Cache, Bus, Interrupt Order and Slowdown on a CPS3

Darksoft probes, board-cost model and slowdown census for Street Fighter III: 3rd Strike

J.R.Dunkley

Board research: XeD

16 October 2026

Abstract

This paper measures how Street Fighter III: 3rd Strike spends its frame on a physical CPS3. Two diagnostic builds of the game ran on one board, fitted with a Darksoft kit and a standard SH-2, and their results were filmed off the screen and recovered bit for bit through checksummed, parity-protected readouts. On the board the CPU never parks, and IRQ10 is raised only when a DMA completes. Of a 16.78 ms frame, the VBlank handler takes 2.6 ms, half of it in the sound driver; the game pass takes 8.7 ms; and the scheduler spins idle for 5.2 ms. The pass costs about four times what the instruction table used by FBNeo and MAME charges for it, and memory accounts for the difference: a work-RAM line fill costs 36 CPU clocks, a flash line 20 and a BIOS line 112 on its 8-bit bus; the cache is the manual's four-way LRU cache; and a write buffer hides about six instructions behind a long store.

The free-running-timer compare runs at priority 15 and delays the DMA interrupt by 65 ticks in the 20–29% of frames where it is left pending across the VBlank timer reset. With an estimator calibrated on two emulator builds of known frame length, the same statistics give $419{,}512 \pm 40$ CPU clocks per display frame, a 25.005 ± 0.004 MHz CPU and a 59.604 ± 0.011 Hz refresh. Three physical behaviours made modified programs fail on the board and never in an emulator: a read beyond the 512 KiB work RAM froze the board with every interrupt blocked; the score-screen RAM is 8 bits wide; and a cache purge is ordered behind a pending cache-through store. A repair built on these findings was confirmed on the board.

A timing model built from these measurements, its game pass still 1.8% short, reproduces the board's attract demo frame for frame over 685 paired frames, including the single overrun, and from power-on puts its first overrun on the same frame as the board. Over 129,310 frames of recorded and generated play it predicts one lost frame in 90,183 frames of live fighting, and five, in three stalls, at every round end in the recorded matches. On the board, slowdown is decided by the memory system and by the transitions that load graphics. A closing section sets MAME, FBNeo, Fightcade and two FPGA cores against the behaviours a board-exact CPS3 has to reproduce, and the companion data lets any implementation check its CPU model against the board's loops.

Contents

1 Introduction **4**

1.1 What was measured, and on what 4

1.2 Evidence classes 5

2 The machine as the game uses it **5**

2.1 Parts and clocks 5

2.2 The address map, with measured costs . . . 6
2.3 The SH-2's own machinery . . . 6
2.4 Two kinds of CPU . . . 8
3 How the game spends a frame on the board 8
3.1 The operating system, briefly . . . 8
3.2 The VBlank handler . . . 9
3.3 Inputs are edges, and edges need a pass . . . 9
3.4 Where the frame goes . . . 10
4 Two probes and how their results came back 11
4.1 Constraints . . . 11
4.2 Probe P1: forty loops and seven hundred frames . . . 11
4.3 Probe P2: a laboratory and five windows . . . 12
4.4 Recovering the data . . . 12
5 What the CPU pays 12
5.1 Reading a loop . . . 12
5.2 The device costs . . . 13
5.3 Line fills, and an 8-bit BIOS . . . 13
5.4 The cache: four ways, sharp thresholds, 36 clocks per conflict . . . 14
5.5 The pipeline hides work . . . 16
5.6 One board, two sessions . . . 17
6 Interrupt order and the frame clock 17
6.1 Two bands of IRQ10 service . . . 17
6.2 Where the fourth compare lands . . . 17
6.3 How many CPU clocks in a frame . . . 18
6.4 The absolute clock . . . 20
7 Graphics DMA 20
7.1 IRQ10 means a DMA finished, and nothing else . . . 20
7.2 The DMA cost the CPU sees is CPU work . . . 21
7.3 How fast the transfer is . . . 21
7.4 Two smaller facts . . . 22
8 Why programs fail on the board and not in an emulator 22
8.1 The score-screen RAM is eight bits wide . . . 22
8.2 Hardware read through the cache . . . 23
8.3 The freeze: a read beyond work RAM . . . 23
8.4 Cache coherency: the purge waits for the store . . . 24
8.5 Silent ends . . . 25
9 A board-cost model, and how close it gets 25
9.1 What it implements . . . 25
9.2 What was fitted . . . 25
9.3 What was not fitted: the attract demo . . . 25
9.4 From power-on: the board's own counters . . . 28

10 Where the board drops frames **29**
10.1 The census . . . 29
10.2 Live play: one frame in 90,183 . . . 30
10.3 Transitions: five frames at every round end . . . 30
10.4 The round wipe, on the board . . . 31
10.5 What a player sees . . . 31
11 The display pipeline **32**
12 Earlier conclusions, revisited **32**
13 What a board-exact CPS3 has to reproduce **33**
13.1 Where other implementations stand . . . 34
14 The next measurements **36**
15 Limitations **37**
16 Conclusion **37**
A Identity **38**
B Companion data **39**
C Addresses used in this paper **40**
D The forty P1 loops **41**
E P2 loop groups **42**

1 Introduction

Previous work established how the game spends a frame in the emulator most players use, and why that emulator slows down where it does [1]. The game's operating system runs eight coroutine tasks from a round-robin scheduler. The VBlank interrupt re-arms the sleeping tasks, and a task that is still running at VBlank is simply not re-armed, so an overrun costs exactly one frame and nothing is caught up. Fightcade's build installs an idle shortcut on the scheduler's read of task 0's state. That read is the first instruction of every scheduler pass, so the shortcut fires immediately after the VBlank handler has made the game runnable. The CPU then stays suspended until the emulator's periodic IRQ10, and the game's work starts with three, two or one slices of the frame left, in a rotation of period three. Which frames slow down is therefore decided by the rotation. That work closed by naming the measurement that was missing: a board.

This paper is about the board. Five questions organise it:

1. What does the board's CPU actually pay for the instructions and memory accesses the game makes, and how far is that from the one-cycle-per-instruction table that FBNeo and MAME charge?
2. In what order does the board serve its interrupts, and what is the exact relation between the CPU clock and the display?
3. Which physical behaviours make a program fail on the board while it runs correctly in every emulator?
4. Where does the board drop frames in real play, and how does that compare with the emulators?
5. What must a timing model contain to reproduce all of this, and how close does ours get?

Three targets are worth keeping apart: compatibility with a particular emulator, arcade behaviour, and player-facing latency [1]. This paper is about the second. Section 11 touches the third only to place the board's display pipeline correctly.

1.1 What was measured, and on what

All physical results come from one CPS3 main board belonging to XeD, who ran every board experiment in this paper. It is fitted with a Darksoft kit: the kit's BIOS programs the flash SIMMs from an SD card and keeps the region setting in work RAM at `0207F000`. The security cart carries a **standard Hitachi SH7604-family CPU**, not Capcom's custom encrypting CPU (Section 2.4). The board's own boot screen reads `990512 / USA / NO CD`. The first probe read the CPU's bus and cache configuration back from the chip: CCR = `01`, BCR1 = `0020`, BCR2 = `00E8`, WCR = `AA57`, MCR = `0000`. The game program is `sfiii3nr1`, Japan 990512 NO CD rev1; its SHA-256 is in Appendix A.

Three kinds of program ran on that board between 5 and 15 October 2026:

- **Four freeze diagnostics** (D1–D4) built on a modified program that froze when a team-assist was called. They printed checksums of both program SIMMs, interrupted program counters, interrupt breadcrumbs and bounded versions of the game's own hardware waits. Together with static analysis of the program they located the freeze and the score-screen width problem of Section 8. A repair built on that analysis was then confirmed on the board.
- **Probe P1** (build `09`). Forty CPU and memory loops run at boot, a 700-frame ring of per-frame timestamps over the first attract fight, and running counters, all shown on screen in checksummed rows. It came back twice: eight photographs (the first return), then a video of every benchmark and 33 log pages (the second return).

- **Probe P2** (build `50320002`). It records 80 per-frame records and 30 task events in five windows of the attract fight, then takes the machine over permanently for a laboratory of 226 experiments. The results are shown as 63 pages of Reed–Solomon-protected rows. One video of one boot came back.

Every probe image is the original program plus a private block in a 64 KiB range of the program flash that the original leaves at zero, a handful of hook words, and private work RAM above the game's own. Section 4 describes them, what they cost, and how their results were recovered.

1.2 Evidence classes

Claims are tagged by the kind of evidence that supports them, and the tag travels with the number.

Tag	Meaning
BOARD	A value recovered from the physical board's own screen, with its decoding provenance (Section 4.4).
ROM	Instructions, addresses and ordering read from the original program or BIOS.
MANUAL	The Hitachi SH7604 hardware manual [2].
SRC	A pinned source: FBNeo, MAME's driver and board notes [3], the JTCPS3 HDL and schematics [4]. Never promoted to board truth on its own.
MODEL	A run of our board-cost emulator (Section 9), whose identity is pinned by hash.
INFER	A conclusion from stated premises, with the alternatives it excludes.

Table 1: Evidence vocabulary.

Units follow the hardware. The CPU clock is the base unit. The game's free-running timer (FRT) counts at clock/8, so one *tick* is eight CPU clocks, 0.32 μs at 25 MHz. A display frame is about 52,433 ticks. Section 6.3 measures both clocks rather than assuming them.

2 The machine as the game uses it

This section is the reference the rest of the paper leans on. Wherever a value was measured on the board, the measured value is given and the source of the nominal one is named beside it.

2.1 Parts and clocks

The CPS3 is a main board with custom chips, a plug-in security cart that holds the CPU, and 72-pin flash SIMMs that the BIOS fills from a CD (or, with a Darksoft kit, from an SD card). Table 2 lists the parts that matter for timing.

Two facts in this table drive results later in the paper. **The CPU and the display run from different crystals**: the CPU's on the cart, the display's on the main board. The number of CPU clocks in a frame is therefore a ratio of two independent oscillators. Its nominal value is 25,000,000/59.5995 = 419,467. Every cabinet has its own value within the crystals' tolerance, and Section 6.3 measures this one. **The score-screen RAM is a single byte-wide SRAM hanging off the sound chip**, so the CPU's 32-bit view of it has only its odd bytes populated. That is why the BIOS memory test masks its readback with `00FF00FF`, and why code that compares whole longwords read from it fails on a board (Section 8.1).

Part	Device	Role and timing facts
CPU	Capcom DL-3229 (an SH-2 with fetch decryption; decapped as a Hitachi HD6417099 variant SRC); on the measured board a standard SH7604-family part	6.25 MHz crystal on the cart, ×4 to 25 MHz SRC; measured 25.005 ± 0.004 MHz BOARD (Section 6.3)
Video master	42.954545 MHz crystal on the main board	pixel clock /5 (384 wide); the game programs a horizontal total of 454 and a vertical total of 262 into the CRTC from a table at `0613E380` ROM, read as 455 counts of XTAL/6 per line and 264 lines SRC: 59.5995 Hz nominal; measured 59.604 ± 0.011 Hz BOARD
Work RAM	HM514260 256k×16 DRAM, 70 ns, behind the CCU (DL-2829)	512 KiB at `02000000`, CS1, 16-bit SRC ROM; 36-clock line fill BOARD
Program	two SIMMs of 4 × 29F016A flash	16 MiB at `06000000`, CS3, 32-bit; 20-clock line fill BOARD
BIOS	29F400TA, 512k×8 flash on the cart	512 KiB at `00000000`, CS0, 8-bit SRC BOARD
Sprite RAM	HM514260 256k×16 DRAM	`04000000`; 19.5 clocks per long write BOARD
Colour RAM	KM681002 128k×8 SRAM	`04080000`; 7.4 clocks per word write BOARD
Character RAM	4 × TC5118160 1M×16 DRAM (8 MiB)	CPU window `04100000`, bank-switched; 20 clocks per word BOARD; filled by the decompressing DMA
Score-screen RAM	62256 32k×8 SRAM on the SSU	`05040000`; **8 bits wide**: only odd bytes exist ROM SRC; 17–19 clocks per word BOARD
Sound	DL-3329 SSU, also the score-screen layer generator	registers at `040E0000`; samples streamed from the graphics SIMMs SRC
DMA	DL-3429 GLL1 bus/DMA controller	character DMA (decompressing), palette DMA, sprite-list DMA; completion raises IRQ10 SRC BOARD

Table 2: Timing-relevant parts. Chip identities from the pinned MAME board notes [3]; CS area and width from the BIOS's BCR writes and the CPU's mode pins; measured costs from P1 (Section 5).

2.2 The address map, with measured costs

The game sees the SH-2's four external areas as in Table 3. The costs are effective CPU clocks per access measured in P1's loops, under the board's own bus configuration. They are throughput figures from whole loops (Section 5 explains what each one includes). For every one of these accesses, FBNeo and MAME charge the instruction's single clock, whatever it touches.

2.3 The SH-2's own machinery

Every address appears four times to the SH-2 (MANUAL sections 7–8). The top three address bits select a partition. `0xxxxxxx` is cached, `2xxxxxxx` is the same memory cache-through, `4xxxxxxx` is the associative-purge window (a write invalidates the matching line, two clocks per line), `6xxxxxxx` is the cache address array, `Cxxxxxxx` is the cache data array, and `E`/`Fxxxxxxx` are the on-chip peripherals. The game reads hardware through the cache-through aliases: `25000000` for the inputs, `240C000C` for the DMA status. That matters on a board and nowhere else (Section 8).

Cache. 4 KiB, mixed instruction and data, 64 sets of four 16-byte lines, selected by address bits 9–4, with LRU replacement kept as six order bits per set. Writes go through, and a write miss does not allocate. In two-way mode (CCR bit 3) ways 0–1 become 2 KiB of on-chip RAM and

Range	Device	Area, width	Board, CPU clocks per access
00000000–0007FFFF	BIOS	CS0, 8-bit	29 / long read; 112 / line
02000000–0207FFFF	work RAM	CS1, 16-bit	6.2 / word, 10.6 / long read; 36 / line 3.7 / word, 8.0 / long write
04000000–0407FFFF	sprite RAM	CS2, 16-bit	19.5 / long write; 21.6 / long read
04080000–040BFFFF	colour RAM	CS2	7.4 / word, 15.3 / long write
040C0000–040C00FF	video and DMA registers	CS2	DMA status at 040C000C
040E0000–040E02FF	sound registers	CS2	
04100000–041FFFFF	character RAM window	CS2	39.5 / long write
05000000–05000007	input ports	CS2	5.0 / word read
05040000–0504FFFF	score-screen RAM	CS2, 8-bit cells	17.2 / word, 34.9 / long write; 19.4 / word read
05100000, 05110000	IRQ12, IRQ10 acknowledge	CS2	
06000000–06FFFFFF	program flash (SIMM 1, 2)	CS3, 32-bit	6.0 / long read; 20 / line
FFFFFE10–FFFFFE19	FRT registers (on chip)	—	12.0 / byte read

Table 3: The CPS3 address map as 3rd Strike uses it. Ranges from the pinned MAME map [3] and the program; costs measured with the cache on, through the cache-through alias where the access is not a cached hit (Section 5). A cached hit costs one clock at any address.

only ways 2–3 cache. The BIOS enables the cache in four-way mode (CCR 10 then 11). The game's bus set-up routine at 0613110E repeats the sequence: CCR = 10, then BCR1 = 0020 and WCR = AA57 under the A55A write key, and the routine at 06131120 writes CCR = 11 ROM. P1 read CCR = 01 back from the running game BOARD.

Bus state controller. BCR2 = 00E8 makes CS1 and CS2 16-bit and CS3 32-bit. CS0's width is set by mode pins, and Section 5 shows it is 8-bit on this board. WCR = AA57 gives two idle cycles after a read in every area, one wait state with external WAIT sampling on CS1–CS3, and long wait on CS0; BCR1 = 0020 sets that long wait to five states (A0LW = 10, MANUAL 7.2.1) ROM BOARD. The WAIT pin matters: a device that never releases it holds the CPU in the bus cycle indefinitely, with interrupts unserviceable. A write buffer lets the CPU continue after issuing a store while the store completes on the bus; later accesses wait for it (MANUAL 7.11).

Free-running timer. FRC is a 16-bit counter at clock/8. The game sets OCRA = 3333 (13,107) with clear-on-compare-A. The compare flag rises while FRC equals OCRA and the counter clears one timer clock later, so **the period is OCRA + 1 = 13,108 ticks** (MANUAL Figure 11.7). Four periods are 52,432 ticks, within a few ticks of a display frame (52,433 nominal). Capcom chose 3333 so that the fourth compare falls just before the next VBlank; Section 6 shows what that costs. The game gives the FRT interrupt priority 15 at 06000EFE–06000F0A, above the external IRQ10 (DMA) and IRQ12 (VBlank) ROM. FRC is reached through a shared TEMP latch, high byte first. An FRT register access costs 3 to 12 clocks depending on phase (MANUAL 11.7), and the board measured 12.

Interrupt acceptance. The instruction immediately after an LDC/LDS/STC/STS-class instruction cannot accept an interrupt (MANUAL 4.6). The scheduler's resume path is LDC R0,SR in a delay slot, then LDS.L @R15+,PR at 06000DC2, so the first instruction able to take a pending VBlank is 06000DC6. That is why diagnostics that sample the interrupted PC keep finding 06000DC6

(Section 8.3).

