Contents

Reproducing Cartridge Behavior and FPGA Video Extensions 2
Abstract 2
1. Introduction 2
2. Method and boundaries 2
3. Sound/Speech Cartridge controller corrections 3
4. Reproducing the FPGA video extension 4
5. Diagnostics that a person can use 5
6. Results 5
7. Attribution 5
Acknowledgments 6
References 6

Reproducing Cartridge Behavior and FPGA Video Extensions

Sound/Speech Controller Corrections and Optional 256-Color Compatibility 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

Compatibility work on a historical computer becomes more demanding when a modern implementation must reproduce both an original peripheral and a later extension. I describe two related changes to the portable VCC Plus engine: corrections to the Sound/Speech Cartridge controller and optional support for the CoCo3FPGA 256-color video extension. For the cartridge, I trace documented TMS7000 behavior through the original controller firmware, including interrupt delivery, timer reloads, port direction changes, and instruction scheduling. For video, I derive the register and pixel behavior from the FPGA source, including cases a conventional byte-per-pixel implementation would miss. Exhaustive digital checks, deliberate defect controls, a native Windows build, and an executable Color Computer diagnostic establish a portable-core foundation. A later Mac phase adds native configuration and follows the guest raster through preview and recording. I report digital audio measurements and a separate listening assessment from the earlier FPGA work. [1, 2, 3]

The Sound/Speech Cartridge corrections and optional FPGA 256-color support described here will be included in VCC Plus 3.8.0.

1. Introduction

In my earlier VCC Plus architecture paper, I described the separation of a historical Windows application into a portable emulation engine and native host interfaces. That separation creates a useful place to investigate compatibility: a device can be tested through its actual engine implementation without requiring every experiment to begin in a graphical application. It also creates an obligation. A successful engine test does not demonstrate that every native setting, configuration snapshot, or display path carries the resulting behavior correctly. [4]

I began with Sound/Speech Cartridge research for a CoCo 3 FPGA implementation. When I compared that work with VCC Plus, I found controller behavior that needed closer examination. I also wanted to know whether the emulator could support the FPGA project's extended palette and 256-color modes while preserving the ordinary CoCo 3 as its default machine.

These features meet at a concrete resource boundary. The FPGA extension uses address `$FF7E` to commit a palette entry. The cartridge also decodes that address. Making both devices available would give a guest write two conflicting meanings. I made this an explicit machine choice enforced by the engine and kept standard CoCo 3 graphics as the default.

I report the portable-core phase and the subsequent native Mac phase separately. Each phase retains its own source and test identities so readers can trace a finding to the implementation and environment that produced it.

2. Method and boundaries

I use the Texas Instruments documentation for the controller-family contract and the CoCo3FPGA and MiSTer source for the optional video extension. The executed tests establish how VCC Plus behaved for specified inputs. I use all three forms of evidence to judge the changes within the scope of this study. [5, 6, 7]

I retained a pre-change Windows build as the baseline and kept the new work on a separate development branch. The first phase covered the portable engine and Windows build. The second covered native Mac

integration and kept the Mac application's existing development identity, version 3.7.1, build 67. I retained the existing CPU, video, session, reset, and cartridge checks alongside the focused tests. [3]

I used tests at several levels. Controller checks inspect timer and interrupt state. Original firmware checks exercise actual program execution and downstream sound activity. Video contract checks compare C++ with expressions and initialization data extracted from the FPGA source. Finally, an assembly guest program exercises the register interface through the emulator's CPU, memory mapping, raster, keyboard, and Disk BASIC paths. Selected records retain the distinctions between these levels. [1, 8]

3. Sound/Speech Cartridge controller corrections

3.1 Timer reloads must preserve elapsed time

The controller timer contains a prescaler and a reloadable counter. In internal clock mode, I use a period of `8 × (prescaler + 1) × (counter latch + 1)` controller phi cycles. An explicit control write can restart the timer chain. An automatic reload after underflow preserves elapsed time already accumulated by the scheduler. Resetting the scheduler during each automatic reload loses residual time and makes batched execution differ from single-cycle progression. [5]

The correction keeps the counter reload local and leaves elapsed-time ownership with the timer tick path. Tests cover all 32 prescaler settings, selected counter extrema, repeated periods, stop/start behavior, and equivalence between batched and single-cycle advancement. I include interrupt-entry cycles in the timer's elapsed time.

