Hardware Deviations — All Closed — Release 1

Kenneth J Hamer-Hodges

May 2026

Hardware Deviations Report

v1.0 — 2026-04-29 CONFIDENTIAL

Generated by documentation audit (Task #85). Authoritative sources: `hardware/hw_types.py`, `hardware/layouts.py`, `hardware/boot_rom.py`, `hardware/tperm.py`, `hardware/call.py`, `hardware/ret.py`, `hardware/core.py`, `hardware/fused_unit.py`, `simulator/assembler.js`, `simulator/app.js`.

Resolved Corrections (applied in place)

ID	Summary	Files corrected
C-1	Integrity seal: FNV → CRC-16/CCITT (poly 0x1021, init 0xFFFF, 89-bit input)	`handbook.md`, `overview.md`, `getting-started.md`, `namespace-json.md`, `json-information.md`, `longevity.md`, `risks.md`, `boot-permission-rules.md`, `chipflow-technical-summary.md`, `church-machine-pico-ice.md`, `paper-sliderule-comparison.md`, `tunnel-messaging-example.md`
C-2	Minimum lump size refs: 32 → 64 words in prose	`abstractions.md`, `longevity.md`, `json-information.md`
C-3	Permission bit ordering: E(30) S(29) L(28) X(27) W(26) R(25)	`CM_LUMP_SPECIFICATION.md`
C-4	GT type values: NULL=00, Inform=01 (were swapped)	`lambda-instruction.md`
C-5	Church instruction count: 6 → 10; added LAMBDA/ ELOADCALL/ XLOADLAMBDA	`church-instructions.md`
C-6	Removed non-existent LOADX/SAVEX/LDM/STM	`church-instructions.md`, `overview.md`
C-7	SAVE operand labels: CRd=c-list dest (S perm), CRs=source GT (B=1)	`isa_encoding.md`, `instruction-set.md`
C-8	RETURN derivation: updated to show CR6/ CR14 via mLoad; CR5 installed by CHANGE from Zone ④ bounds	`lambda-instruction.md`
C-9	Namespace capacity: 131,072 → 65,536 (16-bit `object_id`)	`overview.md`, `garbage-collection.md`
C-10	R002 risk: 25-bit FNV → 16-bit CRC-16 seal	`risks.md`, `longevity.md`
C-11	TPERM presets 0x0A–0x0C: LE/SE/LSE → reserved (E isolation)	`isa_encoding.md`

Not changed: Patent files (`patent-*.md`), HTML figures, `trusted-security-base.md` comparison tables, `instruction-matrix.md` — preserved as historical design intent.

Deviations — Status Summary

All deviations D-1 through D-12 are **CLOSED/RESOLVED**. No open hardware deviations remain.

D-1: Minimum Lump Size — CLOSED

- **Decision (architect, May 2026)**: 64 words is an **advisory minimum** — the `n_minus_6` encoding physically cannot express sub-64-word lumps (`n_minus_6 = 0` gives 64 words; the field has no value for smaller). Hardware never receives a smaller-than-64-word lump because the encoding cannot represent one. The minimum is therefore self-enforcing by encoding, not by a run-time fault check.
- **Action taken**: `architecture.md` updated; no hardware change required. **CLOSED.**

D-2: RETURN MASK Field — CLOSED (bit 6 reserved; full mask not implemented)

- **Decision (architect, May 2026)**: The mask field in RETURN is currently **not implemented** in hardware (`ret.py` has no mask logic). Mask bit 6 is **reserved and must be zero** — hardware always re-derives CR6 unconditionally via cload; a set bit 6 has no effect and must not be encoded. The remaining mask bits (0–5, 7–11) are also unimplemented; hardware ignores them. No hardware mask mechanism will be added at this time. The boot epilogue should use bare `RETURN` (mask=0).
- **Action taken**: `instruction-set.md` and `isa_reference.md §8.4` updated (E-2 closed). Assembler should warn if any mask bit is set. Task #8 retracted. **CLOSED.**

D-3: TPERM Faulting Model — CLOSED (Task #873 + doc fix May 2026)

