles, physical-floppy access, application preferences, and host filesystem writes. Its explicit

ownership permits focused checks of drive state, buffers, mounted images, and cancellation. The changes include intentional improvements: malformed geometry is rejected before indexing or shifting, invalid sectors cannot alias an adjacent track, drive/side changes abort active transfers, and partial sectors do not commit. Whole write-track streams are validated before any sector changes. [57]

Multi-sector commands now advance until the next sector is unavailable. In the retained Windows implementation used for adaptation, the multi-sector flag was recorded without being used to advance the transfer. A separate completion correction prevents an end timeout from reporting lost data after the complete payload has already been serviced. The tests distinguish this condition from a genuinely unserviced transfer: a completed write commits once without a false lost-data flag, while the unserviced path reports lost data. These are observable controller semantics, not cosmetic changes to a host file dialog. [57, 58]

DMK and VDK support extends the representation layer. DMK retains address-table information, actual sector sizes, deleted-data marks, recorded CRCs, and mixed-density records. Writes update the relevant track data while preserving unrelated tracks. VDK parsing honors the declared header length, including extended headers. The *Shamus* fixture exposed a false address marker in a protected directory track; an authored-byte regression preserves the failure mechanism without requiring the original application disk. [57, 58]

The recorded real-media probe covers 54 distinct DMK images and two VDK images, found in 44 packages. An independently located sector was read from each image on both guest CPUs. For 55 writable images, the probe wrote a pattern to an owned in-memory copy, checked surrounding bytes, ejected and remounted it, reread it, and attempted a protected write. The remaining image rejected the initial write because of its header protection. I also recorded several application startup observations: *Shamus* reached its title screen, *Color FLEX 5.0:4* reached its date prompt, and *Kidwriter* reached its loading screen. [57]

The virtual hard-disk adaptation corrects the Windows EOF loop's stale-last-byte behavior and an inclusive upper-bound error. Reads beyond the current image return zero-filled sectors within the defined maximum capacity, while writes can extend the image with zero-filled gaps. Bounds are checked before access, and DMA into I/O cannot recursively issue device commands. The portable image object tracks dirty-sector revisions so that a completed host save cannot acknowledge a newer guest write accidentally. The host-save protocol remains asynchronous: accepting a guest flush request is not equivalent to durable completion on the host filesystem. [59, 60]

I keep the controller model deterministic and make image updates transactional. Those choices support repeatable reads and preserve media during writes.

8.6. Debugger continuation and cartridge-bus state

The debugger adds execution paths that ordinary running software may not exercise. HD6309 `TFM` is particularly important because one instruction can transfer many bytes and span complete video frames. The stepping implementation clears stale transfer continuation after PC changes, resets, and invalid-instruction traps. Otherwise continuation state associated with an earlier transfer could follow unrelated instructions indefinitely. Targeted regression programs cover native and emulation modes, interrupted and invalid transfers, and transfers spanning multiple frames. Actual raster boundaries continue to govern completed-frame counting. Explicit whole-instruction Step and Continue from a breakpoint retain their documented, different stopping behavior. [61]

The native Multi-Pak bus separately distinguishes the physical startup selector from live ROM and disk-port routing. A software write to `$FF7F` changes the selected routes. The adaptation updates cartridge-present signaling when the ROM route changes, whereas the retained Windows router updated that signal on physical selection but omitted the software-write path. The change prevents a deselected ROM from leaving stale autostart state. Authored bus programs exercise separate ROM and disk slots on both CPUs; a PIA check confirms that cartridge presence follows the selected ROM. This is a documented behavioral difference for which a fresh Windows runtime comparison remained open in the milestone record. [62]

8.7. Native analog audio-routing correction

The audio case belongs to the native integration, not the inherited Windows mixer. A game's title screen produced a loud buzz with Speech/Sound present, although the same configuration was reported quiet on a physical CoCo. Disposable tests without that cartridge had not reproduced the problem. The native sampler mixed cartridge output regardless of the PIA analog selector and sound-enable gate. An idle unsigned cartridge

midpoint of 128 became a DC offset; joystick polling switched the selector and chopped the offset into an audible waveform. [63]

The correction models the selected analog source: CoCo DAC, cassette input, cartridge sound, or grounded input. The sound-enable control gates that path. Cartridge PCM is centered at 128, with zero retained as a valid negative peak. Single-bit sound and the modeled Orchestra-90 external stereo outputs remain independent. Mixing saturates instead of wrapping, and device clocks continue while their output is muted or unselected. The authored routing test exercises these distinctions on both CPUs. [63, 64]

The unchanged-game probe provides unusually clear before/after evidence. With Speech/Sound present, the recorded title-screen interval had AC RMS 0.101581 on MC6809 and 0.101695 on HD6309 before correction; afterward every sample in that idle interval was zero. FD-502-only and FD-502-plus-Orchestra configurations remained digitally silent throughout. Subsequent ordinary game input produced movement effects with a recorded peak of 0.242188, showing that silence was not manufactured by muting the whole application. The retained input record identifies the unchanged game disk used for these observations. [63]

A native output test also recorded continued field advancement, sample consumption, and one audio-engine start during the silent title interval. The physical-machine observation independently supported the original diagnosis. [63]

The original CoCo 3 service manual describes the DAC, cassette, cartridge, and silent analog sources; the shared joystick/DAC role; and sound-enable gating with a separate single-bit path. Its explanatory prose supplies a primary hardware reference for the routing correction, distinct from the before/after game measurements. [73]

8.8. Transactional disk-image editing and mounted-state exclusion

A native filesystem editor introduces a second writer with a different view of the disk. The guest may retain directory, allocation, and open-file state in RAM while the host editor changes the corresponding image. A successful host write alone therefore cannot establish a safe transition back to guest execution. The VCC Plus editor separates three responsibilities: filesystem mutation within a selected region, preservation of the complete backing image, and coordinated replacement of media owned by the machine. This is a native integration contribution; the emulator's ability to mount a representation does not imply that the filesystem editor can safely modify it. [66, 67]

Filesystem operations reuse ToolShed's DECB and OS-9 RBF libraries. VCC Plus supplies the native wrapper, validation, transactions and mounted-machine coordination, including a separate wrapper path for multi-sector OS-9 allocation clusters. The contribution is safe composition and ownership control around existing filesystem routines. [66]

`DiskImageEditorStore` owns a private complete-image working file and a separate extracted working volume. Retained C operations and image I/O execute through this actor outside the main actor. Each mutation first retains the complete image's previous bytes. Successful changes to the extracted volume are merged into the selected region; a failure restores the complete image and reextracts the volume. A batch import or move consequently has a whole-operation rollback boundary even if a lower-level writer has already changed some private bytes. Undo and Redo likewise retain complete-image states, preserving allocation metadata and edits across volume selections. History is bounded, while retaining at least the most recent undo state; these snapshots are an explicit memory cost of the chosen design. [67]

Region selection is deterministic. A recognized OS-9 filesystem begins at byte zero and declares its extent. Numbered HDB-DOS volumes occupy fixed-size Disk BASIC slots at byte zero or after that declared extent. Invalid and unformatted slots retain their positions and bytes; discovery does not scan for arbitrary partition signatures or initialize unknown space. External OS-9 boot extents are validated against the complete image, excluded from overlapping writable slots, and preserved when filesystem bytes are merged back. Supplementary DECB tracks and opaque tails similarly remain outside the allocatable file area. Free-space accounting describes the selected filesystem, not every byte in its backing file. [66, 68]