External-event mode is a separate path. Accepted logical rising events on A7 advance its prescaler and counter; internal phi ticks do not. Holding the input high, applying a falling edge, or stopping the timer must not invent events. The pin interface delivers those events from the cartridge host-data latch. I model accepted digital transitions. Qualifying physical pulse widths and edge timing requires separate measurements. [5]

3.2 Interrupt levels, pending pulses, and reset

I model the external interrupt pin and the latched pulse as separate sources of a pending request. An accepted pulse remains pending after the line goes inactive, even while the interrupt is masked. Acknowledgement or a write of one to the clear bit clears the pulse latch. A pin that remains asserted can still request service. When it goes inactive after acknowledgment, that request disappears unless a new pulse has been latched. INT3 captures the timer value on assertion even when its interrupt is disabled. [5]

TI specifies that CPU reset disables interrupts globally and individually. It leaves the flag flip-flops' post-reset state unspecified and advises firmware to clear them explicitly. In VCC Plus, I preserve the externally driven level and retain any unacknowledged pulse across a controller CPU reset. I use this convention for repeatable emulator behavior. A complete cartridge reset also initializes board state. [5, 3]

I also account for interrupt entry within the CPU's execution budget. In the earlier implementation, a pin callback could service an interrupt outside that budget. Deferring service to the CPU execution path lets the scheduler account for the entry cycles. If entry consumes the slice, execution stops there.

During the Mac phase, I looked beyond the short firmware fixture after longer Sound Lab and C-workspace cases lost expected audio. The first branch implementation acknowledged the event and copied an asserted level back into the pulse latch. When the firmware released the pin from within the service routine, the copied flag still caused another service. The host-port trace showed repeated reads of the same command. I verified the revised model with tests for release after acknowledgment and after explicit clearing, as well as masked pulses, the emulator's reset convention, capture, and accounted entry. The previously failing diagnostic cases then passed. [3]

3.3 Direction changes can change output pins

A port contains a retained data latch and a direction register. Changing C or D from input to output can expose a value already in the latch even when no new data write occurs. I publish the retained data masked by the new direction when the direction register changes. Input reads continue to combine externally supplied and driven bits. Exhaustive checks cover all 65,536 data/direction combinations for each of the two ports. I treat the released outputs as a digital approximation of the board's high-impedance pins. [5]

3.4 Execute the clock budget actually consumed

An instruction-oriented CPU generally cannot stop at an arbitrary fractional clock boundary. If the device scheduler repeatedly subtracts the requested budget instead of the cycles actually executed, instruction overshoot disappears from its accounting. Over many audio callbacks the emulated controller can run too far ahead, even when each individual callback looks plausible.

I subtract the cycles the controller actually consumes and carry any signed cycle debt across callbacks and sample-rate changes. A PIC-only reset clears old scheduling debt without resetting unrelated sound engines. In the original-firmware fixture, the first 1,323 audio ticks executed 26,853 phi cycles against a nominal 26,846.58. The small difference is consistent with the tested instruction quantization. The measurement tracks digital controller timing. [1]

I kept the scope on controller execution and its connection to the board. I used tone and speech activity to verify that the firmware reached the synthesis components. Listening and analog fidelity are separate measurements.