2.4 Two kinds of CPU

Capcom's cart CPU decrypts instruction fetches with a key held in battery-backed RAM. When the battery dies the key becomes zero (the "dead" or "suicided" cart, also the 2nd Impact configuration): the CPU then decrypts *only instruction fetches* from `06xxxxxx` (and its cache), with an address-derived mask, and data reads pass through unchanged SRC [3, 4, 5]. Darksoft supports two kinds of cart. On a standard cart the custom CPU has been replaced by a stock SH7604-family part, which runs plain code; that is the board measured here. XeD's practical test tells them apart: a cart that runs 4th Strike, is a reproduction, or has no A/B/C/D sticker on its large square chip is standard. A custom (dead) cart needs a program whose instruction words are encrypted and whose data words are not, which requires an exact instruction/data map of the whole 16 MiB image.

The measurements in this paper are of a stock SH7604-family CPU on a CPS3 main board. The bus, memories, DMA and video are the CPS3's own. The CPU core, cache and on-chip timer belong to Hitachi's standard part, and the configuration is the one the BIOS and game program into it. Every CPU-side feature we measured (cache geometry and replacement, CCR modes, FRT period and priority, write buffering) behaves as the SH7604 manual describes. Whether Capcom's custom part adds fetch latency for its decryption is not measured. A cart-to-cart comparison with the same probe would settle it.

The same fact cuts the other way: the CPU-side measurements are not CPS3 facts alone. The Sega Saturn and the 32X are built on SH-2 parts of the same family, and the RTL model the FPGA cores share (Section 13.1) is a general SH7604. Everything in Section 5 that stays inside the CPU transfers to those systems as measured: the cache geometry and replacement, the write buffer's overlap, the multiply states, the load-use interlock, the 12-clock timer-register access, and the purge ordering of Section 8.4. The bus costs do not, because they belong to the CPS3's memories and to the wait states its BIOS programs.

3 How the game spends a frame on the board

3.1 The operating system, briefly

The scheduler is described in full in [1]. Here is what the rest of this paper needs. Eight 64-byte task contexts start at `0200D794`, and byte 0 of each is its state. The scheduler at `06000860` scans the slots in order, with interrupts masked around each slot. It dispatches state-2 (re-armed) and state-3/4 (running) tasks as coroutines through `06000CDC`. A task returns through `06000D28` with a countdown: zero makes it state 5 (finished), non-zero makes it state 1 (sleeping). The re-arm routine `0600094C`, called from the VBlank handler, decrements each sleeping task's countdown and makes it state 2 at zero. It leaves a running task alone. **So the slowdown rule is: a task still running at VBlank misses that re-arm, finishes late, and next runs after the following VBlank.** One logic frame is lost, the display repeats a picture, and nothing is caught up ROM.

In play, slots 2, 3 and 4 run once per frame. Slot 3 (context `0200D854`) is the game: characters, effects, collision, the round. Slot 5 (`06004E12`) is an always-runnable request dispatcher that yields straight back when it has nothing to start. When the frame's work is done, the CPU therefore loops through the scheduler and slot 5 until the next VBlank. **On the board that loop is real execution**: the CPU spins, fetches, reads the task states and swaps DMA queues (`06137C14` is called on every scan from `06000942`). Nothing suspends it.

3.2 The VBlank handler

The VBlank handler (IRQ12, vector 06000528) runs with interrupts masked from 06000576 until it returns at 0600071A. In order, it does the following ROM:

1. Zeroes the FRT counter and the software tick counter (0600058E–06000596). It does *not* clear a pending compare flag (Section 6).
2. Kicks the character-DMA queue (0613809A). The start routine 06138168 copies three longwords per queued command through the character-RAM window.
3. Writes the eight layers' positions to the video registers (0612DDCC) and triggers the sprite-list DMA (0613696C, four 8/9 toggles of 240C0082).
4. Reads the inputs (06132FD8) and runs the coin and service logic.
5. Re-arms the tasks (0600094C, directly or through the frame mask).
6. Waits for the sprite-list DMA (06136996, bit 0 of 240C000C).
7. Calls the sound driver (060006F8), then acknowledges IRQ12 at 05100000 and returns.

The game takes two other interrupts. **IRQ10** (vector 060004D0) masks to level 15, spins at 06138100 while 240C000C & 6 shows a character or palette DMA still running, then starts the next queued character batch or palette request, and acknowledges at 05110000. **FRT compare A** (vector 0600073E, priority 15) increments the tick counter at 02000000 and, at tick 2, debounces the coin, service and start switches. Nothing in that handler touches play.

3.3 Inputs are edges, and edges need a pass

The VBlank input reader 06132FD8 (and its alternate 06132E60) keeps exactly one previous sample per player. It copies the current word to the previous word, then assembles the new current word from the ports at 25000000 ROM. What a lost frame costs an input depends on who derives "newly pressed" from that pair, and the program has two kinds of consumer ROM:

- **The system screens read the reader's pair directly**, as current and not previous: the start button in the entry task (06001ADE), the break-in check (060044AC), character and super-art selection, name entry, the continue screen and the scene-skip helper (06091BEA). For these a press whose first sampled frame falls on a lost frame is never a rising edge, unless the player presses again.
- **The fight keeps its own history.** Each pass copies the player's current word into the command work (061195E0 takes the current word only), and the command checker (060B55F8, called from 060B59BE) moves the last pass's word into its old slot, stores the new one and derives the edge as new and not old. Nothing in the attack path reads the reader's previous word. The history is per pass, not per VBlank, so a press that is held across a lost frame is seen at the next pass and recognised then, one frame late.

What a lost frame does take from the fight is the frame itself. An input that exists only during the lost frame, a one-frame tap or a one-frame direction inside a motion, is never sampled by any pass, and every held input is acted on one frame later than it was pressed. Directions held across the gap survive as holds. This applies only where the board actually overruns, and Section 10 shows where that is.

3.4 Where the frame goes

P1 recorded, for 685 frames of the first attract fight (Ibuki against Chun-Li, from power-on, no inputs), where each part of the frame begins and ends. All values are in timer ticks from the handler's timer reset, which comes about 450 ticks after VBlank onset (the native and probe prefix). Figure 1 and Table 4 set them beside the same probe image run on Fightcade's schedule.

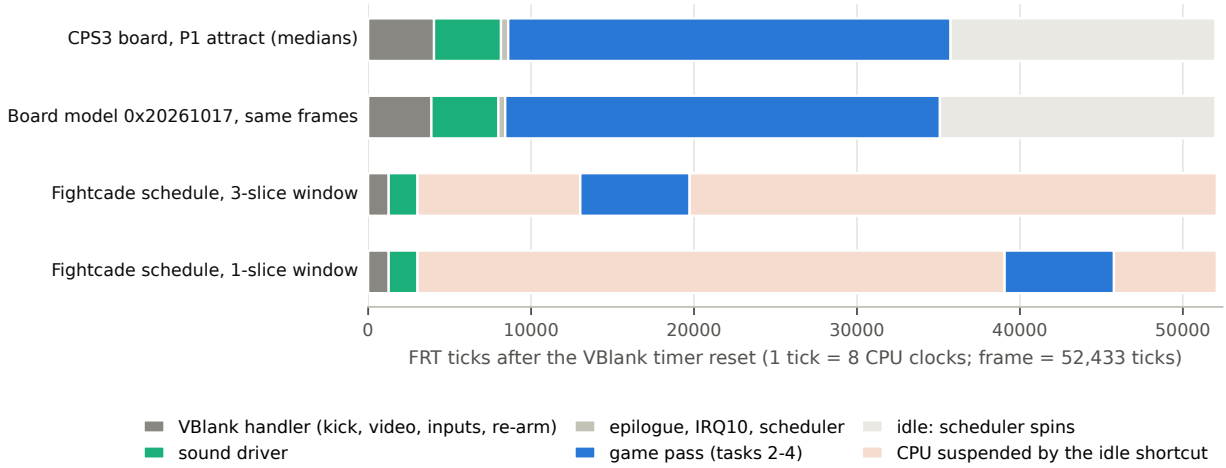


Figure 1: One frame, drawn to scale. Top: medians of the board's 685 recorded frames. Second: our board-cost model on the same frames. Bottom: the same probe image on Fightcade's schedule [1], in the three-slice and one-slice phases of its rotation (native machine, instruction-table costs). On the board the pass begins as soon as the VBlank handler ends and costs about four times as much.

Segment (median)	Board, ticks	ms	% of frame	Fightcade schedule, ticks
VBlank handler, timer reset to end of sound driver	8,152	2.61	15.5	3,027
of which the sound driver	4,111	1.32	7.8	1,766
Epilogue, IRQ10 service, scheduler to first dispatch	375	0.12	0.7	—
Game pass: first dispatch to last return (tasks 2–4)	27,156	8.69	51.8	6,708
Idle: last return to the next VBlank entry	16,205	5.19	30.9	suspended
Timer reset to next VBlank entry (PER)	52,000	16.64		52,009

Table 4: The board's frame in the attract fight (P1, BOARD; 685 frames, 675 for the pass after excluding the overrun and frames with unknown predecessors), against the same probe image on Fightcade's schedule (MODEL, native machine, 225 recorded frames). A frame is 52,433 ticks; the ticks between the VBlank entry and the reset (about 450, mostly probe wrapper) are not in PER. 1 tick = 8 clocks = 0.32 μs.

The pass spans 23,041 to 38,537 ticks (p95 31,832) in this fight. The idle time left after it ranges from 20,910 ticks down to 4,801. The ratios against the instruction table are 2.7 for the VBlank handler, 2.3 for the sound driver and 4.0 for the pass. **Under the instruction table a typical 52,000-cycle pass is 12% of a 25 MHz frame [1]; that is the emulator's accounting.** On the board the same kind of pass takes about 217,000 clocks, half the frame, and the VBlank handler takes another sixth. The game runs with roughly a third of the frame to spare, not seven eighths. Section 5 shows where the factor of four comes from.

4 Two probes and how their results came back

4.1 Constraints

Every physical number in this paper had to come back as pixels. The board is XeD's, each build gets one or two boots, and nothing can be read out except the screen. That fixed the design. A probe measures with the CPU's own free-running timer, stores compact records in private RAM, freezes them, and then draws them as rows of hexadecimal with checksums that a phone camera can capture.

The second constraint is that the probe must not change what it measures. Each probe image is the original program with these changes only: a private code block in `06700000`–`0670FFFF`, a range the original leaves at zero; a handful of hook words that redirect existing calls or vectors into that block (eleven in P2); and the three literals that make the program read its region from the kit's RAM copy. Private RAM lies between `0207A800` and `0207EFFF`, above the game's own start-of-day clear and below the kit's region bytes at `0207F000`. A fresh rebuild of each delivered image reproduced its bytes exactly.

Before shipping, each image ran through our whole-RAM differential harness over a 99-episode replay corpus. The original program and the probe image ran side by side, and every byte the probe writes outside its own range, every read the game makes of probe memory, every register the hooks return, and every scheduler dispatch were compared. That gate found a real defect in P1's boot wrapper. After the game's renderer returned, the wrapper's own checks changed the T bit of SR, and the caller yielded immediately; the task-switch code saved the changed SR into task 0's context (`0200D7AC`), and sixteen frames later the task resumed with a different flag. One episode diverged from frame 69. P2's wrapper saves R0–R7, MACH, MACL and SR around all of its work, and with that change the structural gate passed on all 99 episodes. P1, which ran on the board, carries the original defect. The model runs the same image, so the board comparisons in this paper are unaffected. **An instrumented program can change the game through a single saved status bit**, and only a byte-exact differential gate finds that. A probe that costs time can also tip marginal frames, so the same gate run with the probe's real instruction costs leaves timing windows open: 29 of them over the corpus for P2. The probes are diagnostics. They are not transparent copies of the game, and the analysis treats them that way.

The third constraint is that the probe's own cost is inside every number it records. P1's wrapper adds about 450 ticks between VBlank entry and the game's timer reset. Its status-row redraw every 23rd frame produces five 1,471–1,608-tick spikes at records 115, 253, 299, 483 and 529. In P2, the hashing done before a selected record is finalised delays the game's timer reset, so the records inside a window read about 2,600 ticks shorter than the first record of each window. These effects are identified in the data and kept out of every conclusion that does not survive them.

4.2 Probe P1: forty loops and seven hundred frames

P1 runs forty loops at boot, each timed by FRC over a fixed number of iterations. The loops cover cached ALU work, loads, cache-through reads and writes of every device, line fills from RAM, flash and BIOS, multiplies, branches, calls and the FRT registers (Appendix D). The game then starts normally. From the first live frame of the first attract fight, P1 writes a 24-byte record for each of 700 frames into a ring and freezes it. Each record holds: PER, the extended timer value at VBlank entry since the previous reset; PASSSTART and PASSEND, the first non-idle dispatch and the last non-idle return; VEND, the timestamp when the sound driver returns; the sound-driver and sprite-wait durations; the IRQ10 count, first-service time and handler time; the character-DMA output of the VBlank kick in 64-byte units; the random-number index at `020155E8` as a witness of trajectory; and flags for live play, a kick, and **slot 3 still running at**

VBlank. Running counters record the number of armed frames, live frames, slot-3 detections (OVR3) and the frame of the first detection.

P1's timestamp corrects a hardware trap. The game zeroes FRC in the masked VBlank handler but never clears the compare flag. A compare that is already pending at the reset therefore counts a period that the counter does not contain. P1 reads the flag immediately after the reset and subtracts one period when it is stale. This correction is the origin of P2's stale marker, and Section 6 shows it carries information about the frame clock.

4.3 Probe P2: a laboratory and five windows

P2 stores 80 records of 80 bytes in five windows of 16 consecutive live frames, starting at live frames 0, 120, 240, 480 and 640. Each record adds the raw FRC, FTCSR and TCR, the game's compare counter and the stale marker, a kick stamp, entry and return stamps for six task slots, two fingerprints of the player structures and the random-number area, and the four raw input words. At offsets 0, 7 and 15 of each window it logs the activation and return of slot 3 as events. After the last window it takes the machine over permanently and runs a laboratory of 226 experiments: 185 CPU loops, each sampled eight times (even samples after a requested cache purge, odd samples warm); twelve value tests of cache coherency and timer semantics; 28 DMA brackets; and a free-running series of 64 VBlank timestamps. The results are drawn as 63 pages. Each page has sixteen 16-byte data rows and two parity rows over GF(256), every row carries a CRC-16 bound to the build and row number, and the whole capture carries a CRC-32.

4.4 Recovering the data

P1, second return. All forty loop results were read, each in several photographs. One digit of B23 stays ambiguous (`3958` or `395B`, three ticks), and both readings are carried. Of the 700 ring records, 604 were accepted by a first reader with checksum-valid readings in at least two frames. A second reader, trained on those labels with each target page held out, reproduced 603 of the 604 rows and 27,783 of 27,784 characters, and added 81 rows under the same rule. That makes 685 rows usable. Eight visible rows stay unresolved and seven were never filmed. They are kept as unknown, never filled in.

P2. XeD's PSone LCD samples the CPS3's line with a drifting pixel clock (period about 136 source pixels), so one-pixel strokes vanish in bands and a plain font reader reads 0 as 2. The reader therefore models each column's sampling phase. 1,122 rows were read identically in at least two frames, and four single-frame rows were confirmed by their page's parity. Four missing data rows were rebuilt from parity, and no two valid readings conflicted. The 16,128-byte capture's CRC-32 (`881D2B40`) equals the value the board printed. **The P2 capture is the board's memory, bit for bit.**

The completion boundary. P2's laboratory ran tests 0–186 (all 185 CPU loops and two coherency tests) and then took an exception on its first execution of code written to RAM, test 187. Every vector points to the probe's trap, which stored the stacked PC `0202AA00` and went to the readout. Tests 188–225 never ran. These include the timer-semantics tests, the DMA brackets and the free-running VBlank series; their result slots are zero and are treated as missing, not as zero. Section 8.4 explains the fault. Every recovered value is in the companion data [6].

5 What the CPU pays

5.1 Reading a loop

Each experiment is a loop of N iterations around a short body, timed by FRC from a call at a fixed address. Its total includes the body, the loop's own branch and counter, the call and return,

Return	Recovered	Not recovered
D1–D4 (freeze builds)	SIMM sums, crash-page absence, interrupted PC/PR, interrupt and task breadcrumbs	a time-ordered event log
P1, first return	8 photographs of running counters and maxima (first overrun frame, OVR3, LIVE, armed frames)	loop results, ring rows
P1, second return	40/40 loops (B23 to 3 ticks); 685/700 ring rows; summary page	8 unresolved rows, 7 unfilmed
P2 (MAIN)	entire 16,128-byte capture, CRC-verified: 80 records, 30 events, 185 CPU tests × 8 samples, 2 coherency tests, the fault PC	tests 187–225 (not run); no CONTROL boot

Table 5: What came back from the board.

the timer read and the instruction fetches. Subtracting the empty loop (4.117 clocks per iteration on the board) and dividing by the number of operations gives an *effective* cost per operation within that instruction stream. It is a throughput, not an isolated latency. Where two lengths of the same body were run, P2 takes the slope between them, which removes the fixed costs. Every value below is a measurement of this kind, and the raw integers are in the companion data.