Mounted editing adds a machine-owned lease. The serialized controller refuses a busy transfer, pauses execution, releases held keyboard state, and captures current guest-visible bytes with a token, machine generation, and media identity. Execution and conflicting media changes remain excluded while that lease exists. Cancel releases the lease with original media intact and leaves the machine paused. Apply requires explicit restart confirmation, current-session validation, compatible size and geometry, writable media, and resolution of unrelated dirty media or captured output. Replacement is prepared before commit; the controller performs

the cold restart and media replacement without an intervening guest instruction. The restart prevents guest code from reusing filesystem state derived from the old image. Native drafts and unrelated media assignments remain outside that replacement. [67, 68]

The final file path retains source-identity and external-change checks. Standalone replacement requires the original bytes still to match; mounted automatic-save paths use their existing coordinated writes, while temporary media retain dirty bytes without silently changing save policy. An edited copy can be exported when replacing the source is prohibited. These are bounded transaction and conflict-detection guarantees. They do not establish crash-consistent commits across guest state and the host filesystem, nor do they promise recovery from every storage-device failure. [67]

The public fixtures expose the relevant invariants directly. `DiskImageBackendTests` checks source-copy isolation, malformed-chain rejection, disk-full rollback, supplementary-byte preservation, descriptor-preserving OS-9 moves, exact-image Undo/Redo, hard-disk region isolation, and external boot preservation. `DiskEditorSafetyTests` checks mounted locks and cancellation on both guest CPUs, unconfirmed-commit refusal, stale sessions, temporary media, aliases, and externally changed sources. Recorded execution on both host slices and selected native import/export and Apply/reopen paths supplement these authored invariants. [68, 71]

Supported editing is deliberately narrower than mounting: validated raw DECB and OS-9 regions, bounded image and file sizes, and specified allocation geometries. DMK/VDK conversion, arbitrary repartitioning, filesystem repair, and precise physical-media behavior are outside this editor. This case extends the architectural argument from execution ownership to data ownership: a reusable machine core needs an equally explicit protocol when native tools modify resources that the guest can also write.

8.9. Mission Recorder as a whole-program SDK integration case

Individually correct libraries can still fail when they share mappings, interrupts, sample storage, and cleanup paths. Mission Recorder tests that composition in an ordinary guest C program: page-flipped GIME graphics, sampled keyboard/joystick input, banked observations, optional Speech/Sound, interrupt-driven Orchestra-90 playback, Disk BASIC files, and serial output. A marked sector is stored through the bank library and read back during saving; subsequent craft movement must not overwrite it. Each display page retains its own drawing state, while background playback requires resident sample data and valid mappings throughout foreground work. [69]

Cleanup follows hardware ownership. Stereo stops before its sample storage or shared resources can be released. Speech timeouts reset the operation and restore the prior analog route. Graphics and bank mappings are closed before ROM-backed file and printer calls, and early failures release resources already acquired. The program preserves an observation across a failed save so retry does not require repeating the flight. Its requested Orchestra-90 configuration is not a hardware-presence probe. [69]

The output path verifies the observation independently of the displayed scene. It chooses an unused temporary/final filename pair, preserves earlier reports and interrupted drafts, writes and closes the draft, reopens it, compares every byte and exact EOF, and checks the second close before renaming. Only then does it transmit the same report bytes through the printer service. This establishes readback and collision-avoidance contracts, not power-loss atomicity or physical delivery to paper. [69]

The harness checks the banked mark, separate ship/marker pixels, speech progress and stereo DAC activity during steering. After cleanup it compares MMU banks, the BASIC interrupt vector and palette with their initial values, evaluates a new BASIC calculation, and requires background DAC writes to stop. A protected-write failure must preserve disk and printer bytes; a later retry must publish the next eligible filename without replacing previous reports, drafts, or the executable. Extracted disk output must equal the captured serial stream. [70]

The retained qualification reports 32 flights: seven FD-502 scenarios at two optimization levels on each guest CPU, plus four Release boots across RGB-DOS and CoCo SDC profiles. The scenarios cover direct/disk launch, silent operation, each audio service separately, requested devices absent, and discard. One marked-sector flight retained 15 at physical address `$50000` and produced the same 27 bytes on disk and serial output: `MISSION RECORDER\rSECTOR 15\r`. A protected-write retry selected `LOG03.TXT` while preserving `LOG01.TXT` and `LOG02.TMP`. These are retained November 2026 observations, not new executions for this paper. [70, 71]

The audio observations record modeled chip and DAC activity. The print check compares serial bytes. The flight validates how the libraries share resources, recover from failed writes, and return control to BASIC.

9. Evaluation Design and Recorded Evidence

9.1. Evidence model and experimental units

I evaluate the design through mechanism-specific regressions, subsystem integration, and selected probes using real programs. The principal unit is a declared claim under a stated configuration, rather than an undifferentiated count of passing tests. An instruction fixture establishes a sampling or state-transition expectation; a complete-machine fixture exercises CPU, MMU, scheduler, and devices together; a native check addresses presentation, asynchronous control, or host output; a selected application probe adds evidence of compatibility for its observed path. None substitutes automatically for the others.

I report results with the corresponding cases and give the fixture, inputs, and configuration in the supporting record. [49, 52, 54, 57, 59, 61, 63, 65, 71]

9.2. Running the core independently

I use authored, ROM-free fixtures to run the extracted core independently of its graphical host. The baseline core and device suite passed, including instruction and reset checks for both emulated processors. For experiments using original software and media, the harness identifies the input bytes and directs writes into owned copies. [77]

9.3. Sustained timing and the value of retained failures

With a 50 ms audio reserve, the earlier 30-minute runs recorded 1,888 missing sample frames for MC6809 and 1,029 for HD6309. The latter already used interactive queue priority. That result pointed me toward the reserve as the next mechanism to test. [65]

I increased the reserve to approximately 100 ms and filled it before output started. A modeled 80 ms producer stall directly exercised the change. The subsequent 30-minute runs recorded zero missing or overflow samples on either guest processor. The maximum sampled audio timing deviation was 7.738 ms for MC6809 and 10.018 ms for HD6309. [65]

The timing harness sampled counters about once per second and counted audio before output gain. That let me compare frame progress and sample consumption during the run. [65]

9.4. Debugging-service integrity criteria

Correct source displays require more than plausible values. My evaluation criteria include unchanged guest cycles and state during inspection; agreement among registers, MMU mapping, and captured memory; invalidation after source, artifact, generation, or mapping changes; refusal of wrong-bank or modified-code source claims; and explicit unavailability at unsupported frame boundaries. Historical trace bytes must remain unchanged after self-modification or later debugger edits. Execution-changing operations, such as source stepping or audio capture, must state their effects and retain their execution identity.

These criteria tie each source association to a supported frame, the verified executable bytes, and current mapping. The result tables report the behavior of the specified inspection and execution operations.

