COLECOVISIONADAM.COM · ADAMARCHIVE.ORG

Frob-Apple Card & FROB BURNER Technical Reference and Reconstruction Guide

FROBCO Apple-card design family (1982–1983), incorporating the Bradley Bell / retroabandon hardware reverse engineering (2025), with a modern Arduino/RPi reimplementation for ColecoVision development

Document version	v0.3 — incorporates retroabandon hardware RE
Compiled	June 10, 2026
Hardware copyright	1982 by P.P.C. (silkscreen); product sold by FROBCO
Firmware copyright	1982–1983 by FROBCO (Ken Clements et al.)
Primary artifacts	PCB photos (Richard DiRocco); FROB2629.DSK and FROBBR{10,13}.DSK; FROB BURNER Preliminary Manual (FROBCO, 1983); FROBCO source listings; **retroabandon/apple2-re/frob KiCad 9 schematic by Bradley Bell, 12 April 2025**, plus `hwdoc/{components,memory-map,pinout,brochure}.md`
Confidence convention	Every factual claim is tagged **CONFIRMED**, **INFERRED**, **OPEN**, or **DESIGN**.
Changes from v0.2	The Frob-Apple control register bit map is now fully CONFIRMED from the Bradley Bell schematic. PPC360AN identification resolved (= 74LS367). IC reference designations in §5.1 corrected. A new §6.2 documents the bus-arbiter architecture in which bit 4 selects whether the Apple II or the VCS owns the on-card SRAM.

PART I

Original FROBCO Architecture

1. Executive Summary

The **Frob-Apple Card** is a slot card for the Apple II family that provides the host-side interface to the FROBCO Frob development system. Paired with a Frob Interface Unit on a target console, it lets a developer write, run, and debug 6502 cartridge code (Frob-26 for the Atari 2600) or Z-80 cartridge code (Frob-Coleco for the ColecoVision) from an Apple II workstation, with no EPROM burning required during the iteration cycle.

The **FROB BURNER** is FROBCO's companion EPROM-programmer card, used to commit a finished cartridge image to a 2732 EPROM after development is complete. The two cards share an unmistakable design family — same page-window architecture, same I/O alias scheme, same PMOVE bulk-transfer routine — and this document treats them together for that reason.

This reference compiles findings from five independent evidence streams: (1) physical inspection of an original P.P.C. 1982 Frob-Apple Card; (2) recovery and disassembly of FROBCO development disks (FROB2629.DSK and the BURNER disks); (3) the FROB BURNER Preliminary Manual (FROBCO, 1983) with its complete hardware register documentation and software appendices; (4) cross-validation against a 2025 reimplementation by Thom Cherryhomes (Frobble); and **(5) Bradley Bell's KiCad 9 schematic reverse engineering of the Frob-Apple Card and the retroabandon project's accompanying `hwdoc/` markdown documentation (April 2025)**. Where multiple sources agree, claims are tagged **CONFIRMED**.

The document then carries the architecture forward into two reconstruction paths:

- **Part II — Faithful Reproduction.** Building working replicas of the original Frob-Apple Card and FROB BURNER using period-correct or directly-equivalent components, suitable for use with surviving Apple II systems.

- **Part III — Modern Reimplementation.** A 2026-era replacement for the Apple II host: an Arduino Mega 2560 implementing the bus-control and bidirectional-port functions, paired with a Python program on a PC or Raspberry Pi providing the modernised LZRMON / AMON equivalent. The original FROBCO protocols are preserved exactly, so a recreated target-side debugger (DBUG.FRB) communicates identically with either the original Apple II host or the modern host.

2. Provenance

2.1 Physical artifact (Frob-Apple Card)

A P.P.C. 1982 Frob-Apple Card was photographed in March 2025. Solder-side legend reads *"COPYRIGHT © 1982 BY P.P.C."*. The card is in Apple II slot form factor with a standard 50-pin gold edge connector. The EPROM socket near the centre of the board is empty in this specimen.