5.2 The device costs

Figure 2 sets every loop against what the instruction table charges for it. Instructions that stay in the cache cost what the table says. Everything that leaves the CPU costs between 2.8 and 46 times as much, and the multiplier depends on the device, the width and the direction. **No single CPU multiplier, wait-state number or uniform memory latency can describe this board.** Increasing every cost would make the cached ALU loops wrong without fixing the video writes.

5.3 Line fills, and an 8-bit BIOS

Three P1 loops stream through more data than the cache holds, so every 16-byte line is fetched from its device. Their effective additions are 20.1 clocks per flash line, 36.0 per work-RAM line and 112.2 per BIOS line BOARD.

The flash value can be cross-checked inside the same return. B12 calls an 8 KiB straight-line body of `NOP`s, twice the cache's size, so every fetch line misses: 1,031 flash line fills. Solving B12 for the flash line cost gives 20.06 clocks. With that value fixed, B10 (a flash *data* stream) is predicted to within 0.09%, and B13 (a mostly warm 1 KiB instruction body) to within 0.38%. **Instruction fetches and data reads from the program flash pay the same 20-clock line fill.**

The BIOS value settles the BIOS bus width. With the programmed long wait (seven clocks per transfer), an 8-bit bus predicts 61,537 and 117,364 clocks for B22 and B23, against 60,480 and 117,440–117,464 observed. A 16-bit bus predicts 32,865 and 60,020. The board's own documentation lists the BIOS as a 512k×8 flash, and the SH-2 takes area 0's width from its mode pins, not from BCR2. **The BIOS bus is 8 bits wide**, so code and data fetched from the BIOS cost twice what a 16-bit model charges. This matters only before the game starts and inside the stray-interrupt path of Section 8.

Operation (P1 loop)	Clocks	Reading
cached ALU instruction	0.993	one clock, as in the table
cached load, literal-pool load, stack load	1.49–1.50	fetch and memory access share a port: half a clock on average
taken `BRA+NOP`; `JSR+NOP+RTS+NOP`	2.98; 5.99	as in the table
`MUL.L` / `DMULS.L` repeated; `MULS.W`	3.75; 1.87	issue rate, not result latency (Section 5.5)
work RAM, cache-through read: byte / word / long	6.20 / 6.21 / 10.60	16-bit DRAM bus; a long is two transfers
work RAM write: byte / word / long	3.73 / 3.74 / 7.95	write buffer overlaps part of the bus cycle
work RAM line fill (streamed read + add)	35.97	eight 16-bit transfers
program flash, cache-through long read	6.01	$2 + 1$ wait $+2$ idle $+1$
program flash line fill	20.06	four 32-bit transfers of five clocks
BIOS, cache-through long read	29.02	four 8-bit transfers of seven clocks, plus one
BIOS line fill	112.2	sixteen 8-bit transfers of seven clocks
colour RAM write: word / long	7.43 / 15.32	
sprite RAM: long write / long read	19.51 / 21.59	
character-RAM window, long write	39.54	about 20 clocks per 16-bit word
score-screen RAM: word write / long write / word read	17.17 / 34.94 / 19.41	byte-wide SRAM on the sound chip
input port or I/O register, word read	5.01	
FRT counter or status, byte read	12.00	slow end of the manual's 3–12 cycles

Table 6: Effective cost per operation on the board (P1, BOARD), after subtracting the empty loop. Cache-through accesses use the 2xxxxxxx aliases. The reading column gives the decomposition that fits; where it is not unique it says so.

5.4 The cache: four ways, sharp thresholds, 36 clocks per conflict

P2's conflict loops load one word from each of d different lines per iteration, 256 iterations, then return to the first line. At a stride of 1,024 bytes all d lines map to the same set; at a stride of 16 they spread over neighbouring sets. Figure 3 shows the result in the three CCR modes.

In the normal four-way mode, four lines in one set stay hits and five miss on every load. In two-way mode the cliff moves to three lines. With the cache off, both strides cost the same and grow linearly. Spread controls never cliff. Each conflicting load adds 35.73 to 35.94 clocks in all six thrashing cases, the same 36-clock work-RAM line fill measured independently by P1. The hit path is exact arithmetic: the spread controls take $163 + 64d$ ticks for $d = 1 \ldots 6$, which is five clocks of loop per iteration, two per line (the load and the pointer add), and a fixed 24 clocks. **The cache is the manual's four-way cache with the manual's two-way mode, and a set holding one line more than it has ways misses on every access**, as least-recently-used replacement does on a cyclic pattern.

Instruction working sets show the same structure. Four calls through a 4 KiB straight-line body take 1,349 ticks warm in four-way mode against 1,815 after a purge. The same body in two-way mode takes 3,639, because half the cache is gone. An 8 KiB body takes about 7,218 ticks in either mode, and a 1 KiB body about 263.5. The loop's caller, return, literals and timer read compete for the same sets, so "fits in 4 KiB" is a property of the whole working set, not of the body alone.

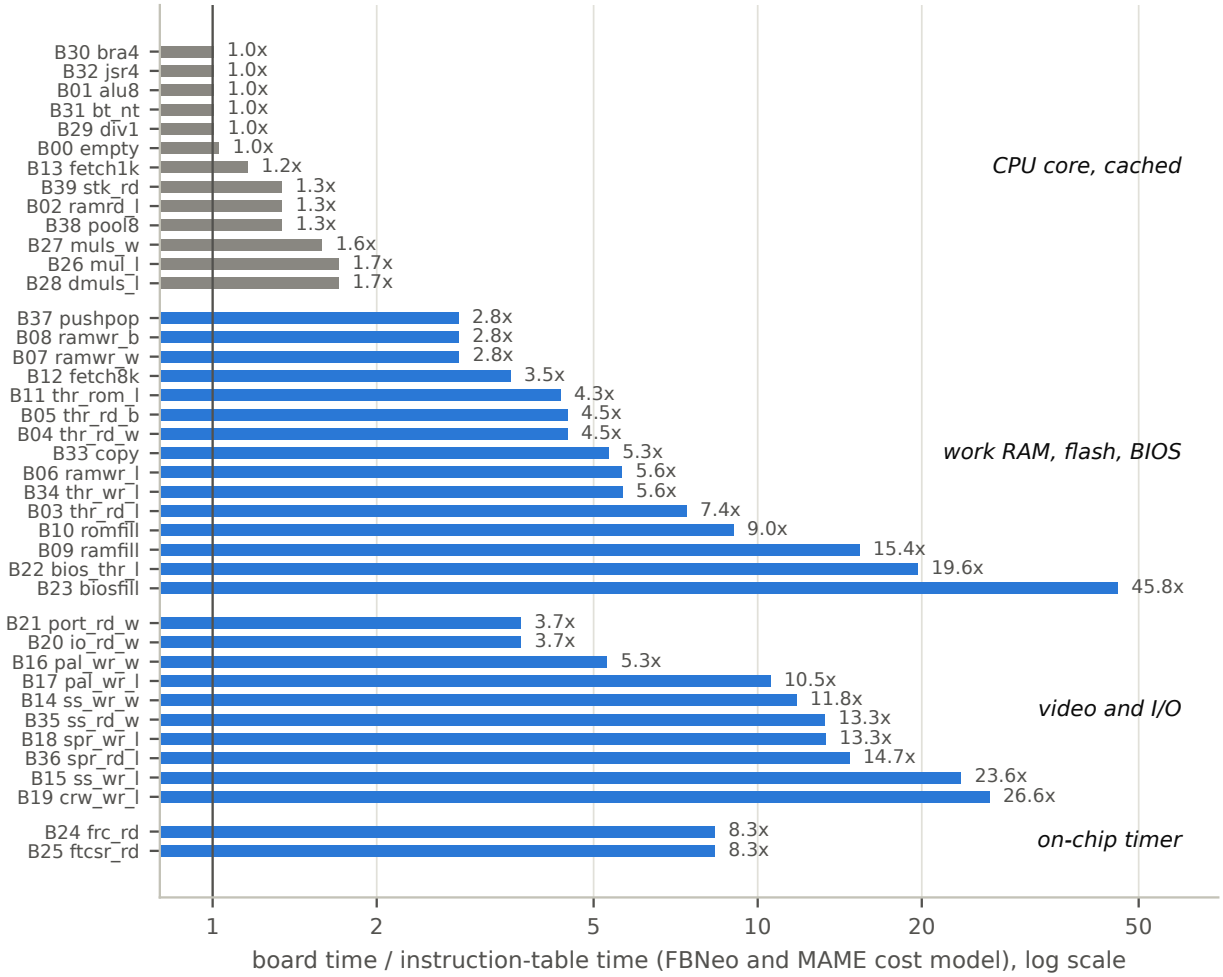


Figure 2: All forty P1 loops: the board's total over the instruction-table total for the same instructions (FBNeo's cost table, which MAME's SH-2 core matches for these instructions). Grey: the cached CPU core. Blue: everything that crosses the bus. Log scale.

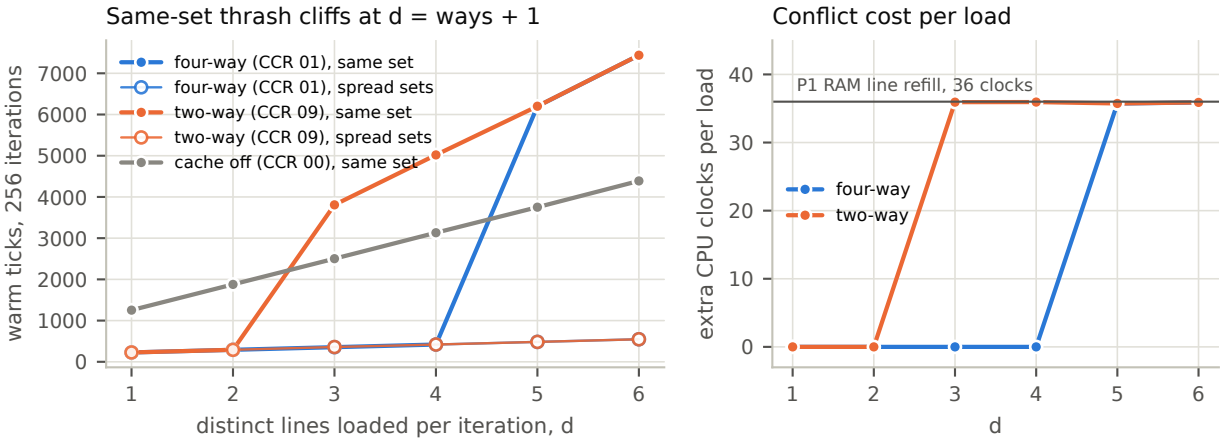


Figure 3: Cache conflict loops (P2, warm medians, BOARD). Left: total ticks; filled markers same-set, open markers spread-set controls. Right: the cost of the conflict per load, $8(T_{1024} - T_{16})/(256d)$, with nothing fitted.

The consequence for any program meant to time like the original: **a load costs 1.5 clocks if its line is cached and 37.5 if the set is full**. Moving a function by a few hundred bytes, or a table by a kilobyte, changes which lines share a set. That changes the cost of a frame by amounts that dwarf any single instruction difference. Arithmetic equivalence is not timing equivalence, and neither is an equal instruction count.

5.5 The pipeline hides work

Three groups of P2 loops show that the board overlaps work in ways a serial cost table cannot express (Figure 4).

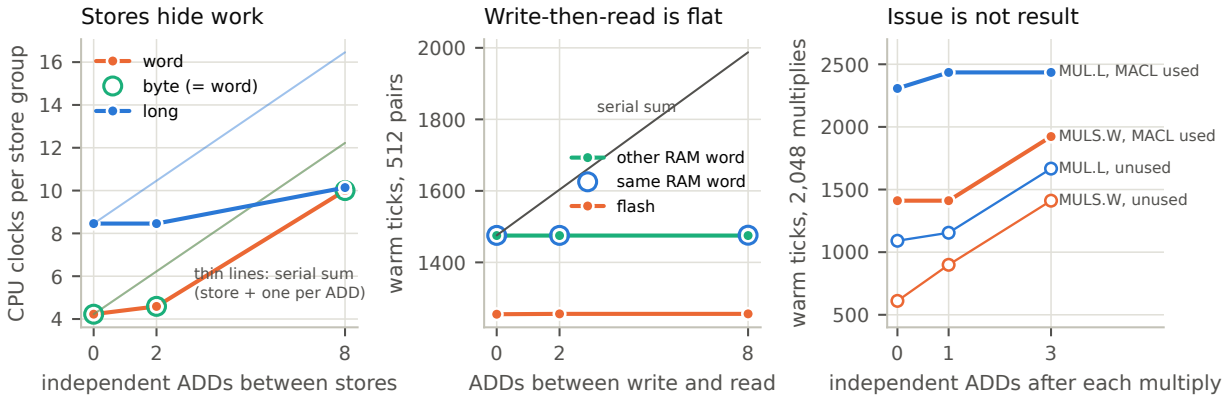


Figure 4: Overlap on the board (P2, warm medians, BOARD). Left: clocks per group of one store plus 0, 2 or 8 independent `ADD`s (two-length slopes); thin lines show what a serial model predicts. Byte and word stores are identical. Middle: 512 long write-then-read pairs with 0, 2 or 8 `ADD`s between them; the thin line is the serial sum. Right: 2,048 multiplies with 0, 1 or 3 independent `ADD`s after each, with and without reading the result.

Stores. A long store costs 8.46 clocks per group with no `ADD`s and still 8.46 with two; with eight it costs 10.14. A serial model would add two and eight clocks. Byte and word stores cost 4.22 with none, 4.59 with two and 10.0 with eight. The write buffer lets the CPU run ahead while the bus completes the store, so up to about the duration of the bus write, independent instructions are free.

Write then read. 512 pairs of a long RAM write followed by a long RAM read take 1,475.5 ticks with no `ADD`s between them, 1,475.5 with two and 1,476 with eight. That is 4,096 extra instructions for half a tick. The read has to wait for the write to reach the bus anyway, and the instructions in between fill that wait. Same-address and different-address pairs cost the same, so the board does not forward a pending store to a load. Pairs whose read comes from flash take 1,255–1,256 ticks, also flat.

Loads and their consumers. A cached load followed immediately by an instruction that uses its result costs the same for byte, word and long (898.5–899 ticks per test). One independent instruction between them raises the total to 1,026–1,027, and four raise it to 1,922–1,923. A single load-use interlock plus the fetch/memory port contention reproduces all nine within 0.3%.

Multiplies. Table 7 gives the warm totals for 2,048 multiplies of each kind. Reading the result roughly doubles the cost of a long multiply. Adding two more instructions between multiply and use costs nothing at all, because they fill time the result path takes anyway. A word multiply can hide one instruction completely but not three. A model that charges a multiply a fixed number of clocks fits the repeated-multiply loops of P1 and misses these by up to 31% (Section 9).

The timer registers. 1,024 pairs of FRC high/low byte reads with 0, 1, 2, 3, 4, 5, 7 or 11 `NOP`s

	result not read			result read and used		
Instruction	gap 0	gap 1	gap 3	gap 0	gap 1	gap 3
`MUL.L`	1,091	1,155	1,667	2,307	2,435	2,435
`MULS.W`	611	899	1,411	1,411	1,411	1,923
`DMULS.L`	1,091	1,155	1,667	2,307	2,435	2,435

Table 7: Multiply issue and multiply result are different states (P2, ticks, BOARD). "Gap" is the number of independent `ADD`s after each multiply; "read and used" means `STS MACL` followed by an `ADD` that depends on it. Operands are the constants 3 and 5.

between them take 3,202, 3,459, 3,714.5, 3,714.5, 3,714.5, 3,970, 4,227 and 4,738 ticks. The cost rises in steps of about 256 ticks (two clocks per pair), with a plateau from two to four `NOP`s. This fits the manual's account of on-chip timer accesses, which synchronise to the peripheral clock and take 3 to 12 cycles depending on phase: the instructions between the reads change where the next access falls. Together with the read-spacing and copy loops, this is where our model is still furthest off (Section 9).

5.6 One board, two sessions

P2 reran P1's forty loop bodies with the same iteration counts under a different runner, two days later. Thirty-eight of the forty P2 warm medians are within 1% of P1's values. The two exceptions are explained by the runner: the empty loop differs by 2.5% because the call path changed, and the 1 KiB fetch loop is 13.7% faster warm than P1's single cold measurement because P2's warm sample starts with the body cached. The sound-driver medians of the two returns' live traces agree to 1.5 ticks (4,111 and 4,112.5). **The board's costs are stable to well under 1% between sessions.**

6 Interrupt order and the frame clock

6.1 Two bands of IRQ10 service

On the board the DMA interrupt is never taken during the VBlank handler: the handler masks to level 15 at `06000576`, and a character batch kicked at the start of the handler finishes while the mask is still up. IRQ10 is serviced only when the handler returns, so its service time follows the end of the VBlank work, not the end of the transfer. P1 shows this directly. Across 483 frames, the first IRQ10 service stamp correlates with the VBlank-end stamp at 0.998. The delay between them falls in two narrow bands, 384 records at 151–166 ticks (median 160) and 98 records at 217–231 ticks (median 224), with nothing in between; the one remaining record, at 444 ticks, is the late IRQ10 of Section 7 BOARD.

P1 could not say which frames fall in which band. P2 recorded, for every frame, whether the timer compare had been left pending across the reset (the stale marker). **The marker splits the bands exactly**: all 47 unmarked frames sit at 153–167 ticks (median 159) and all 20 marked frames at 219–231 (median 224.5) BOARD. The difference, 65.5 ticks or 21 μs, is the FRT compare handler. It has priority 15, so when the handler unmasks with both the stale compare and IRQ10 pending, the timer handler runs first. Figure 5 shows both returns and the model.