- **Hardware**: `tperm.py` lines 78–84 — reserved presets (codes 11–15 and B-variants 0x1B–0x1F) fault with `TPERM_RSV`. Non-monotonic restriction faults with `DOMAIN_PURITY`. `core.py` propagates these as hard faults.
- **Simulator fix (Task #873)**: `_execTperm` now calls `this.fault('TPERM_RSV', …)` for reserved presets; GT bounds check added (Z=0,N=1 on fail, not a hard fault). Simulator matches hardware.
- **Doc fix (May 2026)**: `instruction-set.md` and `church-instructions.md` updated — “TPERM never faults” replaced with correct faulting description; preset table row 10–15 updated from “Z=0, no fault” to “FAULT (TPERM_RSV)”.

- **Health-check Mode 1**: Not present in hardware. No hardware change planned. This sub-item is deferred indefinitely.
- **Status**: **CLOSED.** All three artefacts (hardware, simulator, docs) now agree on reserved-preset behaviour.

D-4: CR Register Remap (Dual Names)

CR12 — RESOLVED: `CR_THREAD = 12` (old alias) has been removed from `hw_types.py`. The single canonical constant is now `CR_THREAD_STACK = 12`. The one caller in `hardware/registers.py` was updated to use `CR_THREAD_STACK`.

CR14 — RESOLVED: `CR_CODE = 14` (old alias) has been removed from `hw_types.py`. The single canonical constant is now `CR_CLOOMC = 14`. All callers in `hardware/core.py`, `hardware/cload.py`, and `hardware/registers.py` have been updated to use `CR_CLOOMC`.

D-5: Navana.Init Wiring — CLOSED

- **Decision (architect, May 2026)**: Navana is a **normal LUMP** with a callable Init entry point. The boot ROM's direct CALL is a bootstrap shortcut, not the final design. Navana.Init must be reachable as a standard method call so that any authorised caller can re-initialise or extend the namespace controller.
- **Action taken (Task #17, May 2026)**: `boot_rom.py` now injects a 3-word Navana lump body at ROM word 320 (NS slot 5, byte 0x0500): header (cw=2), method_table[1]=2 (Init body offset), and a RETURN AL instruction as the Init body. A hardware `CALL CR_navana, 1` no longer faults with PRIVATE_METHOD — the method table entry at index 1 resolves to the Init body. `abstractions.md` Navana section updated with Init method index, signature, and permission. **CLOSED.**

D-6: FPGA MMIO / LED Pin Routing — CLOSED

- **Decision (architect, May 2026)**: A short MMIO summary table added to `architecture.md` (Memory Architecture section). Per-platform detail (pin numbers, active polarity) remains in `hardware-tang-nano-20k.md` and `hardware-ti60-f225.md`.
- **Action taken**: `architecture.md` updated with MMIO base address and register summary. **CLOSED.**

D-7: SAVE Operand Roles — Hardware Mismatch in app.js INSTRUCTION_DATA

- **Hardware mismatch**: `assembler.js` SAVE: fld_a = CRd (c-list, S perm), fld_b = CRs (source GT, B=1). `app.js` INSTRUCTION_DATA labels these fields with swapped CRd/CRs roles relative to assembler encoding.
- **Resolution (C-6 / Task #6)**: `app.js` INSTRUCTION_DATA SAVE entry corrected — CRd = C-List (S permission), CRs = Source GT (B=1), mSave validates all. Now matches assembler field assignments.

- **Status**: RESOLVED

D-8: CR5 Thread-Register Behavior (Installed by CHANGE)

- **Hardware (corrected, Task #451)**: The incorrect `cr5_stack` mechanism (a 256-entry hardware stack pushed by CALL, popped by RETURN) has been removed from all variants (`hardware/`, `ctmm_cap_amaranth/`, `ctmm_amaranth/`, `church_machine/`). CR5 is now correctly a thread register: `hardware/change.py` synthesises and installs the CR5 (Heap GT) from the incoming thread's Zone ④ bounds (base = `thread_base + 17*4`, limit_offset = `heapWords − 1` from the lump header `cc` field) in the new `INSTALL_CR5` FSM state after `READ_THREAD_HDR`.
- **Resolution (Task #451)**: All docs (`architecture.md`, `call-stack.md`, `garbage-collection.md`, `church-instructions.md`, `AMARANTH-SIMULATOR-AUDIT-2026-04-23.md`) and simulator comments updated to match the specification in `CM_LUMP_SPECIFICATION.md`. The `saved_cr5_gt` signal and `RESTORE_CR5`/`PHASE0` FSM states have been removed from all CALL and RETURN units respectively.
- **Status**: RESOLVED