2.2 FROBCO documentation

The **FROB BURNER Preliminary Manual** (FROBCO, copyright 1983) was recovered with all 30 pages of body, appendices A–F, and the rubber-stamp marking *"Preliminary Documentation — Subject to Change"*. The manual is dated 1983 and lists FROBCO at 603 Mission Street, Santa Cruz, California 95060, telephone 408-429-1552.

The manual is direct primary documentation for the BURNER hardware. Because the BURNER and the Frob-Apple Card share a register architecture (see §4), the BURNER manual indirectly documents portions of the Frob-Apple Card design that have not yet been independently verified.

2.3 Software artifacts

Six original Apple II DOS 3.3 disk images were recovered:

Image	Role	Notable contents
FROB2629.DSK	Primary Frob dev disk v2.9	AMON, FMON+OBJ, PMOVE+OBJ, EXPLORER+OBJ, XCONTROL+OBJ, FLOAD, FSAVE
FROBDV13.DSK	Earlier Frob dev disk v1.3	Same structure as v2.9

Image	Role	Notable contents
FRB26211.DSK	Frob-26 dev disk (assembler era)	ASM 6.2, VCS TEST, TREE, ADV, REUSE
FRB26227.DSK	Frob-26 dev disk (later)	Same as 211 with minor differences
FROBBR10.DSK	FROB BURNER 1.0 disk	BURNER v1.0, FBSAVE, PMOVE v1.1, sample VCS PROM images
FROBBR13.DSK	FROB BURNER 1.3 disk (clean)	BURNER v1.3, FBSAVE, PMOVE

The PMOVE source listing carries the credit *"Author: Ken Clements. Date: 10/13/82. Last Modified: 1/27/83. Version 1.3"*; FMON v1.3 is dated 1/26/83.

2.4 Independent verification (Frobble)

In March 2025, Thom Cherryhomes published Frobble — an Apple II/VCS sprite editor that drives an original Frob system via the bidirectional port. Frobble was developed against real hardware in 2025 and constitutes an independent, working implementation of the host-side protocol. Where Frobble and the original 1982/83 sources agree, those facts are treated as **CONFIRMED**.

2.5 Independent hardware reverse engineering (retroabandon)

Curt Sampson's retroabandon collective hosts the `apple2-re` GitLab repository, which contains a `frob/` subdirectory with a complete **KiCad 9 reverse-engineered schematic of the Frob-Apple Card**, drafted by Bradley Bell and dated 12 April 2025. The project also includes detailed markdown documentation:

- `hwdoc/components.md` — identifies every IC on the card, including the previously unknown PPC360AN parts (= 74LS367 hex bus drivers).
- `hwdoc/memory-map.md` — Apple-side and VCS-side memory and I/O maps, with the Frob-Apple control-register bit map documented at primary-source level.
- `hwdoc/pinout.md` — Apple II slot connector pin usage and Atari 2600 cartridge port pinout.
- `doc/burner.md`, `doc/manual.md`, `doc/brochure.md`, `doc/frob-52.md` — text transcriptions of the FROBCO documentation set.
- A MAME emulation of the FROB (PR #13596) which provides an executable model of the hardware behaviour.

The retroabandon schematic and our independent disassembly work agree on every testable point. v0.3 of this document treats schematic-derived facts as authoritative.

3. Board Identification

Figure 1. Frob-Apple Card, component side. Apple II slot card form factor; gold edge connector at bottom. Approximately 23 ICs plus one empty 24-pin DIP socket (centre, EPROM position). Two M58725P SRAMs at upper right comprise the 4 KB Frob RAM.

Figure 2. Frob-Apple Card, solder side. Legend "COPYRIGHT © 1982 BY P.P.C." visible at left. Hand-cut single-sided trace work consistent with a 1982-era small-run PCB.

4. The Page-Window Architecture (shared design DNA)

Both the Frob-Apple Card and the FROB BURNER expose their on-card memory through a **256-byte sliding window in the Apple II slot's $Cn00–$CnFF PROM space**, with a separate control register selecting which 256-byte page of the on-card memory is visible at any moment. This is the canonical FROBCO Apple-card design pattern.