9.5. Observation-integrity experiment

I use an API-level probe to exercise the composed core with an authored 32 KiB replacement ROM and 512 KiB RAM. The guest executes `INC $4000; BRA $8000`, giving the experiment a visible memory change and genuine frame progress without a private system ROM. Each guest is loaded, advanced, paused, inspected, resumed, and destroyed serially. The probe runs the same sequence on MC6809 and on HD6309 in reset emulation mode. [78]

Nine check designs test actual service behavior: initial real-frame progress; explicit I/O availability; repeated nonadvancing observations; logical/physical backing agreement; rejection of invalid ranges; refusal of stale source-step identities; retained memory after an explicit edit; retained frames and memory after real execution; and retained observations after reset. For the repeated-observation check, sixteen iterations interleave logical range reads, whole-physical-RAM reads, a current graphics capture, and an explicit HSCREEN preview. The comparisons include all exposed CPU fields; MMU task and effective-map fields; run state and identities;

cycles and frames; all 65,536 captured logical bytes and their availability flags; all 524,288 physical RAM bytes; breakpoint state; and published frame pixels. [78]

All nine check designs passed on both emulated processors. Six invalid range forms and four independently stale identities were rejected without changing the compared state. The fixture also demonstrated real execution rather than a permanently frozen machine: guest frame counts advanced from 1 to 2 and cycle counts from 14,997 to 29,987 in each reported guest run. Retained results preserved their old bytes and pixels after later edits, execution, and reset. The inspected source matched the reviewed input set; the receipt separately identifies the probe, runner, executable and linked library. [78]

I use this fixture to check that inspection services preserve the captured state they expose.

9.6. Bounded differential comparison with XRoar

An authored common-instruction program was executed in the real VCC core and the installed XRoar 1.12. Both used 512 KiB RAM, the same authored 32 KiB replacement system ROM, and no cartridge or external media. XRoar read an explicit empty configuration and ran with null UI, null audio, and configuration autosave disabled. Every process had a finite wall-time limit. The comparison used MC6809, HD6309 emulation mode, and an HD6309 variant that executed `LDMD #$01` before the common program. [79]

The fixture initializes the common compared registers before entering the observation interval. A sixteen-iteration loop writes bytes 1 through 16 at `$4100`–`$410F`; a second loop reads them through indexed addressing and accumulates their sum. VCC supplies immutable normal execution trace records; XRoar supplies its headless instruction trace. Normalization compares each instruction-entry PC with the post-instruction CC, A, B, DP, X, Y, U, and S fields. VCC additionally reads the final RAM directly. For XRoar, each individual byte read is reconstructed from the accumulator change modulo 256 at `ADDA ,Y+`, with the accessed address obtained from the post-instruction Y value. This is a stated inference from guest execution, not a claim of direct XRoar memory inspection. [79]

Guest configuration	Compared dynamic rows	Equal rows	Verified individual readbacks	Final A / B / CC	Stop PC
MC6809	115	115	16	`$88 / $00 / $54`	`$8027`
HD6309 emulation	115	115	16	`$88 / $00 / $54`	`$8027`
HD6309 after guest `LDMD #$01`	115	115	16	`$88 / $00 / $54`	`$802A`

Every case ended with X and Y equal to `$4110`, U equal to zero, S equal to `$7000`, and DP equal to zero. Both processes exited zero for each case, and the normalized trace hashes matched pairwise. The native-mode VCC observation directly reported MD = 1. XRoar's trace showed the `LDMD` instruction, but this channel did not expose its MD value directly. Initial E/F/V values are unused and excluded; the fixture does not treat differing unspecified reset contents as a semantic failure. [79]

I use the comparison to corroborate the register sequence and memory behavior of the two emulator implementations. [79]

9.7. Target-executed compiler correctness regressions

A further paper-specific experiment compiles the three authored regression programs associated with indexed byte subtraction, indexed function arguments, and logical negation. Fixture source and the guest-execution harness come from the reviewed VCC Plus source set. The staged CMOC 0.1.100 compiler advertises all three correction markers. This corrected development runtime is distinguished from the shipped compiler used in Section 9.8. No stock or deliberately defective compiler is used in this fresh experiment. Historical before-fix evidence remains identified separately. [34, 35, 80]

I compiled the fixtures at both `-O0` and `-O2` and executed the resulting binaries on MC6809 and HD6309. The matrix records **24 guest executions** and identifies the compiler build used for each. [80]

Regression fixture	Expected screen marker	Generated binaries	Passing guest executions
Indexed byte subtraction	`HX AC96806A0001A9F0008000EA42BE`	4	8

Regression fixture	Expected screen marker	Generated binaries	Passing guest executions
Indexed function arguments	`ARGS` , three repetitions of `040C0810141C1820`, then `END`	4	8
Logical negation	`NOT 010 1000 END`	4	8

Every execution terminated with its expected marker observed by the ROM-backed harness. The linked record identifies the compiler payload, fixture-specific options, generated bytes, system-ROM input and guest output. These results concern the corrected compiler; historical before-fix discrimination is reported with the corresponding defects in Section 6.8. [80]

The output markers corroborate the corrected indexed arithmetic, argument loads, and logical negation. The next experiment compares a wider set of mutable output bytes.

9.8. Shipped-compiler optimization study and default selection

The separate CoCoBenchmarks study used the compiler shipped in VCC Plus **3.6.2 build 55** on November 12, 2026. The tested compiler is the universal CMOC 0.1.100 runtime packaged with that application build; the upstream compiler version alone would not distinguish this payload from other integrations. The indexed-subtraction, indexed-argument, add-zero-flags, and debug-metadata markers are all present. No patch was added or compiler rebuilt during that study. Its purpose was to assess generated-project optimization defaults in an already corrected compiler. It is not a stock-CMOC comparison or an estimate of the performance cost of the correctness patches. [40, 81]

9.8.1. Experimental units and correctness gate

Eight authored workloads represent fixed-point arithmetic, sprite-state updates, memory/checksum processing, sorting/state transitions, framebuffer writes, sample-buffer mixing, input scanning, and sector-directory processing. Each operates on synthetic application buffers. The matrix uses seeds 1, 165, and 65521; two repeats; Debug (`-O0 -fno-omit-frame-pointer`), Release (`-O2`), and Release with `-fomit-frame-pointer`; two explicitly selected compiler host slices; and MC6809, HD6309 emulation, and HD6309 native execution. The product is **864 executions**. CMOC emits 6809 instructions in every profile. The harness explicitly enables HD6309 native mode for that separate execution dimension. Normal generated-project relocation and switch defaults are preserved, with a 2048-byte stack allowance. [81]

The study compares every serialized mutable output byte with an independently specified Python reference model. It does not accept a matching checksum as a substitute for matching the covered state. All 864 executions passed. Repeated cycle counts agreed exactly, and equivalent builds from the two compiler host slices had identical guest-binary hashes. For this paper, I rechecked the recorded matrix and reference outputs using the retained evidence. [81]

9.8.2. Cycle and size measurements

The timing interval begins immediately before the workload call and ends after its return. Setup and result serialization lie outside the interval; call and return costs remain inside it. Guest IRQ and FIRQ are masked. An independent three-NOP calibration gives 6 cycles on MC6809 and HD6309 emulation and 3 cycles on HD6309 native mode. The reported values count emulated guest cycles, without baseline subtraction or host elapsed-time conversion. Two deterministic repetitions check repeatability; they are not independent noisy samples from which statistical confidence intervals would be justified. [81]