D-9: LAMBDA Recursion — Idempotent Re-Entry via CR6 — CLOSED (Task #887)

- **Current hardware**: LAMBDA (opcode 7) uses `lambda_active_reg` (1-bit flag) and `lambda_pc_reg` (return address). The parent patent specifies non-nestable LAMBDA: a second LAMBDA while `lambda_active` is set causes a FAULT. CALL-mediated nesting is the only way to nest.
- **Discovery — return address invariance**: When LAMBDA CR6 self-invokes, every recursive call writes the **same return address** (PC+4 of the LAMBDA CR6 instruction) to `lambda_pc`. Re-executing LAMBDA CR6 while `lambda_active` is already set is **idempotent** — the same value overwrites the same register.
- **Discovery — the software already counts**: The method's own recursion argument (e.g., `n` counting down to 0) drives the recursion. The hardware does not need to independently track depth. The software determines when to stop (base case). No hardware counter is needed.
- **Revised design — idempotent re-entry**: Relax the non-nestable FAULT rule **exclusively for CR6 self-invocation**:
  - LAMBDA CR6 while `lambda_active = 1` → **permit** (idempotent, same return address)
  - LAMBDA CRn (n ≠ 6, i.e., different target) while `lambda_active = 1` → **FAULT** (different return address, true nesting — unsafe)
- **Two-RETURN exit path**: Base-case RETURN #1 sees `lambda_active = 1`, restores PC from `lambda_pc` (instruction after LAMBDA CR6), clears flag. RETURN #2 sees `lambda_active = 0`, pops the CALL frame from initial method entry. **Exactly 2 RETURNs regardless of recursion depth.**

- **O(1) trifecta achieved without a counter**:
  - O(1) entry: LAMBDA CR6 re-entry is one idempotent register write + branch
  - O(1) context switch: CHANGE saves `lambda_active` (1 bit, already in NIA indicators) + `lambda_pc` (already in machine status) — two fixed-size values regardless of depth
  - O(1) exit: 2 RETURNs always, never N
- **Hardware changes required**:
  1. `core.py`: Modify LAMBDA FAULT condition — gate on target CR: FAULT only when `lambda_active = 1` AND target is NOT CR6. When target IS CR6, permit re-entry (idempotent write to `lambda_pc_reg`).
  2. No other changes. `ret.py`, `change.py`, `lambda_unit.py` unchanged. No new registers, no NIA repartition, no counter logic.
- **Counter as optional enhancement**: A diagnostic counter (`lambda_depth_reg`, 8–16 bits) could be added later for performance monitoring or safety limits (FAULT on overflow to catch runaway recursion). This is not required for correctness or O(1) performance.
- **Previous design (superseded)**: The initial D-9 proposed a 16-bit `lambda_depth_reg` counter to replace stack frames. Analysis showed the counter is redundant — the software's own argument is the counter, and the existing 1-bit flag + `lambda_pc` register already provide O(1) behavior with the idempotent re-entry rule. The counter was discarded on RETURN anyway (zeroed and thrown away), confirming it served no purpose.
- **Platform strategy**: Change is minimal (FAULT condition refinement in `core.py` only). Safe to implement on both **Ti60 F225** and **Tang Nano 20K** simultaneously — no NIA format change, no stack format change.
- **Software impact**: None. The compiler already emits LAMBDA CR6 via `relambda()`. The simulator already models LAMBDA semantics correctly. The idempotent re-entry is purely a hardware-level refinement of the FAULT condition.
- **Risk**: Extremely low. Single-line FAULT condition change in `core.py`. No format changes, no new registers, no multi-file impact.
- **Affected files**: `hardware/core.py` (FAULT condition only)
- **Patent**: Documented in `docs/patent-ctmm-lambda-recursion-2026.md` (Claims 1–7)
- **Status**: CLOSED — implemented in `hardware/core.py`: `nested_lambda_fault` signal fires when `lambda_active_reg = 1` AND `cr_dst != 6`; `lambda_start_sig` gated to suppress when fault fires; fault wired to `FaultType.INVALID_OP` in the fault chain. LAMBDA CR6 re-entry while active proceeds silently (idempotent write to `lambda_pc_reg`).