For a card installed in slot *n*:

- The slot's 16-byte I/O range $C0n0–$C0nF (e.g., $C0A0–$C0AF for slot 2) contains the card's control register(s). **On the FROB BURNER, all sixteen addresses alias to the same control register** — this is documented verbatim in the BURNER manual §3.3 and is the partial-address-decode philosophy of these cards.
- The slot's 256-byte PROM range $Cn00–$CnFF (e.g., $C200–$C2FF for slot 2) is the on-card-memory window.
- The low nibble of the control register selects which of 16 pages of 256 bytes is visible in the window, giving a total of 4 KB of on-card memory accessible.

This page-window design is what makes the same PMOVE routine work for both cards — PMOVE doesn't care whether the underlying memory is EPROM (BURNER) or SRAM (Frob-Apple), only that pages can be selected via DEV+0 writes and bytes accessed via $Cn00.

```
                  APPLE II SLOT n
                  ═══════════════
$C0n0           Control register (write):
```

```
                    bits 0-3 = page select (0-F)
                    bits 4-7 = control bits (see §6)
                  Status register (read, Frob-Apple only):
                    bit 7 = Write OK
                    bit 6 = Read OK

$C0n1             Bidirectional data port (Frob-Apple only)

$Cn00-$CnFF       256-byte window into on-card memory
                  (page selected by control register low nibble)

on-card RAM/ROM:  page 0 [256B]  page 1 [256B]  …  page 15 [256B]
                  └──────────── 4 KB total ────────────┘
```

> **The $C0AC–$C0AF question, resolved.**
> Earlier documentation citing $C0AC, $C0AD, $C0AE, $C0AF as discrete Frob-Apple registers is reconcilable with the present analysis: those addresses alias to the same single control register, exactly as the BURNER manual §3.3 documents for its sister product. Different software conventions name the same hardware by different aliases. (BURNER manual: *"All sixteen locations address the FROB BURNER control register."*)

5. Frob-Apple Card Component Analysis

5.1 IC inventory

Reference designations and part identifications below are taken from `hwdoc/components.md` in the retroabandon repository, cross-verified against the Bradley Bell schematic and the photographed P.P.C. 1982 specimen.

Ref	Part	Function	Role on card
U1	74LS74	Dual D flip-flop	Bidir port handshake state: U1A = VCS-data-full, U1B = A2-data-full
U2	74LS138	3-to-8 decoder	VCS port select (decodes target-side $FFFx)
U3	74LS367 (PPC360AN)	Hex 3-state bus buffer	Status buffer — drives VCS_RD_OK / VCS_WR_OK / APL_RD_OK / APL_WR_OK onto the data bus during status reads
U4	74LS374	Octal D flip-flop, 3-state	A2-to-VCS data register

Ref	Part	Function	Role on card
U5	74LS244	Octal buffer, 3-state	VCS data read buffer (drives Apple-side byte onto VCS data bus on `IN A,(\$FFF2)`)
U6	74LS367 (PPC360AN)	Hex 3-state bus buffer	Lower VCS address buffer (VCSA0–VCSA7 → RAMA0–RAMA7 when VCS owns SRAM)
U7	74LS367 (PPC360AN)	Hex 3-state bus buffer	Lower A2 address buffer (latched A0–A7 → RAMA0–RAMA7 when Apple owns SRAM)
U8, U9	M58725P / 6116-compatible	2,048 × 8 SRAM	2 × 2 KB = **4 KB Frob RAM**
U10, U13, U18	74LS00	Quad 2-input NAND	Address-decode and write-strobe gating
U11	74LS02	Quad 2-input NOR	Address-decode gating
U12	74LS42	BCD-to-decimal decoder	Apple-side port select (decodes A0–A3 within the slot's I/O range)
U14	74LS374	Octal D flip-flop, 3-state	VCS-to-A2 data register
U15	74LS244	Octal buffer, 3-state	A2 data write buffer (Apple writes → SRAM, or → U4 for bidir)
U16	74LS244	Octal buffer, 3-state	A2 data read buffer (SRAM or U14 → Apple data bus)
U17	74LS174	Hex D flip-flop	**FROB control register** — six bits latched on writes to $C0n0–$C0nF
U19	74LS04	Hex inverter	Signal conditioning
U20	74LS367 (PPC360AN)	Hex 3-state bus buffer	Upper A2 address/page buffer (PAGEA0–3 → RAMA8–RAMA11 when Apple owns SRAM)
U21	74LS367 (PPC360AN)	Hex 3-state bus buffer	Upper VCS address buffer (VCSA8–VCSA11 → RAMA8–RAMA11 when VCS owns SRAM)
U22	74LS30	8-input NAND	Slot-base full-address decode