6.2 Where the fourth compare lands

OCRA = `3333` puts four compares in each frame, the fourth roughly 52,431 ticks after the handler's reset, a few ticks before the next one. Where exactly it lands decides three different outcomes

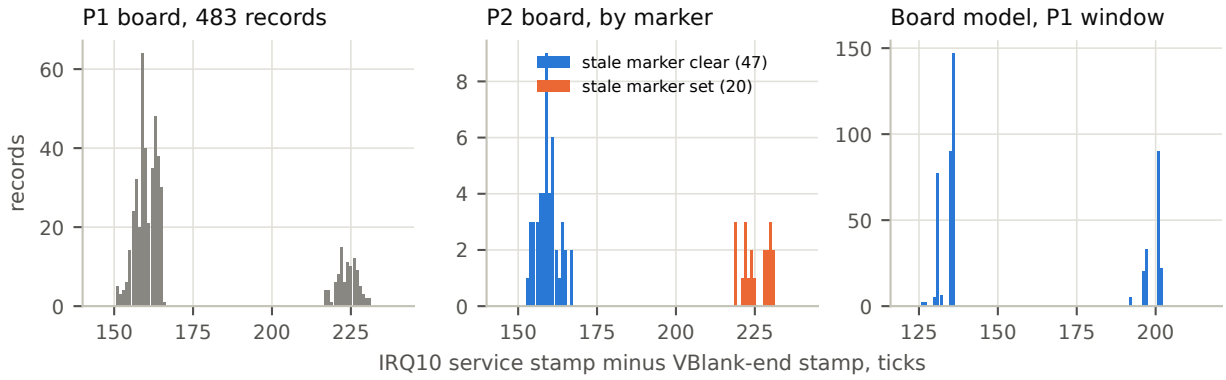


Figure 5: Delay from the VBlank-end stamp to the first IRQ10 service stamp. Left: P1, 483 frames. Middle: P2, split by the stale marker that P1 lacked; the split is exact. Right: our model on the P1 frames, which reproduces the split about 24 ticks early (a faster interrupt-entry path).

(Figure 6).

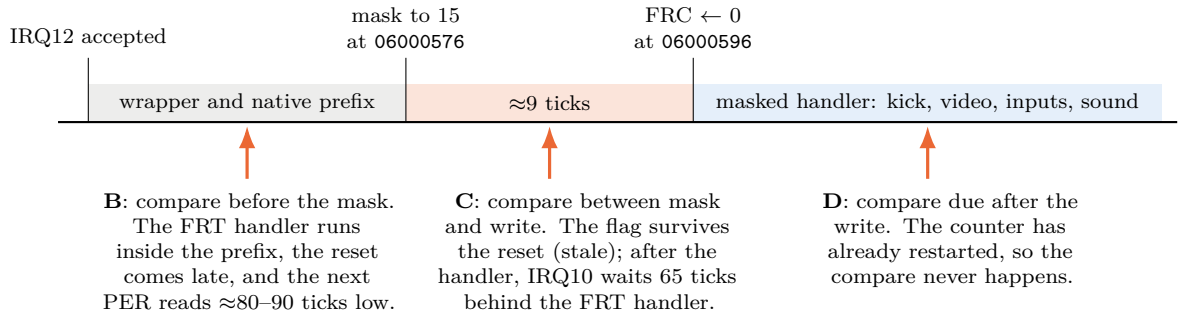


Figure 6: The game's fourth timer compare relative to the start of the next VBlank handler (not to scale). Which case occurs depends on the frame length, the compare period and the prefix cost.

All three cases are visible in the data. Case C is the stale marker, and with it the high IRQ band. Case B shows as a cluster of PER values about 90 ticks below the main body, because the timer handler delays the reset that ends the next interval. Case D predicts that no record ever shows compare count 4 without the marker, and none of the 80 P2 records does: the 57 unmarked records have count 3, the 23 marked records count 4. The probe's extended timestamp can be rebuilt from the stored fields, $E = \mathrm{FRC} + 13{,}107 \min(7, C + [\mathrm{FTCSR} \,\&\, 8]) - 13{,}107\,\mathrm{STALE}$. It differs from the recorded timestamp by exactly 24 ticks on every unmarked record and 22 on every marked one, the cost of the two code paths between the reads BOARD ROM.

The timer handler only debounces the coin and start switches, so it never changes the course of a game [1]. It does change the time. In 20–29% of frames it moves the DMA service by 21 μs, and in case B it moves the VBlank handler's reset by about 80 ticks. These are small and gameplay-inert, but they are real board timing.

6.3 How many CPU clocks in a frame

The stale band answers a question the probes were not designed for. Let F be the frame in timer ticks, p the prefix from VBlank entry to the counter reset, and j the delay before IRQ12 is accepted. Then $\mathrm{PER}_k = F + j_k - j_{k-1} - p_{k-1}$, and the acceptance delays cancel in the mean. A reset is stale exactly when the interval between resets lands in the mask-to-write window after the fourth compare. The window sits at a fixed point P_4 after the previous reset, and is a' wide, the

cost of the 16 instructions from 06000576 to 06000596: between 6 and 10 ticks, two of them being timer-register writes, depending on whether the handler's first work-RAM read misses the cache. Records whose *next* record is stale therefore have PER in a band centred on $c = P_4 - p + a'/2$ (Figure 7). The prefix p then cancels between the mean and the band:

$$F = \overline{\mathrm{PER}} + P_4 - c + \tfrac{a'}{2} + s\,P(B), \tag{1}$$

where $s\,P(B)$ restores the case-B intervals that the timer handler shortened.

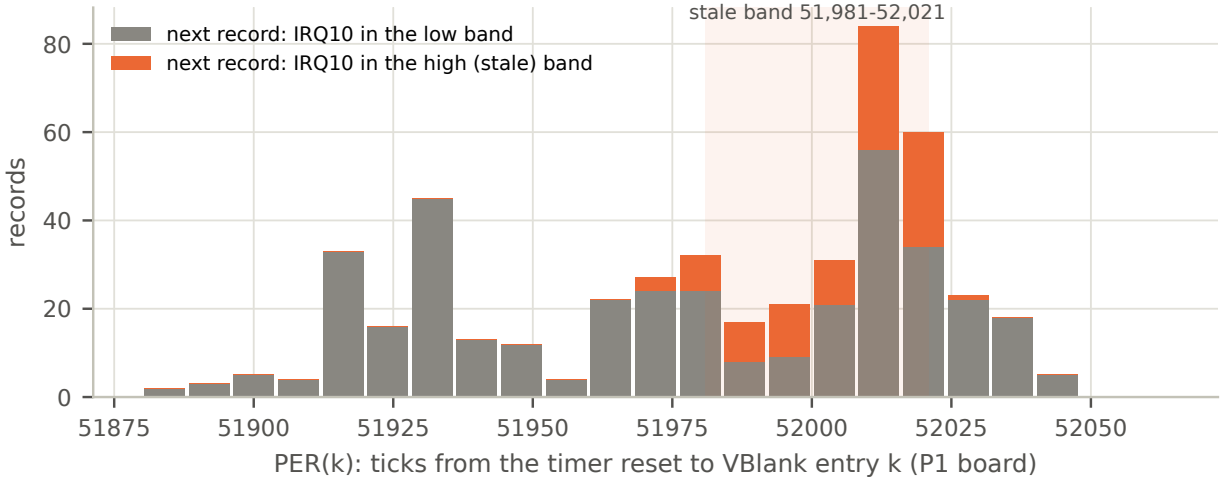


Figure 7: P1's PER values, coloured by whether the *next* record's IRQ10 falls in the high (stale) band. Stale successors occur only in a band 40 ticks wide; the low cluster near 51,920 is case B.

Equation 1 is only as good as its estimate of $P(B)$, and the estimate is biased by the way frames with late interrupt acceptance pair with frames after them. Rather than argue about the bias, we measured it. The same probe image was run from power-on on two builds of our model whose frame lengths in timer ticks are known exactly. Build 24's timer saw 52,436.9 ticks per frame, because of a carry error since fixed, with a period of 13,107. Build 26's timer sees 52,433.24 ticks with the manual's period of 13,108. The estimator was then applied to both, and to the board, without changes (Table 8).

Records	$\overline{\mathrm{PER}}$	band centre c	P_4	truth	estimate	error
model build 24 (700)	52,011.0	52,028.5	52,428	52,436.9	52,427.9	−9.0
model build 26 (700)	52,008.9	52,024.0	52,431	52,433.2	52,428.3	−4.9
board, P1 (685)	51,984.1	52,001.0	52,431	—	52,430.8	corrected: 52,437.8

Table 8: The frame-length estimator tested where the answer is known (MODEL) and applied to the board (BOARD); ticks, $a' = 6$, $P(B)$ from lows that follow ordinary records. With $a' = 10$ every raw estimate rises by 2 ticks and every corrected one is unchanged, because the self-test absorbs it. Two other ways of estimating $P(B)$ have emulator biases of −15 and −3 ticks; corrected the same way they give 52,439.0 and 52,441.2 for the board.

After correction the three variants agree to within 3.4 ticks. Corrected with build 26 alone, whose timer period is the board's, the three give 52,434–52,439 (the variant of the table 52,435.7); with build 24 alone, 52,440–52,444. Allowing for the spread of the bias between the two builds (4 to 10 ticks), the statistical error of the mean and band (2 ticks), and the variants' disagreement:

$$F = 52{,}439 \pm 5 \text{ ticks} = \mathbf{419{,}512 \pm 40} \text{ CPU clocks per display frame,}$$

107 ± 95 ppm above the nominal ratio 25,000,000/59.5995 = 419,467. The result assumes the board's compare period is the manual's OCRA + 1. With OCRA it would be 3 ticks lower. P2's 70 interior records cannot add much: their stale band rests on 18 records, and the probe's hashing changes the prefix structure the correction was calibrated on. Uncorrected, they sit about 13 ticks below P1's uncorrected value, with several times its uncertainty.

The stale share is an independent check on the sign. It depends on how far F sits past P_4, relative to the spread of the reset jitter. Our model, with F exactly nominal, marks 33.9% of P1's frames in the high band. The board marks 20.3% (98 of 483). Reaching the board's share in the model takes a frame about 7–13 ticks longer than nominal. That has the same sign as Equation 1's $+6 \pm 5$ and overlaps it.

6.4 The absolute clock

P2's readout advances a page each time its free-running timer has counted 6,250,000 ticks since the page was drawn. Drawing is not counted, and our model measures it at 216,192 clocks (189 page intervals of 119.71 frames). The phone video timestamps 107 page changes over three cycles of the readout, and a robust fit gives a period of 2.008276 s (standard error 34 ppm). Then

$$f_{\mathrm{CPU}} = \frac{50{,}000{,}000 + 216{,}192}{2.008276\ \mathrm{s}} = 25.0046\ \mathrm{MHz} \pm 165\ \mathrm{ppm},$$

where the error combines the fit (34 ppm), a 3% uncertainty in the draw cost (129 ppm) and an uncalibrated phone clock (100 ppm assumed). **The CPU runs at its nominal 25 MHz to within 0.02%.** Together with the frame ratio,

$$f_{\mathrm{display}} = 25{,}004{,}627/419{,}512 = \mathbf{59.604 \pm 0.011\ Hz},$$

consistent with the nominal 42.954545 MHz/6/455/264 = 59.5995 Hz. It sits about two standard deviations above the 59.583 Hz sometimes quoted for CPS3 cabinets. The CPU crystal is on the cart and the video crystal on the main board, so the ratio is a property of one cart in one board. A reference clock filmed alongside the next readout would make both numbers 30 ppm measurements.

7 Graphics DMA

7.1 IRQ10 means a DMA finished, and nothing else

Every recorded frame on the board fits one rule: **IRQ10 is raised once when a DMA completes, and never otherwise.** In P1's 685 frames, the 483 frames whose VBlank kicked a character batch each took exactly one IRQ10. The 200 frames without a kick took none, and no frame took two. The only exceptions are the first two records of the ring, which have an IRQ10 and no character batch BOARD. P2's 80 frames agree: 66 with a batch and one IRQ10, 12 with neither, and two with an IRQ10 and no character output, one of them the first record.

This is the most consequential single difference between the board and FBNeo. Fightcade's driver, and upstream FBNeo's, raise IRQ10 every third CPU slice whether or not any DMA ran (1.33 times per frame) [7, 8]. In those emulators that periodic interrupt is also what wakes the CPU after the idle shortcut, and so decides their slowdown [1]. MAME has no periodic source. Our model raised it until build 25 and showed one to three IRQ10s per frame on P2. With the periodic source removed it matches the board's IRQ10 count on all 685 P1 frames MODEL.

7.2 The DMA cost the CPU sees is CPU work

On the board the VBlank handler grows with the size of the character batch it kicks. In P2, the stamp taken right after the kick rises 2.65 ticks per 64 output bytes (correlation 0.90). The post-kick VBlank work, sound driver excluded, rises 1.40 ticks per 64 bytes (0.82). In P1, VBlank end minus sound rises 2.97 ticks per 64 bytes over 483 frames (0.83) BOARD. Bus contention between the DMA engine and the CPU would produce exactly this kind of slope, but a run of the same probe on our model, which has no contention at all, produces 2.52 and 1.58 ticks per 64 bytes (Figure 8). The slope is the kick routine `06138168` copying three longwords per queued command into the DMA engine through the character window, at about 20 clocks per word, plus the decompression bookkeeping of a longer command list ROM MODEL. **At the batch sizes of live play, no bus-contention term is needed.**

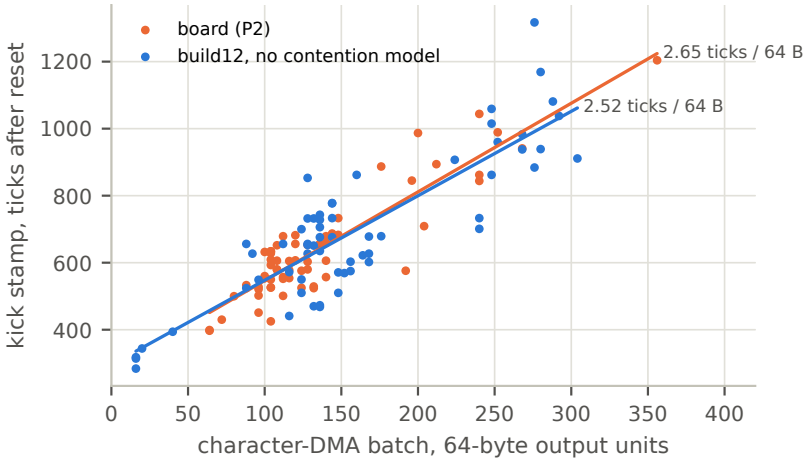


Figure 8: Time to finish the VBlank kick against the size of the batch it starts. Orange: the board (P2). Blue: the same probe on build 12 of our model, which charges the kick's instructions but models no bus contention.

7.3 How fast the transfer is

Because IRQ10 is masked for the whole VBlank handler, a batch that completes before the handler ends is serviced at the same delay as any other. The delay from VBlank end to the IRQ10 service does not depend on batch size (correlation −0.19), so every live batch up to the largest, 22,784 bytes, finished before the handler's end. That batch was kicked 1,204 ticks after the timer reset, and the handler ended at 9,582. **The engine therefore delivers at least 8.5 decompressed bytes per microsecond** BOARD.

P1's largest batch, 24,576 bytes, is the one record whose IRQ10 arrived late: 444 ticks after VBlank end instead of about 160. If the extra 284 ticks are transfer time, the rate is about 9 bytes per microsecond, just above the bound. Part of that delay may also be a masked dispatch interval, so the record supports the bound without fixing the rate.

The batches that accompany the census's tightest live frames are 14–31 KB. At 9 bytes per microsecond, 31 KB is 3.4 ms, about a fifth of a frame. Loads at transitions are another matter. From power-on, the attract sequence issues single batches of 1,949,696 bytes three times and 1.46 MB once MODEL. At the measured bound the largest takes at most 230 ms, about 14 frames; the true duration is unmeasured. Our model completes every batch instantly, as FBNeo does. Section 9.4 shows that this has not shifted the attract sequence by a single frame before the first fight, so the game does not wait for these loads in any way that changes its course there. An FPGA implementation whose decompressor at first ran slower than its memory allowed reports

the same from the other side: the attract transitions showed character RAM still half written, and the game did not wait for it [9].

7.4 Two smaller facts

The late first IRQ10. In both returns, the first record of the fight has an IRQ10 about 12,000 ticks after VBlank end, with no character batch: 12,263 ticks in P1 and 11,970 in P2. The palette request routine `06137DF0` starts a palette DMA at once when the engine is idle, without waiting for VBlank ROM. A palette load issued from the game pass at the start of the fight would produce exactly this record. The probes did not record palette requests, so the source is a strong candidate rather than a measurement.

The sprite-list wait is free. The VBlank handler waits at `06136996` for the sprite-list DMA it triggered. P1 measured 42–45 ticks for the wrapped call and P2 45–47. Our model, whose sprite-list DMA completes instantly, reports 45–48 ticks for the same wrapper, so the physical wait is under one tick. The sprite list is copied in the few microseconds between trigger and wait.