D-10: CALL Method-Table Dispatch — PC=0 Eliminated

- **Patent / original spec**: CALL sets PC=0 after entering the callee lump. The callee's M00 Dispatch method (auto-generated ISUB/IADD/MCMP/BRANCHEQ loop at lump word 0) routes to the target method based on a selector in DR0.
- **Current hardware design**: PC=0 is always a FAULT — the lump header occupies word 0 and is never executable. The CALL instruction carries the method index in imm15 (15 bits). Hardware reads `memory[lump_base + method_index × 4]` to get the callee's first instruction word offset, then sets NIA = lump_base + offset × 4. A zero table entry is a private-method FAULT. Method index 0 is the single-entry-point shorthand (NIA = lump_base + 4, word 1) for abstractions with no method table.
- **ELOADCALL imm15 split**: bits[7:0] = c-list row (word offset into CRsrc c-list, 0–255); bits[14:8] = method index passed to the CALL phase (0–127).
- **Lump layout change**: Compiler writes method table at lump words 1..N (immediately after the lump header). Private methods store 0. Public methods store lump-base-relative word offset of their first instruction.
- **Software Dispatch method eliminated**: No auto-generated ISUB/IADD/MCMP code is prepended to the lump. The method table is data, not code.
- **Backward compatibility**: Existing programs encode imm15=0 (method index 0) → NIA = lump_base + 4. Behaviour is unchanged for single-entry-point abstractions.
- **Hardware changes required**: `hardware/call.py` — add `call_imm Signal(15)`, `call_imm_latched`, `method_entry_reg`, `use_method_table` flag; add `FETCH_METHOD_ENTRY` state after `FETCH_LUMP`; update `nia_computed` to Mux on `use_method_table`. `hardware/decoder.py` — add `call_method_index Signal(15)` (= imm_field) and split ELOADCALL signals (`eloadcall_method_index Signal(7)` = imm_field[8:15], `eloadcall_clist_row Signal(8)` = imm_field[0:8]).
- **Affected files**: `hardware/call.py`, `hardware/decoder.py`, `hardware/core.py` (wiring), `simulator/simulator.js` (_execCall), `simulator/cloomc_compiler.js` (remove _generateAutoDispatch), `simulator/app-compile.js`, `simulator/app-run.js` (codeOffset = methodTableSize + 1).

Simulator vs Hardware NS Entry Word Packing (informational)

- **Hardware** (`layouts.py`): NS Word 1 = `limit_offset[20:0] | gt_seq[6:0] | spare[3:0]`. NS Word 2 = `crc[15:0] | g_bit[16] | spare[31:17]`. `gt_seq` is in Word 1.
- **Simulator** (`app.js`): `word2_seals` stores `version[31:25] | spare[24:16] | CRC-16[15:0]`. `gBit` from Word 1.

- **Status: `namespace-json.md` updated to note divergence. Not tracked as a formal deviation since `namespace-json.md` documents the simulator JSON format, not hardware.**

Hardware Method-Table Dispatch (Task #837)