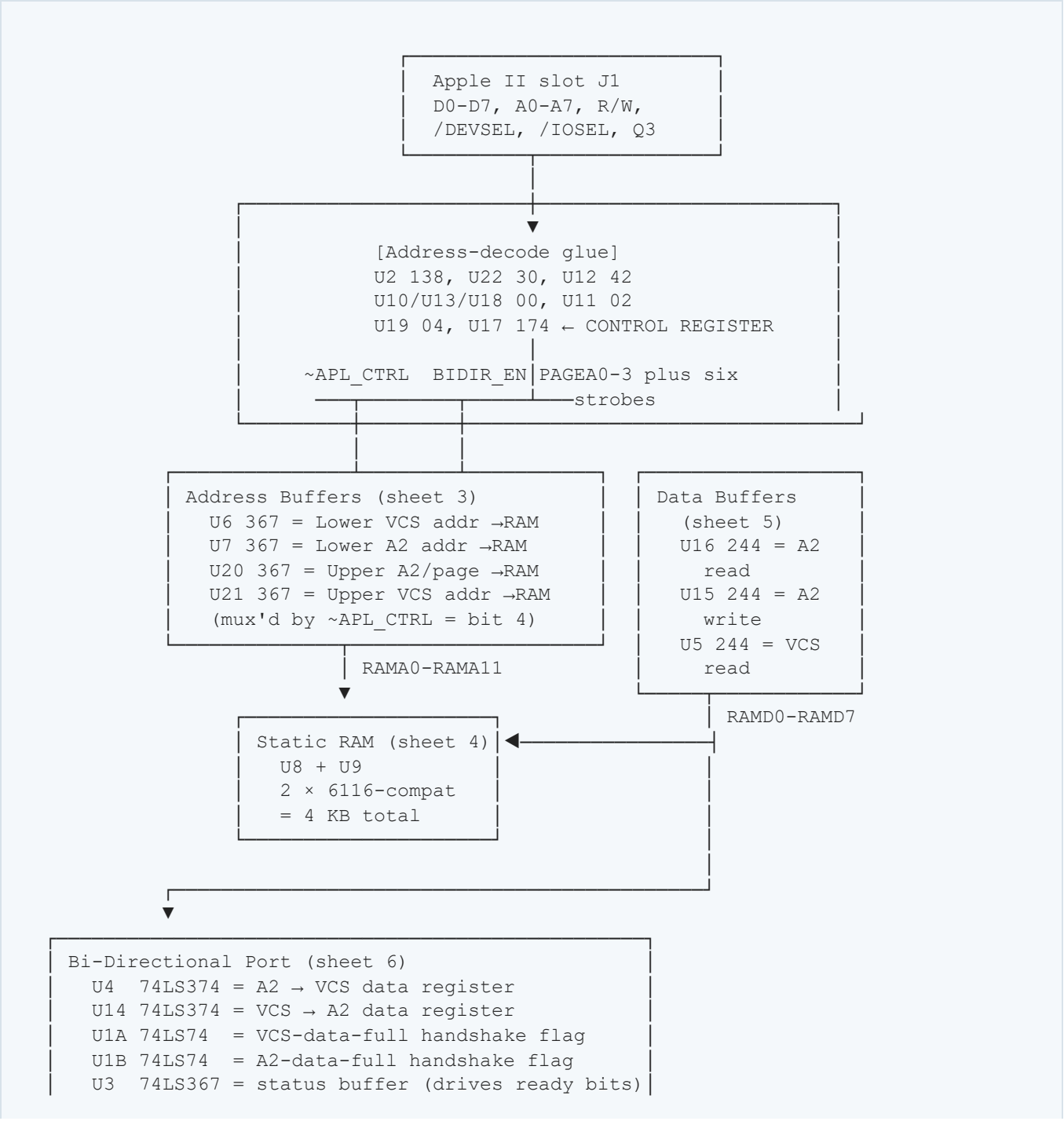
All identifications are **CONFIRMED** against the schematic. The board carries 18 ICs total: 2 SRAMs, 5 PPC360AN (74LS367) bus drivers, 2 octal latches (74LS374), 3 octal buffers