8 Why programs fail on the board and not in an emulator

Before any timing was measured, the board was already teaching us. A modified version of the game (the 3v3 team mode of our project) booted and played on the board, but its team HUD never appeared, and calling a team assist froze the machine. The same program ran cleanly in every emulator, in a strict interpreter that faults on misaligned accesses and undefined opcodes, and under sixty variants of DMA and CPU timing. Four diagnostic builds and a static review found three physical behaviours that no emulator reproduced. The P2 laboratory then found a fourth. Each of them is a property of the hardware, and they apply equally to anyone else who modifies or re-implements the program.

8.1 The score-screen RAM is eight bits wide

The score-screen (text) layer's RAM at `05040000` is a single 32k×8 SRAM on the sound chip. A CPU longword access sees one real byte, at offset 1 of each 16-bit half. The original program knows this ROM:

- the BIOS power-on test at `00000D04` writes `00FF00FF` and masks the readback with `00FF00FF` before comparing, while work, sprite and character RAM are compared in full;
- the game's cell writer `060D5E18` stores the tile number in one 16-bit half and ORs tile bit 8 into bit 0 of the attribute half, because the even byte that would carry bit 8 does not exist.

MAME maps the RAM byte-wide behind a `00FF00FF` lane mask and takes tile bit 8 from the attribute byte [3]; FBNeo and Fightcade store full 32-bit cells and decode tile bit 8 from the even byte [7, 8]. Both FPGA cores store one byte per 16-bit half, as the board does [4, 9]. FBNeo's driver is a port of MAME's early CPS3 driver [8], and FBNeo is the emulator the modification scene tests against, so a modified program can pass every test that scene has and still fail on the board for this reason alone. Our HUD checked ownership of a cell by comparing a full longword with `00350002`, which never matches on a board, and drew tiles `130`–`13F` as `01300002`, which a board shows as tiles `030`–`03F`. Masking the comparison with `00FF00FF` and setting the attribute's bit 0 fixed both, and the board confirmed it BOARD. **Any read-back of score-screen RAM must ignore the even bytes, and tile bit 8 lives in the attribute.**

8.2 Hardware read through the cache

The program reaches hardware through the cache-through aliases (`25000000` for inputs, `24080000` for colour RAM). Our modification read the Start button through the cached alias `05000000` and captured palette entries through `04080000`. With the cache on, the first read fills a line and later reads can return the old value: a held button that never releases, or a palette captured before the DMA that changed it. Emulators have no cache, so both work there. Both reads were moved to the cache-through aliases in the board repair.

8.3 The freeze: a read beyond work RAM

The freeze was the hard case. The diagnostic builds narrowed it down step by step BOARD:

1. **D1** printed sums of both program SIMMs at boot. They matched the delivered files, so the flash held the intended program. It also replaced the game's silent exception loop with a crash page. On the assist call the picture froze with the assist's attack on screen and *no* crash page: no CPU exception had been taken.

2. **D2** printed the interrupted PC, PR and a counter on every VBlank. The frozen screen read `06137C20` / `06000948` with the counter stopped: the last VBlank arrived during the scheduler's DMA-queue swap, and after that no VBlank was serviced at all.

3. **D3** bounded both of the game's masked hardware waits (the DMA-complete wait at `06138100` and the sprite-list wait at `061369AC`) and redirected the 114 interrupt vectors that point into the BIOS to the crash page. The freeze came back with neither wait timing out and no crash page, at PC `06000DC6`. It was intermittent: across three boots it struck on the second call, on the first call, and not at all.

4. **D4** added a breadcrumb. Every VBlank routine had returned, and the last dispatch was the game task.

Between builds XeD also established that assists of other characters, called from either side, froze the same way. A second setup, the UltraSIMM flash they use at local events, froze too. That ruled out one character's data and one faulty set of SIMMs.

`06000DC6` is not a crash address. It is the first instruction after the scheduler's resume sequence (`LDC R0,SR`; `LDS.L @R15+,PR`) at which an interrupt can be accepted (MANUAL 4.6), so a pending VBlank is naturally taken there. D2–D4 all show the same thing: an ordinary interrupt arrival, then nothing. An unmasked loop would still be interrupted by VBlank. The program's only masked waits on hardware were bounded in D3 and did not time out, and a lint of the modified program found no other code that runs masked. Something stopped the CPU inside a single instruction.

The static review found the instruction ROM. Our assist code calls a hook at `06ED7900` for every assist, which, during the assist's action phase, reads three bytes of its scratch area as slot numbers of a 128-slot effect pool. At the moment the assist lands, those bytes still hold the assist's entry velocity, $v_x = d_x \ll 11$. The hook accepted any non-zero byte and read a longword at $\texttt{02028990} + (id - 1) \cdot \texttt{800} + \texttt{3B8}$ *before* checking ownership. For an entry distance of 118 pixels the third byte is 176 and the read goes to `02080548`, beyond the end of work RAM at `0207FFFF`. For −120 it is `020A7D48`. Entry distance depends on where the caller stands, which explains the intermittency. Emulator traces already contained reads at `020A6D48`–`020A7D48`: the emulators returned a value, the ownership check rejected it, and play went on.

On the board, area 1 (`02000000`–`03FFFFFF`) is configured with external WAIT, and only its first 512 KiB is populated. **A read that no device answers leaves the bus cycle open, and the**

SH-2 cannot take an interrupt in the middle of a bus cycle: the picture holds, no handler runs, and no diagnostic can draw. We bounded the slot numbers to 1–128 before the address is formed, using the same fourteen instructions. That repair, delivered together with the score-screen and cache-alias fixes, cured the freeze on the board. The mechanism of the hang (WAIT never released) is inferred from the diagnostics and the bus configuration. That the out-of-range read caused the freeze is established by elimination and by the repair.

On a CPS3, an access to an unpopulated address can stop the machine. Every emulator we know returns zero or open bus and carries on.

8.4 Cache coherency: the purge waits for the store

P2's two data-coherency tests and its fault expose the cache's ordering rules. Each repetition stores 17 through the cache-through alias `2202AA00`, writes CCR = `11` (purge, cache on), reads `0202AA00` through the cache, stores 34 through the alias, and reads the cached address again. The second test inserts a second CCR purge before the second read. The two reads are packed as bytes, so `1122` means 17 then 34. Table 9 shows the result.

Test	Rep. 0	Rep. 1	Rep. 2	Rep. 3
data_alias (one purge)	`1111`	`1122`	`2222`	`2222`
data_purge (two purges)	`2222`	`2222`	`2222`	`2222`
any emulator, every repetition	`1122`	`1122`	`1122`	`1122`

Table 9: The board's coherency values (P2, BOARD). A memory with no cache, or a purge that takes effect at once, gives `1122` everywhere.

A purge that never works fails repetition 1. A purge with a fixed delay fails the second test. One rule fits all eight values: **the CCR write is ordered behind the pending cache-through store in the write path, while cache hits are served at once.** A read that hits returns the old line before the purge lands. A read that misses waits for the bus, so the store and the purge land first and the line is filled fresh. Walking the sequence:

- data_alias, rep. 0: the line is invalid; the first read misses and fills 17; the second read hits 17. Result `1111`.
- rep. 1: the line holds 17; the first read hits 17 while the purge waits; the purge lands; the second read misses and fills 34. Result `1122`.
- reps. 2–3: the line holds 34; the first read hits 34; then as before. Result `2222`.
- data_purge: the line holds 34; the first read hits 34; the second purge waits behind the second store; the miss waits for both. Result `2222` every time.

The next test, `code_alias`, wrote the instructions `MOV #17,R0; RTS; NOP` through the alias, purged, and jumped to `0202AA00`. The line still held `00000022` from the previous test, so the fetch hit the stale line and read the opcode `0000`, which is illegal. The trap recorded PC `0202AA00` and the laboratory ended.

The manual says as much: a CCR write takes extra time, and instruction reads are served from the cache while it completes (MANUAL 8.4.6). Hitachi's prescribed sequence is to disable the cache, purge, then re-enable, which is what the BIOS does (`10` then `11`). The model predicts that a cache-through read of the same address between the store and the CCR write, which must wait for the write buffer, makes data_alias return `1122` every time and lets the RAM code run. For loading code, an associative purge of the affected lines after such a read is the cheapest safe sequence. The

stock game never executes from work RAM and reads hardware through cache-through aliases, so it is not exposed. A modified program that writes code or tables through one alias and reads them through the other is.

8.5 Silent ends

On a board every failure looks the same: a frozen picture. A CPU exception through vectors 4, 6, 9 or 10 enters Capcom's stubs at `06000418`–`06000460`, which save every register at `02007F30` and stop in `BRA $` at `0600106A`. Capcom's own register-dump screen is still in the ROM behind that loop, unused. The other 114 vectors, including NMI, user break, most `TRAPA` numbers and every unused interrupt level, point into the BIOS at `00000400`. That code disables the cache, copies a routine to on-chip RAM and never returns ROM. A stray interrupt therefore freezes the board without a trace. A custom (dead) cart adds one more case: a program with plain instruction words decodes to garbage at its first fetch from `06000000` and stops there (Section 2.4).

9 A board-cost model, and how close it gets

9.1 What it implements

The measurements above were turned into a timing layer for the SH-2 core of our FBNeo-based emulator. Its version is `20261017`, and it is the default for `sfiii3nr1` in our client. It keeps FBNeo's instruction execution and data handling and adds a timing shadow. That shadow is a small, fully serialised state per CPU, which charges extra clocks for what the instruction table leaves out (Table 10). It also changes the driver's frame loop where the board differs from FBNeo.

Two of these are not bus costs, yet they change every frame. Removing the suspension means the game's pass starts as soon as the VBlank handler ends, every frame, as on the board. Removing the periodic IRQ10 removes 1.33 interrupt entries per frame that the board never takes. Removing it also changed the trajectory: one boot step of the P2 image finished a frame earlier, and the attract mode then chose a different demo. That is how sensitive the attract sequence is to timing.

9.2 What was fitted

The bus-service integers, the handoff deadline and the multiply occupancy were fitted to the forty P1 loops and the 185 P2 loops. On those data the model is a calibration, not a prediction. Thirty-one of the forty P1 loops are within 1% (median absolute error 0.15%, worst 5.4%), and 106 of the 185 P2 loops (median 0.62%, worst 11.8%) MODEL. Figure 9 compares it with build 12, the model as it stood on 14 October, calibrated on P1 alone. The multiply group went from errors as large as 31% to within 0.4%, and the write/read group from errors as large as 19% to within 1% in 9 of 12 loops. Load-use was already right. The RAM read-spacing, copy and FRT-phase groups are still off by 3% at the median and up to 12%, and those residuals are kept, not hidden.

9.3 What was not fitted: the attract demo

The real test is the game. P1 recorded 685 frames of the attract fight from power-on, with no inputs. Running the same probe image from power-on in the model gives the trajectory of Table 11. Nothing in it was tuned to these records.

The model plays the board's demo: same characters, same frames, same graphics loads, same interrupts, same overrun. The attract mode chooses its demo from random numbers drawn by the passes it completes. Agreement on 684 of 685 witnesses and all 685 batch sizes, 4,618 frames after power-on, is the signature of a model that completed the same passes at

Mechanism	Implementation	Evidence
Cache tags	64 sets × 4 ways, six-bit LRU per set; two-way mode uses ways 2–3; CCR purge clears all; associative purge clears one line	P2 cliffs; MANUAL
Fetch	one longword per two instructions; a hit costs nothing extra	P1 B12/B13
Port contention	+1 clock when a memory access is followed by an instruction at a $4n$ address	P1 loads at 1.5
Load use	+1 clock when the next instruction uses the loaded register	P2 load-use
Bus service	BIOS 7 per byte, 112 per line; work RAM 4 per 16-bit transfer (+1 on reads), 36 per line; flash 5 per access, 20 per line; score-screen 18 per word; colour RAM 8; sprite RAM 10; character window 20; other I/O 4	P1, P2
Write buffer	one store in flight; the next bus access waits for it; a RAM read after a RAM store waits for the store plus 3 clocks	P2 spacing, handoff
Multiply	long multiply occupies the unit 4 clocks; its result is ready 5 or 6 clocks after issue, depending on the fetch half; word multiply 2; `STS MACL` waits for the result and interlocks like a load	P2 multiply
FRT access	12 clocks per FRT register access, the measured end of the manual's 3–12	P1 B24/B25
FRT period	compare-match clear holds OCRA one tick: period 13,108	MANUAL Fig. 11.7
Idle	no suspension: the scheduler and slot 5 execute, as on the board	ROM, P1
IRQ10	raised only by DMA completion; the periodic source of Fightcade and upstream FBNeo removed	P1, P2
Frame	$\lfloor 25{,}000{,}000/59.59949 \rfloor = 419{,}466$ clocks; overshoot carried into the next frame's budget only	SRC
DMA	character DMA completes at the kick; palette DMA raises IRQ10 after 2,500 clocks	SRC (unchanged)
Display	draws the sprite list latched by the VBlank DMA, one frame after the pass (Section 11)	ROM

Table 10: The board-cost model `20261017`. All state is plain data, saved in savestates and netplay snapshots. Cache *contents* are not simulated: data reads return memory as in FBNeo.

Quantity, 685 frames paired by frame number	Identical	Model − board, median (IQR)
first live frame	4618 = 4618	
random-number witness `020155E8`	684 / 685	
character-DMA batch size	685 / 685	
flags: live, kick, slot 3 at VBlank	685 / 685	
IRQ10 count	685 / 685	
the one overrun	frame 4678 in both	
last pass return (PASSEND)		−654 (−715 to −587) ticks, −1.8%
pass duration		−463 (−517 to −421)
first dispatch (PASSSTART)		−184 (−263 to −103)
VBlank end		−159 (−184 to −140)
sound driver		−61 (−83 to −45)

Table 11: Model `20261017` against the board on the P1 attract window (MODEL BOARD). The single witness mismatch and the extreme outliers carry the board rows' own OCR status, not a per-field check.

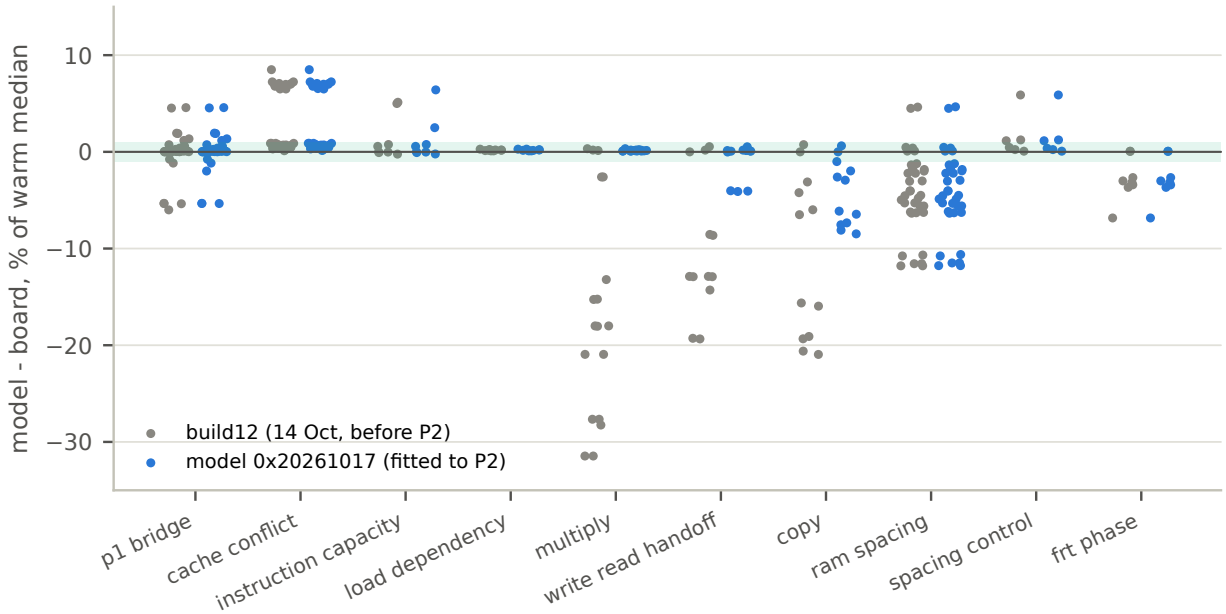


Figure 9: Every P2 CPU loop: model minus board, as a percentage of the board's warm median. Grey: build 12 (calibrated on P1 only). Blue: the current model, which was fitted to these data. The band marks ±1%.