4. Reproducing the FPGA video extension

4.1 Preserve the register protocol

I model the 256-color mode as an optional extension of the FPGA implementation. Palette programming builds a retained 12-bit value through the existing palette-register path. A write with bit 7 clear copies the low six bits into both halves. A write with bit 7 set replaces the low half while preserving the upper half. Writing an index to `$FF7E` commits palette register zero into the selected entry of a 256-entry lookup table. [6, 7]

The table's initialization and the distinction between machine reset and FPGA power-up are part of the contract. The source memory image initializes entry `i` to `i << 4`; an ordinary CoCo reset does not recreate the FPGA. I preserve that distinction in the emulator. Palette writes queued for rendering also retain their interpretation at the time they were accepted, rather than being reinterpreted after a later setting change.

I used the actual color conversion from the reference RTL. It interleaves the 12 palette bits and then applies explicit channel-bit overrides. The C++ implementation follows those assignments and replicates the resulting output nibbles into host color channels. All 4,096 stored values are compared with the extracted reference expressions.

4.2 Pixel format includes the unusual cases

A plausible implementation might assume every 256-color mode consumes one byte per pixel at a conventional row pitch. That matches useful modes but misses the reference's low-resolution latch behavior. For the two lowest horizontal resolution selections, adjacent bytes `a` and `b` generate the sequence `a`, `((a & 15) << 4) | (b >> 4)`, `b`, `(b & 15) << 4`. Two fetched bytes thus contribute four output indices. Compatibility requires preserving that nibble rotation rather than replacing it with a more convenient interpretation. [7]

I traced width, fetch pitch, extended horizontal selection, and virtual stride separately. VCC Plus uses the reference geometry for the active 256- and 320-pixel modes. It preserves the default lookup behavior for the disabled high-resolution selections, including the 640-pixel selection. The completed checks compare 17,408 RGB, palette-write, geometry, latch, and initialization cases. Restoring an incorrect RGB map, pitch, nibble rotation, or power-up table makes its corresponding control fail. [1, 8]

4.3 A machine option must be authoritative

I keep the FPGA graphics option off by default. Before accepting it, the session checks all retained cartridge assignments, including inactive multi-cartridge positions, for an SSC conflict. While enabled, cartridge insertion is rejected as well. A failed configuration request preserves the current machine and its cartridge assignments.

Mac Machine Settings carries the option through the settings draft, saved defaults, profiles, and machine snapshots. A profile without the new field decodes as standard graphics. When SSC is assigned, Machine Settings offers Eject and Enable and Cancel. The queued transaction rechecks session and media identity before changing the option and removing the conflicting assignments. It preserves the running CPU and unrelated cartridge devices. While the option is enabled, the interface disables SSC insertion and the core enforces the same constraint. Disabling the option leaves the cartridge slots as they then stand.

Profile preflight checks the complete configuration before pausing or saving the running session. The transaction retains recording and media safeguards and checks source identity again when it reaches the machine

queue. This is important because a valid initial inspection can become stale while an operator responds to a confirmation or a file is read. [3]

I found two integration problems that the initial cases missed. Standard GIME palette writes must still mask values to six bits when the option is off. FPGA palette commits require the full guest byte. I restored the standard mask in register dispatch. The detached preview also needed to recognize the FPGA layout, including the paired low-HRES pixel encoding. For the diagnostic gallery its output now agrees pixel-for-pixel with the live raster used by the capture path. [3]

5. Diagnostics that a person can use

The diagnostic downloads include the exact disks I used, their source releases, and instructions for reproducing each build.

I used two guest diagnostics. Sound Lab provides listening cases and saved reports. In the earlier FPGA phase, I listened to the seven-case showcase on MiSTer and marked all seven cases HEARD. The report identities and integrity were verified on the returned disk. I also ran Sound Lab on a real CoCo and VCC Plus. My seven-case listening result applies to that earlier MiSTer run. [2]

FPGA Colour Lab is a new assembly-only visual diagnostic. A Disk BASIC launcher loads a 160-by-192 gallery containing all 256 lookup indices. Space cycles a grid, ramp, and stripes. A toggles palette animation. Q restores standard text operation and returns to BASIC. The retained master disk is 161,280 bytes. Its build uses Python and LWASM 4.25 without requiring a C compiler.