(74LS244), 1 hex flip-flop (74LS174 — the control register), 1 dual flip-flop (74LS74), and 5 small-scale glue parts (74LS00 ×3, 74LS02, 74LS04, 74LS30, 74LS42, 74LS138).

An empty 24-pin DIP socket near the centre-front of the board (visible in Figure 1) is wired as a 2716/2732 EPROM socket but is not addressed in the schematic and may have been intended for a future revision. **OPEN**: original contents and exact wiring of this socket.

5.2 Internal block hierarchy

The Bradley Bell schematic decomposes the card into five functional blocks, plus the slot-connector and control-register glue:



```
│   (entire block enabled by BIDIR_EN = bit 5)      │
└──────────────────────────────────────────────────┘
                          │
                          ▼ J2 ribbon to VCS-side board
```

The card's defining architectural feature is that the SRAM address pins (RAMA0–RAMA11) are not driven by any one source. They are driven by either *(a) the Apple's latched A0–A7 plus the four PAGEA bits* (when bit 4 = 0 and the Apple owns the SRAM), *or (b) the VCS's A0–A11 directly* (when bit 4 = 1 and the VCS sees the SRAM as a cartridge ROM). The two 74LS367 hex buffers in each direction (U6/U21 for VCS, U7/U20 for Apple) are gated by bit 4 so exactly one set is active at any moment.

This is what makes the card a true ROM emulator rather than a dual-port RAM. The VCS cannot tell that anything other than a 4 KB ROM is sitting in its cartridge slot; the Apple cannot read or write the SRAM while the VCS owns it. The two CPUs are time-multiplexed onto the same SRAM, with bit 4 as the arbiter.

6. Register Map

The Frob-Apple Card exposes two addresses in the slot I/O range: $C0n0 (control/status) and $C0n1 (bidirectional data). Both are mirrored across the 16-byte slot I/O range — partial-address-decode in the FROBCO family tradition — so writes to $C0n2, $C0n4, etc. also hit the control register.

Address	Dir	Function
$C0n0	READ	Status register (see §6.1)
$C0n0	WRITE	Control register (see §6.2)
$C0n1	READ	Bidirectional data port — byte from VCS. Valid when status bit 6 = 1.
$C0n1	WRITE	Bidirectional data port — byte to VCS. Caller must wait until status bit 7 = 1.
$Cn00–$CnFF	R/W	256-byte window into the on-card SRAM. Active only when bit 4 of the control register = 0. The window's offset into the 4 KB SRAM is determined by bits 0–3 of the control register × 256.

6.1 Status register ($C0n0 read)

Bit	Function
7	**APL_WR_OK** — 1 = data-write buffer is empty, Apple may write a byte to $C0n1. 0 = buffer holds an unread byte; Apple must wait.
6	**APL_RD_OK** — 1 = a byte from the VCS is waiting in the data-read register, Apple may read $C0n1. 0 = no byte available.
5–0	Not driven (read as undefined).

Source: U3 (74LS367) buffers two signals — APL_WR_OK from U1B (A2-data-full flag, inverted) and APL_RD_OK from U1A (VCS-data-full flag) — onto D7 and D6 respectively when a status read is decoded.

6.2 Control register ($C0n0 write)

The control register is U17, a 74LS174 hex D flip-flop. Six bits of the Apple data bus (D0–D5) are latched on writes to any address in $C0n0–$C0nF. D6 and D7 are physically not connected.