The following MC6809 cycle counts are arithmetic means across the three seeds, rounded to whole cycles for presentation. Each workload receives equal weight in the cross-workload geometric mean. The byte columns measure initialized DECB payload, excluding container headers and uninitialized BSS. They include linked runtime helpers, startup, setup, and result-collection code, so they do not isolate the workload function's size. Stack and heap reservations are separate from the reported initialized payload. [81]

Workload	Debug cycles	Release cycles	Omit cycles	Release bytes	Omit bytes
Fixed arithmetic	163,697	158,916	158,518	977	960
Sprite state	446,728	410,519	410,467	1,160	1,133

Workload	Debug cycles	Release cycles	Omit cycles	Release bytes	Omit bytes
Memory/checksum	570,717	461,316	459,660	657	630
Sort/state	103,003	83,826	82,446	860	833
Framebuffer	706,813	632,676	632,616	721	699
Sample mixing	191,326	151,049	150,649	669	642
Input scanning	30,119	26,118	25,366	877	845
Sector directory	160,249	131,204	131,140	920	893

Release uses **14.17% fewer MC6809 cycles** than Debug under the equal-workload geometric mean. Relative to existing Release, omission uses **0.69% fewer cycles** on MC6809 and HD6309 emulation and **0.84% fewer cycles** on HD6309 native execution. For a comparison, the calculation takes each workload's ratio of mean cycles, forms the geometric mean of those eight ratios, and subtracts it from one. This weights workloads equally rather than allowing the longest fixture to dominate the result. Each host compiler slice gives the same values. [81]

Summed initialized payload falls from 6,841 to 6,635 bytes across the eight MC6809 programs, a **3.01% reduction**; HD6309 emulation gives the same result. Native-mode variants total 6,865 and 6,659 bytes, a **3.00% reduction**. Per-program savings are 17–32 bytes. Input scanning supplies the largest MC6809 cycle reduction from omission, **2.88%**. These totals describe eight separate executables, including their individual harness and runtime overhead; they are not a predicted saving for one combined application. [81]

9.8.3. Choosing the default optimization profile

The measured Release improvement supports retaining the existing `-O2` default. The much smaller additional cycle improvement from frame-pointer omission does not, by itself, justify making that experimental option universal. Preserved Debug frames support the verified local-variable and caller conventions used by VCC Plus. The recorded engineering decision therefore retains Debug and Release defaults and leaves omission available as an explicit project choice. This keeps the measured performance benefit available while preserving the ordinary debugging convention. [40, 81]

The framebuffer, audio, input, and sector-directory workloads measure CPU-side algorithms. I compare their guest cycle counts and output bytes under the declared build profiles. [81]

The shipped-build study and the corrected development-runtime matrix in Section 9.7 use separately identified compiler payloads; their results do not describe one uniform application build. [81]

10. Discussion

10.1. Architectural extraction as a method of finding assumptions

I answer RQ1 with an architecture that defines explicit operations and ownership for the retained engine and its hosts. Processor code requests memory operations and debug hooks. Register logic requests machine services. Raster code writes to an owned surface through a video contract. A composed session serializes the retained machine state, and the native bridge publishes observations across the language and thread boundary. The approach preserves useful existing algorithms while changing the dependencies through which they interact with their environment.

The architecture lets me test at three scales. A flat memory bus exercises instruction behavior. A recording host shows whether register writes reach a device. The composed machine then exercises the timer, processor, and interrupts together. [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

The work also demonstrates why an interface should not be confused with elimination of all inherited constraints. The retained global CPU state, scheduler, and host bindings still prevent independent machines from sharing one process without additional work. The occupancy claim guards lifecycle exclusion, while serialization governs actual operations. These are different mechanisms. Explicitly documenting both is more useful than describing the engine as reentrant on the basis of its new classes alone.

10.2. Correctness includes the placement of observations

For RQ2, I identify several categories of defect in the case studies. Initialization and memory-safety changes address undefined or unsafe host execution. Mode-reset and interrupt cleanup changes address internal consistency after lifecycle transitions. Device changes repair register and transfer semantics while retaining an intentionally simplified hardware model. Native sound routing corrects a new integration path. These categories should remain separate in any assessment of how much the inherited emulator changed.

The Scripsit case illustrates the distinction between total instruction time and the moment a guest observes hardware. The polling instruction and the interrupt handler competed to read or acknowledge the same synchronization flag. Moving the operand read within the instruction changed whether the loop could finish while preserving the instruction's duration. [49, 50]

The audio case offers a different lesson. Testing a game without the relevant sound cartridge configuration did not reproduce the user's failure. The eventual reproduction preserved the game and its inputs while varying the device topology. The correction followed source selection, gating, sample centering, and mixing semantics rather than suppressing the observed waveform globally. A compatibility fixture is stronger when it retains the relevant configuration and demonstrates both removal of the unintended signal and preservation of intended output. [63, 64]

10.3. Debugging interpretation requires an evidence chain

For RQ3, I separate an observation from the interpretation attached to it. A consistent memory capture can accurately report bytes while offering no justified C frame. An executable symbol can identify a real entry without supplying a source statement. A bank declaration can describe intended placement without proving the guest has installed the module. The source-map, relocation, runtime-guard, and metadata services address different links in this chain. Their checks are complementary rather than interchangeable. [23, 24, 25, 26, 27, 28, 29, 30]

The conservative unwinder and call-context service illustrate the operational consequence. They require supported code and storage relationships before naming callers or arguments. They do not scan for plausible return addresses to produce a longer stack. Unavailability is therefore part of the service's successful behavior when its prerequisites are absent. This principle is particularly important for assistant clients, which can otherwise turn an apparently authoritative numeric response into an unsupported explanation.

Each check has a specific role. A content hash identifies bytes. A mapping fingerprint checks whether the logical and physical views still agree. The executable guards tie source associations to those bytes, and the bounded watch language evaluates supported expressions over captured state.

10.4. Programmable investigation without a second source of machine truth

I address RQ4 through two ownership boundaries. Within an interactive process, one machine owner governs commands and snapshots. For disposable testing, a separate worker process owns its own machine and writable test-media copy. The first boundary prevents native windows from becoming competing controllers of shared state; the second accommodates inherited process-wide globals while separating investigation from a personal session. [5, 22, 31, 32, 33]

Capability discovery, opaque result identities, explicit field budgets, and retry-safe request IDs make external automation reviewable. A repeated request can return its prior receipt rather than deliver another key press or advance another audio interval. A revoked or incompatible project does not retain an apparently valid observation merely because a client remembers its address. These are service-level properties that support robust experimentation, though they do not establish that an external client will interpret every result correctly.

A responsive host is a separate requirement from a deterministic guest sequence. Bounded asynchronous work, immutable export, and nonadvancing capture reduce avoidable interference. They cannot guarantee identical wall-clock latency under every host workload. The paper therefore treats execution clocks, output-device consumption and application responsiveness as different measurements.