- **Change**: CALL and ELOADCALL now implement hardware method-table dispatch. Software Dispatch method (ISUB/IADD/MCMP/BRANCHEQ loop) is eliminated.
- **CALL imm15**: Method index (0–127). Index 0 → NIA = lump_base + 4 (word 1, single entry point). Index n > 0 → read `memory[lump_base + n×4]`; zero entry → PRIVATE_METHOD FAULT; else NIA = lump_base + entry×4.
- **ELOADCALL imm15**: Split field — bits[14:8] = method index, bits[7:0] = c-list row.
- **Lump layout**: Word 0 = lump header placeholder; words 1..N = method table entries (lump-base-relative word offsets; 0 = private); words N+1.. = method bodies.
- **Hardware files changed**: `hardware/call.py` (added `call_imm` Signal(15), `FETCH_METHOD_ENTRY` FSM state, conditional `nia_computed`), `hardware/decoder.py` (added `call_imm`, `eloadcall_row`, `eloadcall_method_index` split signals), `hardware/core.py` (wired `call_imm` to ChurchCall, changed c-list index to 0).
- **Simulator files changed**: `simulator/simulator.js` (_execCall, _execEloadcall), `simulator/cloomc_compiler.js` (removed `_generateAutoDispatch` and all 5 call sites), `simulator/app-compile.js`, `simulator/app-run.js`, `simulator/system_abstractions.js`, `simulator/repl.js` (lump layout: +1 word for header placeholder, PC start at word N+1).
- **Vocabulary**: NS table positions = “slot”; c-list positions = “row”; method table positions = “index”.
- **Status**: Fully implemented — hardware + simulator.

D-12: SHR/SHL Shift Flags — CLOSED (Task #857)

- **Previous hardware state**: SHR was always LSR (zero-fill only — no ASR mode). C flag was hardwired 0 for both SHL and SHR.
- **Simulator reference**: `_execShr` selects ASR via `imm[5]=1` (sign-extends result). `_execShl` and `_execShr` both set C = last bit shifted out (source[32-shamt] and source[shamt-1] respectively), gated on shamt > 0.

- **Fix applied (`hardware/core.py` lines 1084–1145)**:
  - SHR result uses `Mux(asr_mode, asr_result, lsr_result)` where `asr_mode = imm[5]`. ASR is implemented by sign-extending the 32-bit source to 64 bits (`Cat(src, Repl(src[31], 32))`) then shifting right and taking the low 32 bits.
  - `shr_flags_view.C` = `(source >> (shift_amt-1))[0]`, gated on `shift_amt != 0`.
  - `shl_flags_view.C` = `(source >> (32-shift_amt))[0]`, gated on `shift_amt != 0`.
  - Comment at line 1085 updated: “Flags: N = result[31], Z = (result==0), C = last bit shifted out, V = 0.”
- **Status**: CLOSED — hardware now matches simulator.

D-11: SWITCH instruction — hardware PassKey design vs. simulator CR swap — CLOSED

- **Hardware** (`hardware/switch.py`): SWITCH is a privileged capability-transfer instruction with three hard checks before any register is touched:
  1. **PassKey type check** — `CRs` must hold an Abstract GT (`gt_type == 0b11`). Any other type → `FAULT(INVALID_OP)`.
  2. **Target validity** — only `Tgt=5` (CR13) and `Tgt=7` (CR15) are permitted destinations. All other target values → `FAULT(INVALID_OP)`.
  3. **Sentinel address check** — `CRs.word1_location` must equal the hardware-reserved sentinel for the chosen target: `0xFFFFFFFE` for CR13, `0xFFFFFFFF` for CR15. Mismatch → `FAULT(INVALID_OP)`. On all checks passing, `ChurchMLoad` writes the source capability into the target privileged register (CR13 or CR15) with `m_elevated=1`, resetting G=0 on the namespace entry. The source CR is **not** cleared — SWITCH is not a swap, it is a one-way privileged install.
- **Previous simulator behaviour** (`_execSwitch`, pre-fix): performed an atomic CR swap `cr[crSrc] ↔ cr[target]` with only a NULL check. No PassKey check, target-validity check, sentinel check, or mLoad.
- **Fix (Task #880, May 2026)**: `simulator/simulator.js` `_execSwitch` replaced with the correct PassKey-gated one-way install. All three hardware checks are enforced in the same order as `hardware/switch.py`. Source CR is not modified. Target restricted to CR13 (Tgt=5) and CR15 (Tgt=7). Sentinel constants `0xFFFFFFFE` / `0xFFFFFFFF` match `hw_types.py`.
- **Status**: **CLOSED** — simulator now matches hardware.

Confidential — Kenneth Hamer-Hodges — May 2026