Bit	Signal	Function
7	—	Not used (74LS174 is 6 bits wide; D7 is unwired)
6	—	Not used (D6 unwired)
5	BIDIR_EN	**Bidirectional port enable.** 1 = U4/U14/U1 bidir block active; 0 = bidir port disabled (handshake flags frozen). Note: per `memory-map.md`, the bidir port is unconditionally disabled when bit 4 = 0, regardless of BIDIR_EN.
4	~APL_CTRL	**SRAM ownership selector.** 0 = Apple II owns the SRAM (accessible via the $Cn00 page window); 1 = VCS owns the SRAM (it appears to the VCS as a 4 KB cartridge ROM at $1000–$1FFF, mirrored at $3000, $5000, …, $F000).
3	PAGEA3	SRAM A11 — Frob page-select MSB
2	PAGEA2	SRAM A10
1	PAGEA1	SRAM A9
0	PAGEA0	SRAM A8 — Frob page-select LSB

> **Note on a documentation discrepancy.**
> The retroabandon `memory-map.md` describes the page-select field as "b3..b1" (three bits, eight pages). The schematic at `main.kicad_sch` shows four PAGEA outputs from U17

> wired to RAMA8–RAMA11, supporting 16 pages of 256 bytes for the documented 4 KB SRAM. v0.3 follows the schematic and treats the "b3..b1" wording as a typo for "b3..b0".

6.3 The three (effectively two) operating modes

Bits 4 and 5 together define the card's operating mode. Because the bidir port is always disabled when the Apple owns the SRAM, only two of the four combinations are functionally distinct:

Bit 4 (~APL_CTRL)	Bit 5 (BIDIR_EN)	Effective mode	Used by
0 (Apple owns SRAM)	×	**LOAD mode**. Apple reads/writes SRAM via $Cn00 window; VCS sees no cartridge; bidir port inactive.	PMOVE during code upload; AMON memory editor when paused
1 (VCS owns SRAM)	1	**RUN mode**. VCS sees SRAM as cart ROM and is free-running; Apple can talk to target firmware (FMON) via the bidir port at $C0n1.	Normal Frob operation while target code is executing
1	0	Run-without-comm. VCS sees cart ROM; bidir port inactive; Apple has no channel to talk to the target.	Rarely used; could be useful for "set and forget" demo runs

The canonical AMON sequence to start a target program is therefore: `POKE DEV, $00` (LOAD mode, page 0) → `CALL PM+22` several times to upload code into all 16 pages → `POKE DEV, $30` (RUN mode, page 0 — page bits are now ignored by the VCS-side path) → target reset → target runs and communicates with AMON via the bidir port.

6.4 BURNER vs Frob-Apple — bit-map comparison

With both bit maps now CONFIRMED from primary sources, the family resemblance and the differentiation can be stated precisely:

Bit	FROB BURNER	Frob-Apple Card
7	Unused (74LS174 is 6 bits wide)	Unused (same)
6	Unused	Unused
5	25 V supply ON + data latch OE	BIDIR_EN (bidirectional port enable)
4	CHIP SELECT deassert (safe state)	~APL_CTRL (SRAM ownership: 0 = Apple, 1 = VCS)

Bit	FROB BURNER	Frob-Apple Card
3..0	EPROM A11..A8	SRAM A11..A8 (PAGEA3..PAGEA0)

The page-select semantics are exactly preserved; the bit 4/5 functions diverge in form (programming-voltage control on the BURNER, bus-arbitration and comm-enable on the Frob-Apple) but share the underlying pattern of "two gating bits over a 4-bit page select" that defines the FROBCO Apple-card family.

7. Bidirectional Port Protocol

7.1 Target-side ports (VCS / Atari 2600 side)

From the VCS's perspective, the FROB bidirectional port appears at three addresses in the cartridge's top page when the bidir port is enabled (bit 5 = 1, bit 4 = 1):

VCS address	Dir	Function
$FFF0	WRITE	Send a data byte to the Apple. Caller should test bit 7 of $FFF1 first; bit 7 clears on this write and is set again by the Apple's read.
$FFF1	READ	Status register: bit 7 = ready for output ("VCS may write to $FFF0"); bit 6 = data available ("Apple has written a byte readable at $FFF2"). The test idiom is `BIT $FFF1 / BPL ... / BVC ...` .
$FFF2	READ	Read a data byte from the Apple. Caller should test bit 6 of $FFF1 first; bit 6 clears on this read and is set again when the Apple writes a new byte.