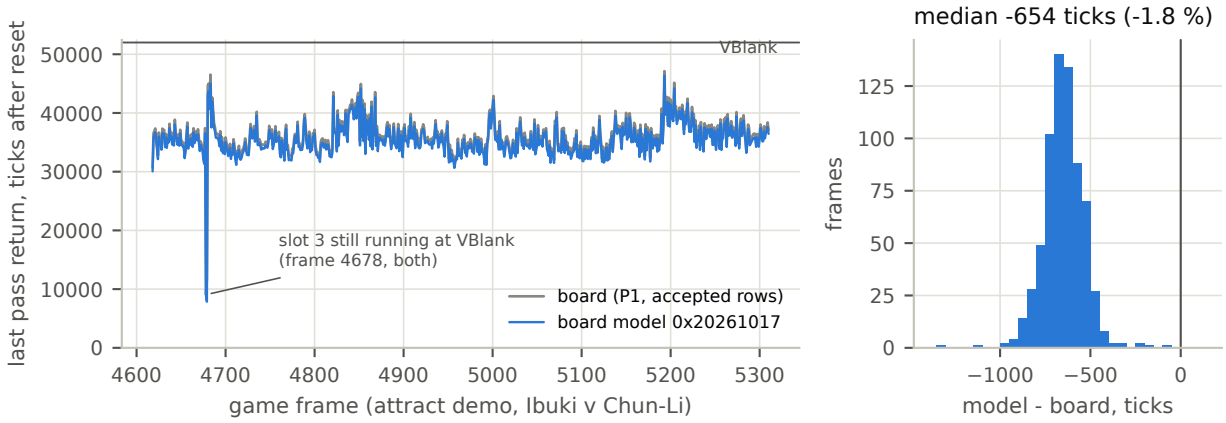


Figure 10: The attract fight, frame by frame: the last pass return on the board (grey) and in the model (blue), and their difference. The overrun at frame 4678, where the logo transition's pass is still running at VBlank, appears in both at the same frame with the same 288-unit DMA batch.

the same VBlanks through the whole boot and attract sequence. Earlier models did not follow the board. The best alignment of a run on Fightcade's schedule found 8 matching witnesses in 224 pairs, and our earlier board-schedule model 39 in 602.

What is left is a nearly constant offset. The model's pass ends 654 ticks (1.8%) early, and the error's slope against pass length is −0.5%, so it is an offset, not a scale error. About 160 ticks of it come from the VBlank handler, which is 2% cheap, and 61 from the sound driver. The IRQ10 service band sits 24 ticks early: the model's interrupt entry is faster than the board's. The pass deficit itself has not yet been attributed. The candidates are the loop groups the model still misses (Figure 9): RAM read spacing, the phase cost of the game's timer reads, and copies.

The deficit shows itself once more in the P2 image. Under build 24, whose spurious IRQ10s added work every frame, the P2 image followed the board into its trace window at frame 4666. Under the current build one boot step near frame 486 finishes a frame earlier than on the board. The attract mode then picks a different demo, and the P2 window starts at frame 4637. The P1 image's trajectory is identical under both builds. A model that is 1.8% fast will occasionally finish, just in time, a pass that the board does not, and in the attract sequence one such frame changes everything after it. While build 24 still followed the board on the P2 image, P2's 80 records paired by frame gave the same picture as P1. The game task's span was 232 ticks (0.9%) short, and the sound driver 56 ticks short.

9.4 From power-on: the board's own counters

P1's first return gave a further test that nobody designed. Its eight photographs show running counters taken minutes apart: armed frames, live frames, the number of frames in which slot 3 was running at VBlank (OVR3), and the frame of the first such event. The probe arms itself at game frame 201, and that offset reproduces the photographs' live-frame counts exactly: `40C`, `412` and `418`, and the legible digits of `?66`. The last photograph's partly obscured `0?08` differs from the model's `0AD8` in one digit and is not used. Board and model can therefore be compared on the same frame base. A read-only copy of the probe's current record, taken every frame in a fresh model run, gives every frame's margin without changing the trajectory: its census equals the reference run on all 9,001 rows.

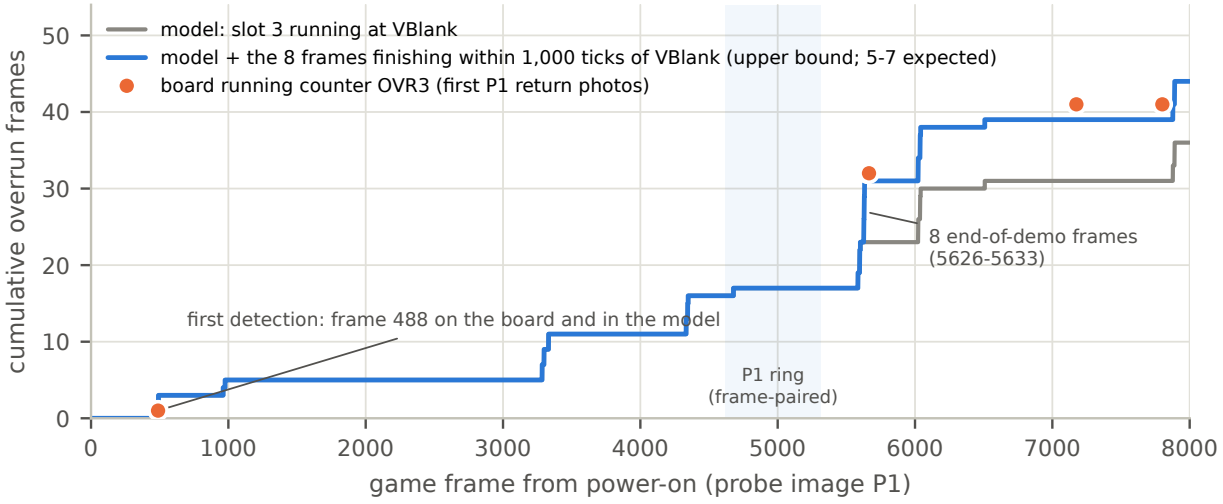


Figure 11: Cumulative overruns from power-on. Grey: frames in which the model's game task is still running at VBlank. Blue: the same, plus all eight model frames that finish within 1,000 ticks of VBlank, an upper bound (the model's measured deficit turns five to seven of them into overruns). Orange: the board's running counter in the first-return photographs, and its first-overrun frame.

The first overrun is on frame 488 on the board and in the model. It comes five frames before the boot sequence's first 1.95 MB graphics load. By game frame 5,653 the board had counted 32 overruns and the model 23. Most of the difference sits in one place. Frames 5626–5633 are the end of the attract demo, where each pass runs about 43,500 ticks against a 52,000-tick frame. In the model those eight frames finish only 238 to 758 ticks before their deadline, 0.5–1.5% of the frame. The model's own measured deficit is 654 ticks at the median, with an interquartile range of 587–715, a little larger on long passes. Applying that measured distribution frame by frame turns five to seven of the eight into overruns on the board. That gives 28–30 against the board's 32 at frame 5,653. Between frames 5,665 and 7,175 the board adds nine overruns and the model eight. The remaining two to four frames are not yet accounted for. The long transfers after the 1.95 MB loads, whose duration and bus effect the model ignores, are the first place to look. **From power-on, the model gets the first overrun's frame exactly and accounts for all but a handful of the board's overruns.** It also shows exactly where its remaining deficit matters: in runs of passes that each use 98–99% of the frame.

10 Where the board drops frames

10.1 The census

With a model that follows the board's own trajectory, the question of where the frame goes can be asked of the board's schedule rather than Fightcade's. We ran the model over the project's replay corpus: 99 episodes, 129,310 frames, about 36 minutes. There are 32 segments of recorded Fightcade matches (1,800 frames each), 55 CPU matches started from training states, 10 generated input sequences and 2 fixtures around a KO. After every frame, at the instant IRQ12 is asserted and before the handler runs, a read-only probe samples the eight task states and the match context. **Slot 3 still running at that instant is a lost frame**: the same definition as the board probe's flag MODEL.

Each episode starts from a saved state converted to the model's own format, with an empty cache. That is a research initial condition, not replay compatibility. No overrun occurred in the first ten frames of any episode. As a control, the same corpus ran with FBNeo's instruction-table timing and the whole frame available, which is the schedule MAME uses and our earlier releases used.

	board model `20261017`		instruction table, whole frame	
State	frames	lost	frames	lost
live fight (game running, timer counting)	90,183	**1**	90,818	0
round intro	10,784	21	11,899	1
round end (KO, wipe, load)	8,790	114	8,402	24
before FIGHT	1,700	34	585	4
result screens	710	24	666	0
character select, continue, demo (G_No 0)	10,403	55	10,528	13
other	6,740	7	6,412	1
all	129,310	256	129,310	43

Table 12: Lost frames by game state (MODEL). Frame counts differ between the columns because the two timings reach transitions at different frames. Earlier model versions (`20261013`, `20261015`) give the same transition structure; the per-episode counts of `20261015` and `20261017` are identical.

10.2 Live play: one frame in 90,183

In the live-fight state the board model lost one frame in 90,183. It happened in a CPU match, Alex against Akuma on Akuma's stage, round 1, with 96 on the timer. The VBlank kicked a 22 KB character batch, the handler ended at 9,137 ticks, the game task was dispatched at 9,562, and it was still running at the next VBlank. It finished 725 ticks into the following frame.

How close live play comes to the deadline is shown in Figure 12. In the probe-image run of the same corpus, the margin at VBlank has a median of 20,960 ticks (40% of the frame), a 1st percentile of 13,246 and a 0.1st percentile of 8,393. Six of 90,161 completed passes came within 2,000 ticks, and the closest within 90. **Every one of the tight frames carries a 14–31 KB character-DMA batch and a VBlank handler of 7,700–9,700 ticks**, and its pass runs 40,000–42,500 ticks, nearly twice the typical live pass. Live slowdown on the board is a coincidence of a long pass with a large graphics load, not one heavy effect. Allowing for the model's 1.8% deficit, the 90-tick and 636-tick frames would also be lost on the cabinet. That makes one to three lost frames in 25 minutes of live play.

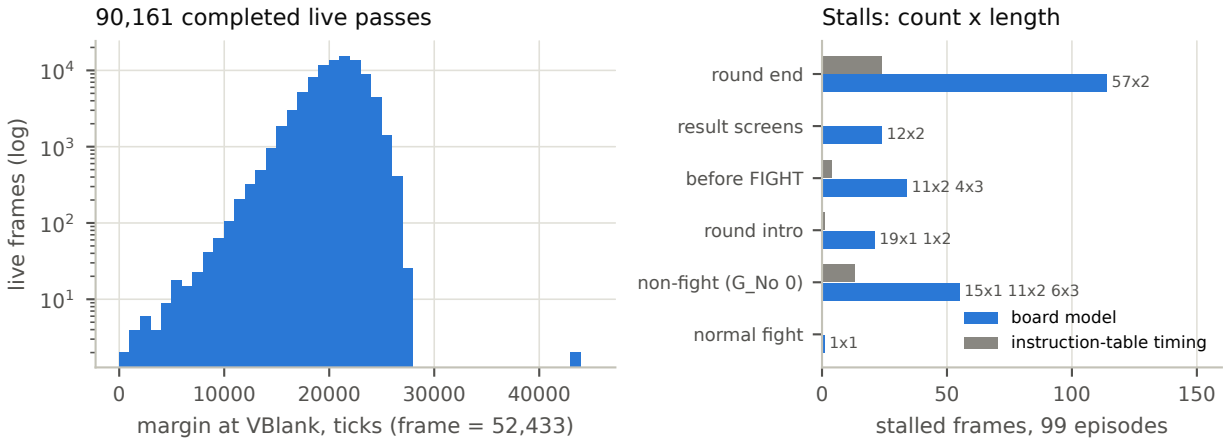


Figure 12: Left: margin at VBlank for every completed live pass in the corpus (log count). Right: stalled frames by game state, board model against instruction-table timing, labelled count × length. A stall of length 2 is the game task spanning two VBlanks, so the picture repeats twice.

10.3 Transitions: five frames at every round end

The board's lost frames are transitions, and they are strikingly regular. Every round end in the recorded matches has the same three stalls MODEL. All 14 segments that contain a whole round end show them, and a fifteenth, which starts inside the transition, shows the last two in place:

1. two frames when the screen wipe starts (frame t);
2. two frames when the next round's graphics load, eleven frames later ($t + 11$);
3. one frame when the round initialises ($t + 15$).

That is five lost frames, 84 ms, at every KO. With the instruction table and the whole frame, the same transitions lose one frame, at the load ($t + 9$ on that trajectory). Under Fightcade's rotating windows the same transitions lose one or two single frames, which ones depending on phase [1]. Round intros, the screens before FIGHT and the result screens add more stalls of one to three frames on the board model, and almost none without it (Table 12). The structure does not depend on the last model revision. The earlier model `20261013` lost 247 frames on the same

corpus with the same pattern, and only 5 of the 99 episodes followed a different trajectory under it than under `20261015`.

10.4 The round wipe, on the board

On Fightcade's schedule the clearest phase-dependent case is the entry and round wipe, where whether a frame is lost depends on the phase of the rotation [1]. Table 13 puts the same passes, in the same three episodes, under the board model.

Episode	Pass	Fightcade schedule [1]	Board model
`rp_..8844`	wipe start	frame 177: overrun (one-slice window)	177–178: **two-frame stall**
	load after wipe-out	187: overrun (two-slice)	188–189: **two-frame stall**
	round initialisation	190–191: fits, margin $\geq$ 47,212 cycles	192: **one-frame stall**
	wipe-in, 8 frames	fits; one-slice margins 11,141–16,841 cycles	fits; margins $\geq$ 14,500 ticks
`rp_..3058`	wipe start	113: fits the three-slice window by 9,949 cycles	113–114: **two-frame stall**
	load after wipe-out	122: overrun	124–125: **two-frame stall**
	round initialisation	124–125: fits	128: **one-frame stall**
	wipe-in, 8 frames	fits; one-slice margins 2,803–13,229 cycles	fits; margins $\geq$ 12,500 ticks
`chunli_q_ko`	wipe start	308 fits (three-slice), 309 overruns (one-slice)	308–309: **two-frame stall**
	wipe-in, 8 frames	fits; one-slice margins 7,669–11,232 cycles	fits; margins $\geq$ 12,200 ticks

Table 13: The entry and round wipe under Fightcade's schedule (Table 3 of [1]) and on the board model (MODEL). Fightcade margins are in emulator cycles; board-model margins in timer ticks (8 clocks). Frame numbers shift after the first stall on the board model.

On the board the wipe start loses two frames every time, whatever the phase. Its pass took 278,000–302,000 cycles on Fightcade's instruction table. On the board model it spans two VBlanks, so it needs at least the 97,600 ticks (780,000 clocks) left in the two frames after its dispatch. The wipe-in frames, which were Fightcade's nine tightest margins and the frames where small code differences were expected to flip decisions [1], are comfortable on the board, with at least 12,000 ticks (23% of the frame) to spare. **The frames that decide slowdown on Fightcade and the frames that decide it on the board are different frames.**

10.5 What a player sees

On a cabinet, a round of 3rd Strike plays without dropped frames, except for a rare single frame when a long pass meets a large graphics load. Every KO is followed by five lost frames in three hitches. The attract demo hitches at its logo transition (frame 4678 in the board's recording) and, on the board, through most of eight frames near its end. Fightcade and upstream FBNeo drop one frame at most KOs, sometimes two, at points chosen by their rotation. An instruction-table model with the whole frame, scheduled as MAME schedules it, drops one.

Section 3.3 showed what a lost frame costs an input. The system screens lose a press whose first sampled frame is lost; the fight acts on a held press one frame late and loses only what existed

for that frame alone. On the board the fight loses a frame about once every eight to twenty-five minutes of play, and the transitions, where the five frames go, read no attacks. **Dropped-frame input loss exists on the cabinet, but it is not something a player meets in a match.**

11 The display pipeline

Slowdown and latency are separate questions, but the second one is easy to get wrong by a frame, so here is the board's display pipeline in the same terms.

The VBlank handler samples the inputs at VBlank N. The game pass that follows (pass N) reads them and writes its sprite list into sprite RAM. At VBlank $N{+}1$ the handler triggers the sprite-list DMA (`0613696C`), which copies that list to the video hardware ROM. The CPS3 composes its picture with full-screen zoom, and both the JTCPS3 implementation and MAME's driver render into a buffer before scan-out [3, 4]. A frame-buffer board therefore renders pass N's picture during frame $N{+}1$ and scans it out during frame $N{+}2$. A press that lands at a uniformly random time before VBlank N is visible after $0.5 + 2 + s$ frames, where s is the fraction of the frame the beam needs to reach the point being watched.

Pipeline	pass N visible from	press to light
board, frame-buffer renderer	frame $N{+}2$	$2.5 + s$
board, if it rendered lines on the fly	frame $N{+}1$	$1.5 + s$
emulator drawing the latched list (upstream FBNeo, MAME, our default)	frame $N{+}2$	$2.5 + s$
emulator drawing live sprite RAM (Fightcade's source default)	frame $N{+}1$	$1.5 + s$
FPGA core rendering through a frame buffer [4]	frame $N{+}2$	$2.5 + s$
FPGA core rendering through a line buffer [9]	frame $N{+}1$	$1.5 + s$

Table 14: Press-to-light accounting in frames, for a press at a random phase. The emulator rows exclude the host's presentation chain (render queue, operating system, monitor).

WydD's camera measurement of a real CPS3 gives 2.58 frames (43.2 ms), with the input read during blanking [10]. That is $2.5 + s$ with the sensor near the top of the picture. A line renderer could reach at most 2.5, and an extra buffered frame would give at least 3.5. **The board is a frame-buffer pipeline: the picture of the pass that read a press appears two VBlanks after the VBlank that sampled it.** Our emulator's default draws the latched list and matches this. In a counterfactual test, removing a press changes the rendered buffer first two frames later, and one frame later with runahead 1 or with live sprites. Fightcade's source draws live sprite RAM by default [7], but the installed client measured the same as the latched pipeline in that test, so its row is unresolved. None of this measures the host's own presentation chain, which is where any remaining difference between an emulator and a cabinet now lies.