The same ownership principle appears at two additional boundaries. The guest SDK temporarily borrows MMU windows, display pages and interrupt service paths; it must restore them before returning to ROM-backed I/O. The disk editor privately owns a proposed filesystem state while the guest is excluded from the mounted image, then requires explicit restart when applying it. Neither a writable buffer nor a paused CPU alone supplies that complete contract. Mission Recorder exercises the first boundary across several services,

while editor fixtures exercise cancellation and byte-preserving recovery at the second. Their evidence is complementary to instruction-level regressions, because correct individual components can still conflict through shared mapping, interrupt or media state. [41, 42, 43, 44, 45, 66, 67, 68, 69, 70, 71]

10.5. Interpreting the evidence

I combine independently specified expectations, before-and-after regressions, real-software probes, and a comparison with another emulator. Each gives me a different way to exercise the behavior that motivated a change.

The XRoar experiment compares registers and memory behavior for a common instruction sequence. Agreement gives me a second implementation against which to check the sequence. [79]

The observer probe compares captured registers, RAM, mappings, identities, and published pixels before and after the inspection calls. I pair that comparison with availability flags so the debugger exposes supported bytes without triggering the hardware effects of an ordinary I/O read.

I use the three compiler fixtures to isolate the corrected transformations. In the shipped-compiler experiment, an independent host reference checks the serialized outputs and repeat runs show whether cycle counts agree. Together these experiments connect the targeted corrections to the choice of a default build profile. [80, 81]

10.6. What the application checks show

I used existing applications to observe specific interactions with the composed machine. The Scripsit editor entry isolated an operand-sampling issue, disk save and reopen exercised the media path, and gameplay exercised the audio correction. Each observation adds a program-level check alongside the smaller fixture. [49, 57, 63, 71]

10.7. Source and evidence availability

VCCPlusOSS supplies the implementation and its build documentation; the supporting-evidence collection supplies the experiments, selected source excerpts and evidence records. Public file references, dated configurations and stated host/guest dimensions identify the material supporting each claim. ROMs and populated collections are separate inputs. The source edition retains executable capabilities with empty collections; reproduction of optional-input experiments still requires their declared inputs. [46, 47]

11. Conclusion

In VCC Plus, I modernized a mature emulator by retaining its implementation, defining explicit host contracts, and grounding the development services in runtime evidence. The work separates processor, memory, register, raster, and selected device behavior from the original Windows application while composing them within a serialized native Mac host. Extraction exposed software-safety and guest-visible defects whose corrections are supported by targeted regressions and selected complete-program observations.

The debugging extensions make a second contribution beyond platform portability. Consistent capture, runtime identity, executable/source verification, physical-bank awareness, compiler-described storage, conservative stack interpretation, interpreter-aware stepping, bounded tracing, graphics observation, and constrained call context establish the conditions under which a runtime explanation is supported. Isolated programmable workers extend those conditions to automated development and investigation. Transactional editing protects filesystem state across native/guest ownership changes; guest SDK contracts protect shared mappings, interrupts and display resources. Mission Recorder supplies a concrete integrated example of those guest contracts and their failure paths.

I used observation-integrity probes, differential comparison, and target-executed compiler regressions to check those mechanisms. The compiler measurements informed a concrete choice: retain ordinary Release optimization and keep frame-pointer omission available as an explicit project setting. In VCC Plus, separating the host and verifying what a debugger can say about runtime state proved to be closely related design decisions.

Acknowledgments and Attribution

I designed VCC Plus with sponsorship from Dezota LLC. I build on the work of the original VCC contributors and the authors of retained processor, device, compiler, assembler, library, and reference material. Their

original copyright and license notices remain applicable. I thank the Color Computer Archive for the historical publications and software provenance used in this paper. The archive retains its own attribution, and each work remains subject to its applicable distribution terms. XRoar is credited to Ciaran Anscomb and its contributors. CMOC and LWTools retain their upstream author and contributor credits. Native hardware artwork is outside the evaluated core and retains its separate notices, including the SockMaster inspiration credit where that artwork is used.

Appendix A. Interface and Service Boundaries

Boundary	Authoritative responsibility	Observation or operation supplied
CPU memory interface	Shared processor/MMU integration	Byte/word access and map selection; explicit isolated test bus
CPU debugger hooks	Owner-installed synchronous hook object	Halt/step eligibility and actual instruction/interrupt observations
Machine host	Composed machine or recording fixture	Required ROM, device, input, interrupt, timer, tape, and printer connections
Video host	Host-owned render surface	Surface access, clearing, active area, artifact policy, and frame completion
Machine session	One loaded machine per process	Serialized lifecycle, execution, safe inspection, media/device composition
Native controller	Application command queue	Asynchronous command ordering and immutable main-queue completion
Source-map verifier	Retained build evidence	Object/listing/link/source consistency and executable guards
Metadata/stack interpreter	Captured state plus supported compiler evidence	Typed storage, verified frame suffix, scoped values, explicit unavailability
Playtest worker	Separate process and owned media copy	Bounded execution/input, captured output, restart and retention receipts

The service catalog in `research/services.md` maps these responsibilities to implementation layers. Public tool schemas expose selected operations; they do not make every internal native operation available to every connection.

Appendix B. Representative Code Mechanisms

The following short listings are selected or line-reformatted for presentation. They are not complete subsystems. Verbatim payloads, original notices, selected line counts, payload hashes, and hashes of the published excerpt files are retained under `snippets/` and in `snippets/manifest.json`.

Listing B1. Explicit termination conditions in the production frame runner. These tests occur in a loop around normal CPU execution; a completed instruction may exceed the remaining budget. [12]

```
if (gpu.FrameFinished())
    return {FrameStop::Complete, cycles};
if (CpuDebugger().IsHalted())
    return {FrameStop::Halted, cycles};
if (cycles >= cycleBudget)
    return {FrameStop::Budget, cycles};
```

The explanatory harness compiles the complete verbatim function against small host stand-ins. It checks completion, halt before and during execution, finite budget, stall, final-instruction overshoot, and missing-CPU rejection. These algorithm checks are separate from the complete-core execution evidence.

Listing B2. Availability is distinct from captured byte storage. The production snapshot service publishes an availability flag and performs a non-destructive read only for safe locations. [16]

```
const bool safe = location < 0xff00 ||
    (location >= 0xff90 && location <= 0xffbf) ||
    location >= 0xfff0;
result.memoryAvailable[i] = safe;
if (safe)
    result.memory[i] = SafeMemRead8(location);
```

Listing B3. A source step is conditional on the paused identity. The call is rejected before execution if a captured generation, program revision, mapping, or PC is stale. Byte and module checks supply additional conditions later in the operation. [16]

```
if (expectedGeneration != generation ||
    expectedProgramRevision != debuggerProgramRevision ||
    expectedMappingFingerprint !=
        mappingFingerprint(GetMMUState()) ||
    expectedPC != impl->CpuState().PC)
    throw std::invalid_argument(
        "The paused program changed; refresh before C stepping");
```

Listing B4. The corrected direct-addressed TST separates sampling from its trailing internal cycles. The surrounding instruction decoder supplies the operand address and flag helpers. [49, 50]

```
Cycles(4);
temp8 = MemRead8(temp16);
cc[Z] = ZTEST(temp8);
cc[N] = NTEST8(temp8);
cc[V] = false;
Cycles(2);
```

Appendix C. Evidence Records

The supporting-evidence collection retains the fixture sources, measured results and source-excerpt provenance linked below. Build and reproduction instructions accompany those artifacts. The reported host and guest configurations delimit the results; generating a reading edition does not repeat an experiment.