The executable guest ran in the test harness under both emulated 6809 and 6309 processors. Checks observe 256 distinct grid colors, exercise the controls, and verify return to BASIC. A further path starts with the delivered disk, uses real Disk BASIC and its ASCII BASIC/LOADM/EXEC sequence, and checks the return without dirtying the master disk. This catches integration errors that a palette helper test cannot. The Mac recording fixture also measures distinct gallery colors from the native framebuffer and verifies recording output while exercising the option's recording safeguard. [3]

6. Results

I compared the portable engine against directed timer, interrupt, port direction, cycle-accounting, and A7 tests. The checks detected six intentionally reintroduced controller defects. The original cartridge firmware booted and produced tone and speech activity in response to host commands. The FPGA video checks passed 17,408 comparisons and detected four deliberately incorrect implementations. [1, 8]

On the Mac, Colour Lab loaded through Disk BASIC and displayed all 256 colors on both emulated processors. The detached preview matched the live gallery, and the recording received the same palette. Monitor switching preserved the session. Sound Lab exercised the direct-register, noise, envelope, timeout, panic, and report paths. The C-workspace fixtures also verified immediate and stored audio commands. I used the guest behavior and digital audio measurements to check these paths. [3]

In my own Xcode run, I saw the same Colour Lab gallery as the reference image supplied with the disk. I also played Quadrapassel with the extension enabled. The game uses standard graphics and behaved normally throughout that run. [9]

7. Attribution

The original video source carries Gary Becker's copyright notice. Its import history identifies the source as his CoCo3FPGA release. The MiSTer source and README credit the adaptation to Stan Hodge, Gary Becker, and Alan Steremberg. I use those records to credit the original implementation and the MiSTer port. The available history leaves the original date of each extended mode open. [6, 7, 10]

I describe the new C++ code as a source-informed independent implementation. I used the FPGA source to trace behavior and derive the checks used for comparison. The original notices and non-endorsement terms continue to apply to their source.

Acknowledgments

I thank the original VCC contributors, Gary Becker for CoCo3FPGA, and Stan Hodge, Gary Becker, and Alan Steremberg for the MiSTer adaptation. I developed this work on the portable VCC Plus engine and used Sound Lab to record listening observations across machines. Dezota LLC sponsored the work.

References

Primary implementation links identify source contracts and authored fixtures. The selected evidence files retain the scope of recorded executions and distinguish them from source inspection.

Reference 1

Portable-core execution qualification, November 18, 2026. Selected results and SHA-256 identities of retained research receipts.

Reference 2

Earlier MiSTer Sound Lab listening assessment, November 18, 2026. Observations from the FPGA diagnostic.

Reference 3

Native Mac integration and diagnostic results, November 18, 2026.

Reference 4

Holman, Brian K. (2026). From Windows Application to Portable Emulation Platform: Architecture, Correctness, and Verified Debugging in VCC Plus.

Reference 5

Texas Instruments (1983). TMS7000 Family Data Manual, SPND001A. Sections 2.2, 2.4, 2.5.2, 2.5.4, and 2.6.1–2.6.4.

Reference 6

Becker, Gary. CoCo3FPGA, coco3vid.v. Original video source, retained copyright and extension logic; source snapshot identities are recorded in the study evidence.

Reference 7

CoCo3_MiSTer contributors. rtl/Mister_Video.v. MiSTer video implementation, color assignments and pixel-latch behavior; related top-level and initialization inputs are inventoried in the study evidence.

Reference 8

Deliberate defect controls and primary-source inventory, November 18, 2026.

Reference 9

Holman, Brian K. (2026). Hands-on Mac development review of Colour Lab and Quadrapassel, November 18, 2026.

Reference 10

CoCo3_MiSTer contributors. README.md. Project and port attribution.