12 Earlier conclusions, revisited

The earlier findings about Fightcade stand [1]: the idle shortcut, the rotation of one-, two- and three-slice windows, the role of the periodic IRQ10 in waking the CPU, and the dependence of its slowdown on phase. What changes is everything that work, and the earlier report behind it [11], could only say about the board by inference.

- **"Ordinary play is far from every deadline."** Still true, but not by the margin the emulator suggested. A live pass takes a median of 23,000 ticks on the board model, 44% of

the frame, not 12%. The VBlank handler takes another 15%. Ordinary play keeps 40% of the frame in reserve at the median and 0.17% at the extreme.

- **"Overruns come from transitions, not combat."** Confirmed, with different numbers: five lost frames at every round end on the board, against one or two on Fightcade, plus about one live frame in 90,000.
- **"Near-deadline frames are rare and clustered"** in the wipe-in. On the board the wipe-in frames have at least 23% of the frame to spare. The board's near-deadline frames are live passes that coincide with large graphics loads, and runs of heavy passes such as the end of the attract demo.
- **The free-running timer is "gameplay-inert".** True of its logic. On the board its handler takes 21 μs in 20–29% of frames and reorders DMA service, and its compare position encodes the frame length to a few ticks.
- **The 26-cycle claim in `move_effect_work`.** On Fightcade's schedule a 26-cycle difference can flip a one-slice frame. On the board it is 3 ticks, and in the whole census only one live frame came within 100 ticks of its deadline. The transitions overrun by tens of thousands of clocks. On the board, a cycle count matters only as part of a percent-level cost difference, and the biggest source of those is where code and data sit in the cache (36 clocks per conflicting load). A rebuilt program that wants board timing must keep the original's addresses, not only its instructions.
- **"The board's per-frame cost depends on wait states, cache behaviour and DMA contention that no cycle table here models."** Now measured, except for contention, which turns out not to be needed at live batch sizes.

13 What a board-exact CPS3 has to reproduce

The list below collects, in order of their effect on which frames are lost, the behaviours a machine must have to drop the frames the board drops and fail where the board fails. Each item names the measurement it rests on.

1. **No idle suspension.** The scheduler and its idle dispatcher run as code. They swap DMA queues on every scan and read task states. The pass begins as soon as the VBlank handler ends (Sections 3, 3.4).
2. **IRQ10 only on DMA completion.** One interrupt per finished batch, none otherwise (Section 7).
3. **Bus costs per device, width and direction**, as in Tables 3 and 6: work-RAM lines 36 clocks, flash lines 20, BIOS lines 112 on an 8-bit bus, character window 20 per word, score screen 17–19 per word, colour RAM 7–8, sprite RAM 10 per word, ports 5, FRT registers 12 (Section 5).
4. **The cache as a cache**: 64 sets of 4 ways, index bits 9–4, LRU, write-through without allocation, a two-way mode that keeps ways 2–3, purge by CCR and by associative writes. A conflicting load costs a line fill (Section 5.4).
5. **The pipeline's overlaps**: a write buffer that lets independent instructions run behind a store, reads that wait for a pending write, fetch/memory port contention, the load-use interlock, and multiply issue separate from multiply result (Section 5.5).

6. **The timer exactly**: clock/8, period OCRA + 1, a compare flag that survives a counter write, priority 15 above the external interrupts, TEMP-latched access, and phase-dependent access cost (Sections 2.3, 6).

7. **Interrupt acceptance**: masked handlers, the one-instruction inhibit after `LDC`/`LDS`-class instructions, and no interrupt inside a bus cycle (Sections 2.3, 8.3).

8. **The clocks**: 25 MHz CPU and 59.5995 Hz display from separate crystals, nominally 419,467 clocks per frame; one measured cabinet has 419,512 ± 40 (Section 6.3).

9. **DMA**: the kick's CPU cost through the character window, a decompressed output rate of at least 8.5 bytes per microsecond, and completion signalled by IRQ10 when the engine finishes, so a 2 MB load takes frames rather than nothing (Section 7).

10. **Functional details that only fail on hardware**: an 8-bit score-screen RAM with tile bit 8 in the attribute; cache contents that can go stale under alias writes, with purges ordered behind pending stores; accesses to unpopulated addresses that never complete; exceptions and stray vectors that end silently; and, on dead carts, decryption of instruction fetches only (Section 8).

11. **The display**: the sprite list latched at VBlank and shown from a frame buffer, two VBlanks after the inputs were sampled (Section 11).

Items 1 and 2 decide whether the transitions behave like the board at all. Items 3–5 decide how many frames each transition loses and whether rare live frames are lost. Items 6–8 decide the second-order signatures, such as the stale-compare share and the interrupt bands. Item 10 decides whether a modified program survives on real hardware. The census of Section 10 needs items 1–5. Our model implements items 1–6, 8 and 11, without cache data or the timer's access phase. It implements item 9 without transfer time, and item 7 only as far as FBNeo's SH-2 core does. It does not model the hardware failures of item 10.

13.1 Where other implementations stand

Table 15 sets the implementations we could read against the list, from their source at the revisions in Appendix A. It records what each one does, not how well it plays; "not compared" marks machinery whose calibration against the board has not been run.

Three things in the table deserve a sentence each.

The FPGA cores have the machinery; what is untested is its calibration. Their CPU model is register-transfer logic with a real four-way cache, a bus state controller that honours the programmed wait states and samples the WAIT pin, a one-deep posted store, a multiplier that stays busy for four clocks after a long multiply, a timer with the manual's semantics and the pipeline's interlocks. Every cost in Section 5 is therefore a question that model can be asked in simulation, without a board: the forty P1 loop bodies with their access contracts, and the 185 P2 loops, are in the companion data with the board's ticks beside them. The line fills (36, 20 and 112 clocks), the flat write-then-read pairs and the multiply table are the three results that would show at once whether an FPGA's memory path costs what the board's does. The differences the table does show are on the board side. JTCPS3 times its display with the CPS1 module, 512 × 262 lines at an 8 MHz pixel enable, rather than with the 455 × 264 the game programs into the CRTC; its register-driven timer exists in the tree but is not connected. The second core raises IRQ10 on MAME's 100 μs placeholder rather than on its engine's completion, and renders through a line buffer, so pass N reaches its screen one frame earlier than on the cabinet of Section 11.

The CPU-to-display ratio is testable without any instrument. The stale-compare share of Section 6.3 depends on where the fourth timer compare lands relative to the VBlank handler's

	MAME	FBNeo	Fightcade	JTCPS3	Second FPGA core	This model
1. Idle	runs the code	shortcut on the task-0 read (on by default)	shortcut	runs the code	runs the code	runs the code
2. IRQ10	on DMA, 100 μs after the request	on DMA, and every third slice	same	on the engine's completion	100 μs after the request; engine separate	on DMA only
3. Bus costs	one clock per access	same	same	programmed WCR/BCR1 waits plus the memory controllers' WAIT; not compared with the board	same model	fitted per-device integers
4. Cache	none	none	none	full cache in the CPU model: tags, data, LRU, two-way mode, both purges	same	tags and LRU only
5. Pipeline	none	none	none	issue and load-use interlocks; one posted store (the next external access waits for its bus cycle); multiplier busy 4 clocks after a long multiply, 1 after a word multiply; a result read waits for it	same	write buffer, port contention, load use, multiply states
6. Timer	period OCRA; a flag clears only after a read as set	period OCRA	period OCRA	OCRA + 1, TEMP latch, flag survives a counter write	same	OCRA + 1
8. Clocks per frame	419,467 from the crystals	419,466	416,666 (25 MHz over 60 Hz) at a 59.59949 Hz refresh	25 MHz CPU; display from the CPS1 timing module, 512 × 262 lines at an 8 MHz pixel enable: 59.637 Hz, 419,200; the game's CRTC values decoded but unused	25 MHz exactly; 59.5637 Hz as shipped: 419,719	419,466 with carry
9. DMA completion	instant; IRQ after 100 μs	instant; palette after 2,500 clocks	instant	real engine, rate set by the SDRAM	real engine, decoupled from the IRQ	instant; palette after 2,500 clocks
10. Score-screen RAM	byte-wide; tile bit 8 from the attribute	32-bit cells; bit 8 from the even byte	same	byte lane	byte-wide	as FBNeo
11. Display	render buffer: pass N in frame N+2	latched list: N+2	live sprite RAM by default: N+1	frame buffer: N+2	line buffer: N+1	latched list: N+2

Table 15: The numbered behaviours of this section in six implementations (SRC: [3, 4, 7, 8, 9, 12, 13, 14]). Item 7, interrupt acceptance, is not compared. JTCPS3 is read at `bfced64d`; a multi-platform port of the same HDL [15] pins `70884b53`. The two FPGA cores share one CPU model, Sergiy Dvodnenko's (srg320) SH7604 as carried in JTFRAME, so their CPU-side rows are the same.

mask, to about ten ticks per frame. A core whose display runs at 59.5637 Hz gives the CPU 419,719 clocks per frame, 26 ticks more than the measured board; its fourth compare lands before the mask in nearly every frame, so the stale share falls to zero and the case-B shift appears in nearly every record. The P1 image run on such a core shows this in its ring without any alignment to the board. JTCPS3 sits on the other side: 419,200 clocks per frame is 39 ticks less than the board and 32 ticks short of four compare periods, so its fourth compare lands after the reset and is never taken, as in an emulator whose frame budget is 416,666 clocks. Both cores miss the measured ratio by more than the nominal crystals do, in opposite directions, and P1's period field shows either in one run.

For the software emulators the list is short. MAME has the right interrupt source, the right score-screen RAM and a timer flag that survives a counter write, and lacks every cost. FBNeo and Fightcade add the idle shortcut and the periodic IRQ10 that decide their slowdown (Section 1); following the board's trajectory needs items 1 and 2 first, then 3–5. Our model implements items 1–6, 8 and 11 as Section 9 describes, and inherits FBNeo's score-screen cells.

14 The next measurements

Each open question above has a definite experiment and, now, a definite expected answer. A third probe should carry them in this order:

1. **Fault-tolerant laboratory.** Per-test checkpoints and a trampoline, so that one exception costs one test. Code-execution tests run last. Every code test reads its code back through the cache-through alias and reports both full longwords.

2. **Purge ordering.** The coherency sequence of Section 8.4 with a same-address cache-through read inserted between store and purge. Expected: `1122` in every repetition, and the RAM code runs. Add an associative-purge variant and a control executing code from flash.

3. **Free-running VBlank series.** Sixty-four consecutive IRQ12 timestamps with the game's timer reset disabled. Expected: $419{,}512 \pm 40$ clocks per frame. A result outside 419,400–419,620 would contradict Section 6.3.

4. **Timer semantics.** TEMP ordering, rollover, compare/clear races and the compare period. Expected: period 13,108, and a flag that survives a counter write.

5. **DMA brackets.** Last-busy and first-clear timestamps for raw and compressed character batches of several sizes, palette loads, and one transition-sized batch. Expected: at least 8.5 output bytes per microsecond. This gives the model its missing transfer time.

6. **Display layers.** At known VBlanks, alternate a palette flash, a sprite block and a score-screen block, filmed at 240 frames per second. This tells which layers are latched together and settles Table 14 without lab equipment.

7. **A reference clock in the shot**, which turns the absolute-clock result into a 30 ppm measurement.

8. **The same probe on a custom (dead) cart**, to see whether Capcom's decrypting CPU adds fetch time.

9. **The forty loops on an FPGA core.** The P1 and P2 loop bodies run in simulation on the SH7604 model the FPGA cores share, against the board's ticks (Appendix D, Table 20). This needs no board, and it is the quickest way to learn whether an FPGA's memory path costs what the board's does. The display cadence needs no simulation at all: Table 15 gives each core's clocks per frame against $419{,}512 \pm 40$.

Two more checks need no probe at all. The first is a 60-frame-per-second recording of a cabinet's attract demo from power-on: the model predicts its hitch frames, including the logo transition and the end of the demo. Recordings from a second cabinet would add a test of their own. One marginal boot frame decided which demo the model played (Section 9), so two cabinets whose crystal ratios differ by enough to move such a frame may play different demos from power-on; whether the attract sequence is such a fingerprint is a question a few recordings would answer. The second is any recording of a KO on a cabinet, which should show five held frames in three groups.

15 Limitations

- **One board.** Every physical number comes from one CPS3 main board, with one Darksoft kit and one cart carrying a standard SH7604-family CPU. Costs that belong to the main board's chips (bus, memories, DMA, video) should hold for other boards. CPU-side behaviour is Hitachi's part, not Capcom's custom CPU. The CPU-to-display ratio belongs to this cart in this board.
- **Instrumented programs.** The probes measure themselves along with the game. Their costs are identified (Section 4), and the conclusions drawn here survive them. Still, the board was never recorded running the unmodified program, and P2 has no reduced-instrumentation control boot.
- **What the laboratory lost.** P2 stopped at test 187. The timer-semantic, DMA-bracket and free-running VBlank tests that would have measured the timer period, the DMA rate and the frame ratio directly did not run. The values given here are derived from the live data instead, and each says so.
- **A fitted model.** The model's bus costs were fitted to the same board's loops. Its tests against the game (the attract window and the power-on counters) were not fitted, but they come from the same board and the same attract fight. The census is the model's prediction, not a board recording. Its live-play conclusions are robust to the model's 1.8% deficit, and its transition structure did not change between three model revisions.
- **Census coverage.** 36 minutes of play, 25 of them live, from replays, CPU matches and generated inputs. No scripted stress scenes (many effects on graphics-heavy stages) were run, so a rarer heavy case may exist.
- **Not modelled.** Cache data (only tags are timed), DMA transfer time, the timer's access phase, any DMA/CPU bus contention beyond the live batch sizes, DRAM refresh, and the remaining 1.8% of the game pass.
- **Clocks.** The frame ratio rests on a calibrated estimator and the manual's compare period. The absolute clock rests on an uncalibrated phone and a modelled draw cost.

16 Conclusion

In Fightcade's emulation an idle shortcut decides when the game may run [1]. On the board nothing decides that: the CPU runs the game's work as soon as VBlank is done and spins through the scheduler until the next one. What decides where a frame goes on the board is what the work costs. Memory is where the cost lives. A 36-clock line fill from work RAM, 20 clocks a word to character RAM, an 8-bit BIOS, a four-way cache in which one line too many costs 36 clocks a load, and a write buffer that hides some instructions and not others together make the game's pass four times what an instruction table says. Half of the VBlank handler's time goes to the sound driver.

With those costs, two interrupt rules (IRQ10 only on DMA completion, the timer at priority 15 with a compare that survives the reset) and no idle suspension, a model follows the board's attract demo frame for frame. It finds the board's first slowdown on the board's frame, and it predicts where 3rd Strike drops frames on a cabinet. That is almost never in a fight. At every KO it is five frames, in the same three places every time. The physical CPS3 also fails in ways no emulator does: it hangs on a read nobody answers, keeps only odd bytes of its text layer, and lets a cache purge wait behind a store. The CPU-side numbers are Hitachi's, not Capcom's, and hold for any machine built on the same part. Anyone changing the program, or building the machine, now has measured numbers for each of these, the data to check them against, and a short list of experiments that would close the rest.

Credits

Board research: XeD. XeD ran all six diagnostic builds and the repaired program on their own CPS3 and filmed every return. They reported what the board did between builds, and explained how a Darksoft kit writes the program SIMMs and how to tell standard carts from custom ones. Every physical result in this paper comes from that work.

Probes, analysis, model and writing: J.R.Dunkley. The community tools and references cited below made the rest possible. The routine names in Appendix C are those of the sfIII3-cps3-decomp project [16], whose source we used to confirm which routines consume the input edges.

A Identity

Artifact	SHA-256 (or revision)
Program `sfiii3nr1`, 990512 NO CD rev1, decrypted	`35ac2d569893926599b0f68d772e98f4 12573fb76b305ba3ed910f6136dd2de0`
P1 build 09 as run on the board (Darksoft region literals)	`263356f46d86bc8816f839e94324f8a9 631c73b147e98fc7839878eaac980ffc`
P1 build 09 as run in the model (standard region literals; differs in 3 words)	`a2a34c47160e59f6d5ccab0f2481aff4 d6e602d8e57094e5cdd9526b3bc98022`
P2 MAIN image, build `50320002`	`652a1f23927f4ac237de44aefece9c05 d8a44b621b92524fed9c38662a829faf`
P2 CONTROL image (not run on the board)	`621b369215f89a0903d8205388f5b9f9 e832bc0c93dec5a8d337a0c6970e8a68`
P1 second-return video	`2b779e225434560c1bb8e0ce598cf603 7005ae98365d486fc4271222903ca047`
P2 return video	`2650ae3a85e9266c447cf2f0efbde80e 2815e28455d057f3b2a202c594211ef7`
P2 recovered capture (16,128 bytes, CRC-32 `881D2B40`)	`821dcbf85b981a6cb1116b0c649d97c8 7875a6026f00a6622c8292cee21dc983`
Model build 26 (`20261017`), x86	`8b1bff0518b2bc5c4d9d47f38748798b 9a79fc191eb11396059ae27a4a7820ce`
Model build 26, x64	`c2b7357e1df6a82c7a99797676f3abd0 c9a45ef90e8bf4248557b380d559d52e`
Model timing header `cps3_timing.h`	`5a6e8f7956ea8a7229946565339f55d5 ad5a92dce80c23b773da3229b27ecb37`

Artifact	SHA-256 (or revision)
Model build 24 (20261015), census	05629a0d10e9bc549e56158f7afa75b4 0e74b88a166eb139a4e6bddebcab1bd4
Model build 12 (20261013; legacy control with timing off)	93083715ce219f2350fd04f8a414867a 7ee75267fb75faa34afc8e0d7e977814
SH7604 hardware manual (PDF)	262cfff2abec2fa0cef5c5475495d6a4 da390eff107ed3a75575827467daab9f
Fightcade FBNeo / upstream FBNeo / MAME / jtcores	c95950140e24 / 2bac03411897 / 8b80cfd15d79 / 70884b5347d9
Read for Section 13.1 on 16 October 2026: MAME / upstream FBNeo / jtcores / second FPGA core / JTCPS3 port / decompilation	41a80b9dbaed / 16484e0d941b / bfced64dad83 / 923d1ba3a60d / 2bc038eeb3dd / 34e22e7138a1
jtcores at bfced64dad83: cores/cps1/hdl/ jtcps1_timing.v, the display timing JTCPS3 instantiates (outside its cores/cps3 tree)	3cb548409d8e9f508a0f5aabe3f666bb 1191df6e294577b7a3762e47e83bc665
jtcores at bfced64dad83: modules/jtframe/src/ jtframe/mem/clocks.go, the clock-enable generator (output order)	26a7e9edf10a6ca152adafaea1d44e3a 363abf47ade9722abf67f57b61ed6f56

No game program, BIOS or probe image is distributed with this paper. The probe images can be rebuilt byte for byte from the probe sources and a separately owned copy of the program.