Record	Technical claim supported
`research/services-qualification.md`	Disk-editor transactions, guest-library contracts and the Mission Recorder integration
`snippets/manifest.json`	Source selections and original notices for the representative mechanisms
`research/results/core-checks.json`	ROM-free core and device results
`research/results/observer-invariants.json`	Nonadvancing inspection and retained-capture invariants
`research/results/xroar-comparison.json`	Common instruction-state and readback agreement
`research/results/cmoc-regressions.json`	Executed regressions for the corrected compiler transformations
`research/results/cmoc-benchmark-summary.json`	Correctness-gated cycle and code-size comparisons
`research/archive-citations.json`	Historical publication editions and inspected locators

References

Implementation and selected technical records are linked by file path in VCCPlusOSS. Paper-specific methods, excerpts and public evidence records are available as supporting evidence. Named versions, validation dates and stated configurations distinguish the reported results.

Reference 1

`docs/macos/PORTING_PLAN.md`. Product constraints, source baseline, implementation scope, source organization, device decisions, and validation distinctions.

Reference 2

`docs/macos/CPU_CORE.md`. Processor extraction, ownership requirements, register representation, execution budget semantics, and authored validation.

Reference 3

`docs/macos/MACHINE_CORE.md`. MMU/register extraction, host connections, memory lifetime, failure semantics, and machine checks.

Reference 4

`docs/macos/GRAPHICS_CORE.md`. Surface contracts, raster/timer ownership, frame execution, and renderer checks.

Reference 5

`docs/macos/MAC_CONTROLLER.md`. Session lifecycle, singleton ownership, native queue, frame delivery, and historical bridge qualification.

Reference 6

`docs/development/REPOSITORY_LAYOUT.md`. Shared implementation and host ownership boundaries.

Reference 7

`src/core/CpuMemory.h`; `src/core/CpuMemory.cpp`. CPU memory contract and explicit isolated-bus override.

Reference 8

`src/core/CpuHost.h`; `src/core/CpuHost.cpp`. Synchronous debug hooks, Windows forwarding adapter, and non-Windows default.

Reference 9

`src/core/MachineHost.h`; `src/core/MachineHost.cpp`. Required machine connections, host installation, and Windows adapter.

Reference 10

`docs/macos/CARTRIDGES.md`; `src/devices/CartridgeDevice.h`. Device ownership, Multi-Pak routing, reset, and persistence boundaries.

Reference 11

`docs/macos/NATIVE_BOOT.md`. Native ROM boot, Metal submission, input, and optional-input validation.

Reference 12

`src/core/FrameRunner.h`; `src/core/FrameRunner.cpp`. Bounded frame-execution algorithm and result semantics.

Reference 13

`src/core/VideoHost.h`; `src/core/VideoHost.cpp`. Extracted surface and video-host contracts.

Reference 14

`docs/macos/AUDIO_CASSETTE.md`; `src/core/AudioBuffer.h`. Scheduled samples, native output consumption, bounded buffering, and interruption limitations.

Reference 15

`src/core/MachineSession.h`; `src/core/MachineSession.cpp`; `platforms/macos/Bridge/VCCMachineController.mm`. Authoritative machine configuration, ownership, snapshot identities, and asynchronous native integration.

Reference 16

src/core/MachineSession.h; src/core/MachineSession.cpp, especially mappingFingerprint, debuggerSnapshot, debuggerMemoryRange, and debuggerStepSource; src/core/FrameRunner.h; src/core/FrameRunner.cpp; platforms/macos/Bridge/VCCMachineController.mm; docs/macos/MAC_CONTROLLER.md.

Reference 17

docs/development/CMOC_SOURCE_DEBUGGING.md; platforms/macos/App/CMOCVariableDebug.swift; platforms/macos/App/CMOCWatchEvaluator.swift; platforms/macos/App/CMOCCallStack.swift.

Reference 18

docs/macos/DEBUGGER.md; src/core/CpuHost.h; src/core/hd6309.cpp; platforms/macos/App/DebuggerModel.swift.

Reference 19

docs/macos/DEBUGGER_MEMORY_TOOLS.md; src/core/DebugDisassembler.cpp; src/core/MachineSession.cpp, graphics-preview and memory-range services.

Reference 20

docs/development/MCP_DEBUG_EVIDENCE.md; platforms/macos/App/MCPDebuggerInspection.swift; platforms/macos/App/CMOCSourceMap.swift; platforms/macos/App/MCPCMOC.swift.

Reference 21

docs/macos/DEBUGGER_TRACING.md; src/core/MachineSession.h, trace record and snapshot declarations; src/core/MachineSession.cpp, trace capture/export; src/core/CpuHost.h, instruction and interrupt trace hooks.

Reference 22

docs/development/MCP_BANKED_PLAYTEST.md; platforms/macos/App/MCPPlaytest.swift; platforms/macos/App/MCPCMOCGuest.swift.

Reference 23

platforms/macos/App/CMOCSourceMap.swift, especially verifiedSections, relocation evaluation, listing/object byte comparison, and retained provenance.

Reference 24

tools/toolchain/patches/README.md; tools/toolchain/patches/cmoc-debug-metadata.patch; docs/development/CMOC_SOURCE_DEBUGGING.md.

Reference 25

platforms/macos/App/CMOCCallStack.swift, particularly capture, frame validation, and direct-call target decoding.

Reference 26

platforms/macos/App/CMOCBankedDebug.swift; docs/development/MCP_BANKED_PLAYTEST.md, build-evidence manifest contract.

Reference 27

src/core/BasicProgram.h; src/core/BasicProgram.cpp; src/core/MachineSession.cpp, BASIC source-transfer/debug services; docs/development/BASIC_WORKSPACE.md.

Reference 28

docs/development/ASSEMBLY_WORKSPACE.md; platforms/macos/App/AssemblyBuildService.swift; platforms/macos/App/AssemblyDevelopmentModel.swift.

Reference 29

docs/development/CMOC_RUNTIME.md; sdk/cmoc; tools/toolchain/patches/README.md; platforms/macos/App/CMOCBuildService.swift.

Reference 30

VCC Plus project (2026). Bounded call-context implementation and authored checks: `docs/development/MCP_CALL_CONTEXT.md`; `platforms/macos/App/CMOCOwnedCallContext.swift`; `platforms/macos/Tests/CMOCOwnedCallContextTests.swift`; `platforms/macos/Tests/MCPCallContextAccessTests.swift`. The technical record distinguishes the implementation contract and observed qualification. The public source index is retained in `research/evidence.json`.

Reference 31

`docs/development/MCP_PROJECT_RELIABILITY.md`; `platforms/macos/App/CMOCProjectInputs.swift`; `platforms/macos/App/CMOCBuildService.swift`.

Reference 32

`docs/development/MCP_PLAYTEST_PROCESSORS.md`; `docs/development/MCP_BUILD45_VALIDATION.md`.

Reference 33

`docs/development/MCP_AUDIO_CAPTURE.md`; `platforms/macos/App/MCPPlaytest.swift`.

Reference 34

`tools/toolchain/patches/cmoc-indexed-byte-subtraction.patch`; `tools/toolchain/patches/cmoc-indexed-argument-load.patch`; `tools/toolchain/patches/cmoc-add-zero-flags.patch`; `tests/toolchain/indexed-subtraction/qualification.c`; `tests/toolchain/indexed-arguments/qualification.c`; `tests/toolchain/logical-negation.c`; `tools/toolchain/patches/README.md`.

Reference 35

`docs/development/CMOC_INDEXED_ARGUMENT_LOAD.md`; `tests/toolchain/indexed-arguments/check.py`; `tests/toolchain/indexed-arguments/check-rule.py`; `tests/toolchain/indexed-arguments/optimizer-rule.cpp`.

Reference 36

`third_party/manifests/cmoc-make-dependencies.patch`; `tools/toolchain/cpp-mcpp.c`; `tools/toolchain/cmoc-runtime.sh`; `tools/toolchain/verify-runtime.py`; `docs/development/CMOC_RUNTIME.md`.

Reference 37

`third_party/README.md`; `third_party/manifests/cmoc.json`; `tools/toolchain/build.py`; `tools/toolchain/build-runtime.py`. Import patches and later build-copy transformations have distinct provenance.

Reference 38

`tools/toolchain/patches/writecocofile-before-dir.patch`; `tools/toolchain/patches/lwtools-project-inputs.patch`; `tools/toolchain/patches/README.md`.

Reference 39

`docs/development/CMOC_RELEASE_QUALIFICATION.md`; `docs/development/CMOC_RUNTIME.md`. Dated installed-runtime and relocation qualification, including the source/staged-binary mismatch.

Reference 40

`docs/development/CMOC_BENCHMARKS.md`. Shipped-compiler benchmark summary and unchanged Release-default decision; detailed source and numerical evidence reside in the separately identified CoCoBenchmarks artifact.

Reference 41

Banked guest assets and authored regression coverage: `sdk/cmoc/include/vcc/bank.h`; `sdk/cmoc/bank/bank.c`; `tests/toolchain/bank/qualification.c`; `tests/toolchain/bank/run.cpp`. Source inspection identifies reservation, preflight, mapping and condition-code restoration contracts; execution evidence is separately dated.

Reference 42

GIME drawing and presentation: `sdk/cmoc/include/vcc/gime.h`; `sdk/cmoc/gime/gime.c`; `tests/toolchain/gime/qualification.c`; `tests/toolchain/gime/run.cpp`. The supported BASIC entry state, physical-page layout, bounded timer wait, and restoration limits are explicit in the implementation and fixtures.

Reference 43
Orchestra-90 foreground and background protocols: `sdk/cmoc/include/vcc/orch90.h`; `sdk/cmoc/orch90/play.c`; `sdk/cmoc/orch90/background.c`; `tests/toolchain/orch90-background/qualification.c`; `tests/toolchain/orch90-background/run.cpp`.

Reference 44
Sampled guest input and upstream composition: `sdk/cmoc/include/vcc/input.h`; `sdk/cmoc/input/input.c`; `tests/toolchain/input/qualification.c`; `tests/toolchain/input/run.cpp`; `third_party/manifests/bcontrol.json`.

Reference 45
Bounded C compatibility: `sdk/cmoc/compat/include/stdio.h`; `sdk/cmoc/compat/include/stdlib.h`; `sdk/cmoc/compat/format.c`; `sdk/cmoc/compat/streams.c`; `sdk/cmoc/compat/heap.c`; `sdk/cmoc/compat/tests/compat-check.c`; `tests/toolchain/streams/qualification.c`; `tests/toolchain/stream-writes/qualification.c`; `tests/toolchain/stream-writes/update.c`; `tests/toolchain/heap/qualification.c`; `third_party/manifests/decbfile.json`; `third_party/manifests/wydealloc.json`. These are bounded guest facilities with file-specific upstream notices, not a complete hosted C runtime.

Reference 46
Shared content-profile implementation and public build configuration: `README.md`; `tools/build_content.py`; `tools/tests/test_build_content.py`; `tools/catalog/package.py`; `tools/catalog/app-schema.sql`; `tools/knowledge/package_archive.py`; `tools/knowledge/archive-library.sql`; `tools/manuals/lookup_resources.py`; `platforms/macos/Tools/bundle-manual-library.py`; `platforms/macos/Tools/prepare-toolchain.py`; `platforms/macos/Packaging/copy-development-starter.sh`; `platforms/macos/Resources/RuntimeTemplates.json`; `platforms/macos/Resources/BASICReference.json`; `platforms/macos/Resources/AssemblyReference.json`; `platforms/macos/VCC.xcodeproj/project.pbxproj`.

Reference 47
Explicit public dependency preparation: `third_party/README.md`; `tools/dependencies/fetch.py`; `tools/dependencies/tests/test_fetch.py`; `third_party/manifests`; `tools/dependencies/manifests/legacy-graphics.json`; `tools/toolchain/verify-sources.py`; `tools/toolchain/build-runtime.py`. Source inspection establishes the preparation and refusal mechanisms; upstream licenses and notices remain component-specific.

Reference 48
`research/services-qualification.md`. Public technical projection of retained, dated SDK and build-profile qualification, preserving configuration and scope limits. Source inspection and executed qualification remain distinct evidence categories.

Reference 49
`docs/macos/CPU_CORE.md`, CPU extraction record and November 4, 2026 Scripsit II timing correction.

Reference 50
`tests/core/CpuChecks.cpp`, especially `testOperandTiming`, `Check6309Native`, and `CheckEventScheduler`; implementations in `src/core/mc6809.cpp` and `src/core/hd6309.cpp`; optional `tests/macos/scripsit-compatibility-probe.cpp`.

Reference 51
`src/core/Event.cpp`, `Event::Event`, `Reset`, and `Rebase`.

Reference 52
`docs/macos/MACHINE_CORE.md`, machine-host boundary, lifecycle rules, corrections, and initial validation.

Reference 53
`tests/core/MachineChecks.cpp`, `CheckMmuLifecycle`, `CheckGimeRegisters`, `CheckPiaInterrupts`, and `CheckMappedCpuExecution`; implementations in `src/core/tcc1014mmu.cpp`, `src/core/tcc1014registers.cpp`, and `src/core/mc6821.cpp`.

Reference 54

`docs/macos/GRAPHICS_CORE.md`, portable interfaces, mode sweep, corrections, and qualification limits.

Reference 55

`tests/core/VideoChecks.cpp`, `CheckVideoPixels`, `CheckVideoModes`, `CheckRasterTiming`, `CheckGimeTimers`, `CheckVideoLifecycle`, and `CheckFrameStops`.

Reference 56

`src/core/FrameRunner.h` and `src/core/FrameRunner.cpp`, bounded execution contract and implementation.

Reference 57

`docs/macos/FLOPPY_CORE.md`, November 1 controller milestone and November 5 DMK/VDK extension.