B Companion data

The companion archive (where_the_frame_goes_board_companion) contains every board observation used here, the model outputs they are compared with, the numbers and figures of this paper, and the scripts that produce them. Board data and model data are kept in separate directories. MANIFEST.json and SHA256SUMS.txt bind every file.

Path	Contents
board/darksoft_diagnostics.json	D1–D4 board readings: boot lines, SIMM sums, frozen PC/PR lines, breadcrumbs, as photographed
board/p1_first_return/	transcribed status rows and summary counters of the eight first-return photographs
board/p1_second_return/	all 700 ring slots with status (accepted, unresolved, unfilmed), the 40 loop readings with every raw view, the loop access contracts, the analysis and its verification
board/p2/	the recovered capture, every validated row, single-frame rows, row votes, parity audit, the typed decode (live records, events, CPU samples, coherency values), the probe's format description and decoder, and the fresh-build audit
model/attract_probe_b26/	the P1 image from power-on in model 20261017: per-frame census and the 700-record ring

Path	Contents
model/attract_margin_b26/	the same run with a read-only per-frame copy of the probe's current record
model/attract_probe_b24/	the P1 image in model 20261015, used in the estimator self-test
model/fightcade_schedule_b09/	the P1 image on Fightcade's schedule (native machine)
model/p1_loops_b26.txt, model/p2_core_b26.json, model/p2_build12.json	model results for the P1 and P2 loops
model/census/	per-episode, per-frame task states and match context for the 99-episode corpus: model 20261017, model 20261015 with the probe image (with records), and instruction-table timing
numbers/	every number in this paper as JSON and CSV, produced by scripts/paper_numbers.py and scripts/frame_period.py
figures/, paper/	the figures (PDF, PNG) and this paper's source and PDF
scripts/verify.py	recomputes the paper's headline numbers from the archive's own files, with the Python standard library, and compares them with numbers/

The census inputs (saved states, replays and the game program) are not included. The per-frame outputs are.

C Addresses used in this paper

The third column gives the routine's name in the sfIII3-cps3-decomp project at 34e22e71 [16]; "in" marks an address inside the named routine, and "provisional" is that project's own marking. The sprite_poly family names a polygon queue, as the PS2 port has one; on the arcade the same queue feeds the character and palette DMA (Sections 3 and 7).

Address	Meaning	Name in the decompilation
06000528	VBlank (IRQ12) handler; masks at 06000576, resets FRC at 0600058E–06000596, calls the sound driver at 060006F8, returns at 0600071A	vblank_frame_dispatch
060004D0	IRQ10 (DMA complete) vector; mask 15 at 060004F2; handler 061380E4, busy-wait at 06138100 while 240C000C & 6	sprite_poly_queue_flush; sprite_poly_queue_drain_wait
0600073E	FRT compare-A handler (priority 15, set at 06000EFE–06000F0A); switch debounce 061332D4	frt_compare_intr; in reset_entry; coin_sw_sample
06000860, 06000882, 0600094C	scheduler, its task-0 state read, the re-arm routine	in Game_Task; task_sleep_tick
06000CDC, 06000D28, 06000D8C, 06000DC2/DC6	dispatch (resumes with LDC R0,SR in the delay slot of the JMP at 06000D1C), return, yield, resume (first interrupt-eligible instruction)	task_dispatch, task_sleep, task_wait
06004E12	slot 5, the idle request dispatcher	in boot_task
0612DDCC	the eight layers' positions written to the video registers, from the VBlank handler	Irl_Family
0613809A, 06138168	DMA queue kick in VBlank; DMA start (three longwords per command through the character window)	sprite_poly_queue_drain_try; sprite_polygon_flush_queue

Address	Meaning	Name in the decompilation
06137C14, 06137DF0	DMA queue swap on every scheduler scan; palette request (starts at once when idle, through 040C00A0–AE)	poly_bank_flip; polygon2d_submit_quad (provisional)
0613696C, 06136996	sprite-list DMA trigger; wait for it (loop at 061369AC)	dma_src_ack_seq; sprite_dma_end_wait
06132FD8, 06132E60	VBlank input readers (one previous sample per player)	switch_read_six_button; in switch_read
06001ADE, 060044AC, 06091BEA	consumers of the readers' one-frame edge: start button, break-in check, scene skip	in Entry_01; Ck_Break_Into; Button_Cut_Hold (provisional)
061195E0, 060B59BE, 060B55F8	per-pass copy of the current word into the command work; the command checker and its own previous-pass word	Player_move; sw_pick_up; pl_lvr_set
0613110E, 06131120	bus set-up: CCR 10, BCR1 0020, WCR AA57; then CCR 11	sram_bus_normal; —
06005C00, 0613E380	CRTC programming: eight words per table row to 240C0060–66 and 240C0070–76; horizontal total 454 and vertical total 262 in all nine rows	screen_window_regs_update
060D5E18	score-screen cell writer (tile bit 8 into attribute bit 0)	tilemap_put_cell
06000418–06000460, 0600106A	exception stubs; the silent stop	exc_illegal_instr to exc_dma_addr_err; in cpu_hang_forever
00000D04, 00000400	BIOS score-screen RAM test (mask 00FF00FF); BIOS target of 114 unused vectors	(BIOS)
0200D794, 0200D854	task contexts (64 bytes each); slot 3, the game task	
02000000, 02007F00	FRT tick counter; VBlank frame counter	
020155E8, 02011389	random-number index (the probes' witness); live-play flag	
0207A800–0207EFFF, 0207F000	probes' private RAM; the Darksoft kit's region bytes	
06700000–0670FFFF	probes' private code (zero in the original program)	
06ED7900, 02028990	the modified program's freeze hook; the effect pool (128 slots of 800)	(not in the original)
240C000C, 25000000, 25040000	DMA status, inputs and score-screen RAM through the cache-through aliases	
FFFFFE92	CCR (cache control)	

D The forty P1 loops

Board ticks are the reviewed P1 readings (B23 has two readings three ticks apart). Instruction-table ticks are the cycles FBNeo's cost table charges for the same instruction trace, divided by eight. Model ticks are the current model on the same bodies. N is the iteration count; the body of each loop and its access trace are in the companion data (benchmark_contracts.json).

ID	Loop	N	Board	Table	Board/table	Model	Model error
B00	empty	1,024	527	512.6	1.03	528	+0.19%
B01	alu8	1,024	1544	1536.6	1.00	1545	+0.06%
B02	ramrd_l	1,024	2061	1536.8	1.34	2062	+0.05%
B03	thr_rd_l	512	5688	768.8	7.40	5644	-0.77%
B04	thr_rd_w	512	3441	768.8	4.48	3593	+4.42%
B05	thr_rd_b	512	3440	768.8	4.47	3596	+4.53%
B06	ramwr_l	512	4335	768.8	5.64	4105	-5.31%
B07	ramwr_w	512	2177	768.8	2.83	2188	+0.51%
B08	ramwr_b	512	2173	768.8	2.83	2185	+0.55%

ID	Loop	N	Board	Table	Board/table	Model	Model error
B09	`ramfill`	256	9851	640.8	15.37	9884	+0.33%
B10	`romfill`	256	5784	640.8	9.03	5784	+0.00%
B11	`thr_rom_l`	512	3339	768.8	4.34	3339	+0.00%
B12	`fetch8k`	2	3616	1027.2	3.52	3618	+0.06%
B13	`fetch1k`	16	1212	1044.8	1.16	1214	+0.17%
B14	`ss_wr_w`	512	9056	768.8	11.78	9226	+1.88%
B15	`ss_wr_l`	512	18152	768.8	23.61	18437	+1.57%
B16	`pal_wr_w`	512	4069	768.9	5.29	4107	+0.93%
B17	`pal_wr_l`	512	8107	768.9	10.54	8200	+1.15%
B18	`spr_wr_l`	512	10254	769.1	13.33	10255	+0.01%
B19	`crw_wr_l`	512	20510	771.0	26.60	20512	+0.01%
B20	`io_rd_w`	512	2830	768.8	3.68	2825	-0.18%
B21	`port_rd_w`	512	2827	768.8	3.68	2828	+0.04%
B22	`bios_thr_l`	256	7560	384.8	19.65	7561	+0.01%
B23	`biosfill`	128	14680/14683	320.8	45.77	14682	+0.00%
B24	`frc_rd`	512	6408	768.8	8.34	6409	+0.02%
B25	`ftcsr_rd`	512	6414	768.8	8.34	6409	-0.08%
B26	`mul_l`	1,024	4362	2560.9	1.70	4363	+0.02%
B27	`muls_w`	1,024	2439	1536.9	1.59	2441	+0.08%
B28	`dmuls_l`	1,024	4360	2560.9	1.70	4361	+0.02%
B29	`div1`	1,024	1546	1537.0	1.01	1548	+0.13%
B30	`bra4`	1,024	2055	2048.6	1.00	2057	+0.10%
B31	`bt_nt`	1,024	1544	1536.8	1.00	1544	+0.00%
B32	`jsr4`	1,024	3594	3584.8	1.00	3596	+0.06%
B33	`copy`	512	9555	1792.9	5.33	9375	-1.88%
B34	`thr_wr_l`	512	4340	768.8	5.65	4107	-5.37%
B35	`ss_rd_w`	512	10203	768.8	13.27	10252	+0.48%
B36	`spr_rd_l`	512	11319	768.8	14.72	11273	-0.41%
B37	`pushpop`	1,024	4338	1536.6	2.82	4105	-5.37%
B38	`pool8`	1,024	2061	1536.6	1.34	2062	+0.05%
B39	`stk_rd`	1,024	2056	1536.6	1.34	2062	+0.29%

Table 19: P1 loop totals in FRT ticks (BOARD MODEL). Model `20261017`: 31 of 40 within 1%, median absolute error 0.15%, worst 5.37%.

E P2 loop groups

Warm medians of the 185 P2 CPU loops against two versions of the model. Build 12 was calibrated on P1 alone; `20261017` was fitted to these data. Every loop's eight raw samples are in the companion data.

		within 1%		median |error|, %	
Group	Loops	build 12	`20261017`	build 12	`20261017`
p1 bridge	40	29	29	0.13	0.15
cache conflict	36	24	24	0.70	0.70
instruction capacity	8	6	6	0.40	0.39
load dependency	9	9	9	0.17	0.20
multiply	18	3	18	18.00	0.13
write read handoff	12	3	9	12.88	0.14
copy	12	2	2	11.06	4.53
ram spacing	36	4	4	4.52	4.52
spacing control	6	3	3	0.80	0.76
frt phase	8	2	2	3.08	3.08
all	185	85	106	1.65	0.62

Table 20: P2 loop groups (BOARD MODEL). Worst single loop: build 12 −31.5% (a long multiply whose result is used), `20261017` −11.8% (`read_b_n256_g2`).

References

[1] J.R.Dunkley. *Where the Frame Goes: Scheduler Suspension, Timing Windows and SH-2 Rebuilds in Fightcade's CPS3 Emulation*, technical note, 30 September 2026. Source audit, ROM disassembly and parity-checked schedule measurements. Zenodo, doi:10.5281/zenodo.23064378.

[2] Hitachi. *SH7604 Hardware Manual*, ADE-602-085C, revision 4.0, 19 September 2001. Sections 4.6 (interrupt acceptance), 7 (bus state controller, write buffer), 8 (cache), 11 (free-running timer, Figure 11.7).

[3] D. Haywood, A. Naive, T. Slanina, ElSemi and MAME contributors. `src/mame/capcom/cps3.cpp`, `8b80cfd15d79`: board notes (chips, crystals, memories), address map, screen timing, DMA delay placeholders, encryption modes.

[4] J. Tejada (jotego) and contributors. JTCPS3 in `jtcores`, `70884b5347d9` (GPL-3.0): `jtcps3_main.v` (score-screen byte lane), `jtcps3_ss.v` (cell decode), the SH7604 cache module (dead-cart fetch decryption), KiCad schematics.

[5] herzmx. *sfiii4s-custom*, `https://github.com/herzmx/sfiii4s-custom`: instruction/data table method for custom (dead) carts.

[6] J.R.Dunkley, with board data recorded by XeD. *Cache, Bus, Interrupt Order and Slowdown on a CPS3: companion data*, 16 October 2026 (Appendix B).

[7] Fightcade (fightcadeorg), a fork of FinalBurn Neo; driver and core authors as in [8]. CPS3 driver and SH-2 core, `c95950140e24`: idle shortcut, periodic IRQ10, live-sprite display option.

[8] D. Haywood, A. Naive, T. Slanina, ElSemi and P. Bennett (MAME CPS3 driver); OopsWare, CaptainCPS-X and BisonSAS (FB Alpha port); J. Buchmueller (SH-2 core, ported by OopsWare); the FinalBurn Neo team. CPS3 driver and SH-2 core, `2bac03411897`: refresh-derived budget, palette-DMA interrupt delay.

[9] Javi (javi-ivaj). *cps3-fpga*, `923d1ba3a60d` (GPL-3.0): a second CPS3 core on JTFRAME with the same SH7604 model, written from MAME's driver with AI assistance; `jtcps3_cpu_cen.v` (exact 25 MHz), `jtcps3_vtimer.v` (video modes and the shipped `CPS3_CLEAN_PXL` rate), `jtcps3_main.v` (IRQ10 after 100 μs), `jtcps3_dma_char.v` (the decompressor and its note on half-written character RAM), `jtcps3_ssram.v` (byte-wide), `jtcps3_video.v` (line buffer).

[10] L. Petit (WydD). *Street Fighter III: 3rd Strike (CPS3)* input-lag result, `https://inputlag.science/engine/results/street-fighter-iii-3rd-strike-cps3`: 43.2 ms, 2.58 frames, input read during blanking.

[11] J.R.Dunkley. *CPS3 Timing, SH-2 Rebuilds, and Fightcade Compatibility*, 30 September 2026 (earlier report).

[12] D. Haywood, A. Naive, T. Slanina and ElSemi (CPS3 driver); J. Buchmueller, R. Belmont and D. Haywood (SH-2 core); MAME contributors. `src/mame/capcom/cps3.cpp` and `src/devices/cpu/sh`, `41a80b9dbaed`: read for Section 13.1; score-screen RAM mapped with `umask32(0x00ff00ff)`, DMA and sprite-list timers, multiply cycle counts, the free-running timer in `sh7604.cpp`.

[13] The FinalBurn Neo team; driver and core authors as in [8]. CPS3 driver and SH-2 core, `16484e0d941b`: the frame loop's periodic IRQ10 and the speed hack are still present.

[14] J. Tejada (jotego) and contributors; SH7604 model by S. Dvodnenko (srg320). JTCPS3 in `jtcores`, `bfced64dad83` (GPL-3.0), read for Section 13.1: `jtcps3_main.v` (interrupt encoding, score-screen byte lane), `jtcps3_chardma.v`, `jtcps3_paldma.v`, `jtcps3_video.v` and `jtcps3_vtimer.v` (display timing; the module instantiated is `cores/cps1/hdl/jtcps1_timing.v`), `cfg/mem.yaml` with the clock-enable generator `modules/jtframe/src/jtframe/mem/clocks.go`, `cfg/macros.def`, `cfg/mmr.yaml`, and the SH7604 model in `modules/jtframe/hdl/cpu/sh7604` (`BSC.sv`, `CACHE.sv`, `FRT.sv`, `MULT.sv`, `SH7604.sv`).

[15] bmo00. *arcfpga-cores*, `2bc038eeb3dd` (GPL-3.0): JTCPS3 ported to neptUNO+ and other platforms from `70884b53`, with a graphics-DMA address patch, build notes and the BIOS test firmware.

[16] Giblet (gibletto) and contributors. *sfIII3-cps3-decomp*, `34e22e7138a1`: C and SH-2 source of the arcade program built with the Hitachi toolchain; `functions.tsv` (arcade address, size, name), `src/cmd_main_2.c`, `src/coin_sw.c`, `src/poly_que.c`, `src/intr.src`, `src/tasksw.src`.