Reference 58

`tests/core/FloppyChecks.cpp`, transfer, image, damaged-record, protection, and session fixtures; `src/devices/FD502/Fd502Device.cpp`, `src/devices/FD502/DiskImage.cpp`, and optional `tests/macos/fd502-media-probe.cpp`.

Reference 59

`docs/macos/HARD_DISKS.md`, device protocol, asynchronous saving, validation, and limitations.

Reference 60

`src/devices/HardDisk/HardDiskDevice.h`, `HardDiskImage::read`, `write`, `capture`, and `acknowledge`; `platforms/macos/Tests/HardDiskCoreTests.mm` and `platforms/macos/Tests/HardDiskTests.swift`.

Reference 61

`docs/macos/DEBUGGER.md`, execution semantics and dated debugger-completion validation; `platforms/macos/Tests/DebuggerTFMTests.mm`.

Reference 62

`docs/macos/CARTRIDGES.md`, routing differences, authored guest checks, and November 1 integration evidence; `src/devices/CartridgeDevice.h`.

Reference 63

`docs/macos/AUDIO_ROUTING.md`, November 11 cause analysis and unchanged Quadrapassel game-disk probe, with digital measurements and native output qualification.

Reference 64

`tests/core/AudioCassetteChecks.cpp`, `CheckAudioRouting`; `tests/macos/audio-game-validation.cpp`; native sampling in `src/core/MachineSession.cpp`.

Reference 65

`docs/macos/TIMING_RELIABILITY.md`, November 1–2 timing and sustained-use evidence, failed 50 ms runs, final 100 ms runs, and interruption limits; `tests/core/TimingChecks.cpp` and `tests/macos/run-sustained-check.sh`.

Reference 66

`third_party/manifests/toolshed.json`, selected ToolShed filesystem revision, provenance, exclusions, and unmodified snapshot; `platforms/macos/Bridge/DiskImageEditorBackend.mm`, filesystem validation, bounded volume discovery, region replacement, and cluster-aware wrapper operations.

Reference 67

`platforms/macos/App/DiskImageEditorModel.swift`, `DiskImageEditorStore` mutation/restore and model Undo/Redo; `platforms/macos/App/DiskEditorSession.swift`, host contract and bounded source-identity/compare/write implementation; `platforms/macos/App/MachineModel.swift` and `platforms/macos/Bridge/VCCMachineController.mm`, mounted lease, prepare/commit/cancel, source checks, and restart ordering.

Reference 68

`platforms/macos/Tests/DiskImageBackendTests.swift` and `platforms/macos/Tests/DiskEditorSafetyTests.swift`, authored filesystem preservation, rollback, region, lease, source-conflict, cancellation, and restart fixtures.

Reference 69

`examples/cmoc/MissionRecorder/main.c`, `examples/cmoc/MissionRecorder/flight.c`, `examples/cmoc/MissionRecorder/report.c`, `examples/cmoc/MissionRecorder/mission.h`, `examples/cmoc/MissionRecorder/Makefile`, and `examples/cmoc/MissionRecorder/vccplus-project.json`, foreground composition, cleanup, report readback/publication, and build declarations.

Reference 70

`tests/toolchain/mission-recorder/check.py`, `tests/toolchain/mission-recorder/run.cpp`, and `tests/toolchain/mission-recorder/profiles.cpp`, declared optimization/CPU/scenario matrix, ordinary guest input, graphics/bank/audio observations, protected retry, exact disk/printer comparisons, restoration checks, and alternate-storage launch probes.

Reference 71

`research/services-qualification.md`, curated retained qualification for native disk editing and composed SDK examples, including the separately dated disk-editor checkpoints and Mission Recorder flight matrix; no new runs are claimed by the present sections.

Reference 72

Motorola, Inc. (1983). *MC6809 HMOS 8-Bit Microprocessing Unit*, DS9845-R2, Figure 18, sheets 3–5, physical pages 24–26. Color Computer Archive catalog document 2492. Archived original. Edition identity and inspected text locators: `research/archive-citations.json`.

Reference 73

Tandy Corporation (n.d.). *Color Computer 3 Service Manual: NTSC/PAL Version with 512K Expansion RAM Card*, catalog 26-3334. Radio Shack. Sections 1.4, 5.3, 5.5, and 5.8; physical pages 9–10, 36–37, 39, and 41–43. Color Computer Archive document 1717. Archived original. Edition and inspected locators: `research/archive-citations.json`.

Reference 74

Spectral Associates; electronic reconstruction/revision by Walter K. Zydhek (1999). *Color BASIC Unravelled II*, revised December 26. Interpreter Mechanics, physical pages 9–10 (printed 7–8); Appendix B, physical pages 48–50 (printed B22–B24), listing lines 1931–2107. Color Computer Archive document 1106. Archived edition. The date identifies the electronic revision. Edition and inspected locators: `research/archive-citations.json`.

Reference 75

Spectral Associates; electronic reconstruction/revision by Walter K. Zydhek (1999). *Extended BASIC Unravelled II*, revised December 26. Appendix B, Extended BASIC 1.1, physical page 32 (printed B4), listing lines 0289–0316. Color Computer Archive document 4570. Archived edition. Edition and inspected locators: `research/archive-citations.json`.

Reference 76

Spectral Associates; electronic reconstruction/revision by Walter K. Zydhek (1999). *Disk BASIC Unravelled II*, revised December 26. Appendix B, Disk BASIC 1.1, physical page 49 (printed B14), listing lines 1249–1259. Color Computer Archive document 2413. Archived edition. Edition and inspected locators: `research/archive-citations.json`.

Reference 77

VCC Plus paper experiments (2026). Supporting universal Release core checks and verified input manifest. `research/results/core-checks.json`; `research/results/core-source-inputs.json`. November 12. Source baseline and executable identities are recorded in the receipts.

Reference 78

VCC Plus paper experiments (2026). Observation-integrity probe and reproduced host-slice results. `experiments/observer-invariants/probe.cpp`; `experiments/observer-invariants/run.py`; `research/results/observer-invariants.json`.

Reference 79

VCC Plus paper experiments (2026). Authored VCC/XRoar differential fixture. `experiments/xroar/common-loop.asm`; `experiments/xroar/vcc-fixture.cpp`; `experiments/xroar/run.py`; `research/results/xroar-comparison.json`. Installed XRoar 1.12, Ciaran Anscomb and contributors.

Reference 80

VCC Plus paper experiments (2026). Target-executed CMOC correctness regression matrix. `experiments/cmoc-regressions/run.py`; `research/results/cmoc-regressions.json`. November 12. Three pinned C fixtures, two optimization levels, two explicitly selected compiler host slices, and two guest processors; 24 passing guest executions. Corrected staged compiler and native arm64 emulator harness are identified separately.

Reference 81

Holman, B. K. (2026). *CoCoBenchmarks: shipped CMOC optimization qualification*, November 12. VCC Plus 3.6.2 build 55; eight authored workloads, three build profiles and three guest execution modes. The paper companion retains the study source in `experiments/cmoc-benchmarks/`, the numerical record in `research/results/cmoc-benchmark-recorded.json`, the recalculated summary in `research/results/cmoc-benchmark-summary.json`, and the evidence-validation procedure in `tools/import_cmoc_benchmark.py`. This is prior execution evidence revalidated for the paper, not a fresh repetition of 864 guest runs.