These addresses live inside the VCS's cartridge address space (the 6502 in the VCS lacks address pins A13 and above, so $1000–$1FFF and all its mirrors map to the cart). Confirmed by both the recovered FMON.T source (labels PSTAT, RDATA, WDATA) and `memory-map.md` from the retroabandon repository.

7.2 Apple-side handshake idiom

From AMON's send-byte and receive-byte routines (lines 3000–3220), generalised:

```
// Apple writes one byte b to VCS
while ((peek($C0n0) & 0x80) == 0) { /* spin until APL_WR_OK */ }
poke($C0n1, b);
```

```
// Apple reads one byte from VCS
while ((peek($C0n0) & 0x40) == 0) { /* spin until APL_RD_OK */ }
b = peek($C0n1);
```

The target-side equivalents in FMON.T (labels WBA = "write byte to Apple", RBA = "read byte from Apple") use 6502 `BIT $FFF1 / BPL / BVC` so the same status bits land in the N (bit 7) and V (bit 6) processor flags for cheap branch testing.

7.3 FMON command set (target-side)

The target-side debugger (FMON, 256 bytes at $FF00 on the VCS) implements a tiny command interpreter that reads one byte from the Apple and dispatches:

Cmd byte	Operation	Trailing bytes (Apple sends)	Response (VCS sends back)
$10	Read memory	hi-addr, lo-addr, count	count bytes from VCS memory
$20	Write memory	hi-addr, lo-addr, count, then count bytes	none
$30	Go from break	none	none (VCS resumes from saved register state)
$40	Jump indirect	hi-addr, lo-addr	none (VCS does `JMP (addr)`)

7.4 Break entry protocol

When the target hits a breakpoint, FMON enters its `BREAK` handler and sends the saved processor state to the Apple *before* entering the command loop:

1. Accumulator (A)
2. Flag-restore code (1 of 3 sentinel values that lets the resume path reconstruct the N and Z flags without using the stack)
3. X register
4. Y register
5. Stack pointer (S)
6. 32 bytes of zero-page-adjacent RAM ($00E0–$00FF, the working scratchpad)

8. Software Architecture

8.1 The Frob-Apple software stack

Component	Side	Language	Role
AMON	Apple II	Applesoft BASIC (52 sectors)	Interactive monitor: command parsing, register display, memory edit, disassembly
PMOVE.OBJ	Apple II	6502 ML, 176 bytes at $0300	Bulk page-mover; BLOAD'd by AMON at startup; provides DNLOAD/UPLOAD entry points
FLOAD, FSAVE	Apple II	Applesoft BASIC	Disk-file I/O via PMOVE
EXPLORER, XCONTROL	Apple II	6502 ML	VCS hardware lab / test programs (not analysed in this document)
FMON	Target (VCS 6502)	6502 ML, 256 bytes at $FF00	Target-side debugger and command interpreter
DBUG.FRB	Target (CV Z-80)	Z-80 ML, ~512 bytes	Frob-Coleco analog of FMON; structural template recovered, binary not yet recovered

8.2 PMOVE — bulk page mover (annotated)

PMOVE is the workhorse that moves data between Apple memory and the on-card Frob RAM. Loaded at $0300, 176 bytes total. Two entry points:

Label	Address	Purpose
PMOVE	$0300	Internal: 256-byte copy primitive (self-modifying)
FSLOT	$0312	Parameter: Apple slot number (1–7)
LFROM	$0313	Parameter: Apple-side memory page high byte
LNPGS	$0314	Parameter: number of 256-byte pages to transfer
STPAGE	$0315	Parameter: starting Frob page-select value
DNLOAD	$0316	Entry: Apple → Frob (called by `CALL PM+22`)
UPLOAD	$0333	Entry: Frob → Apple (called by `CALL PM+51`)

Label	Address	Purpose
FXFER	$034D	Common transfer body
PPOINT	$037B	Patched STA — writes STPAGE to DEV+0 (FROB PAGE SELECT)

9. Sister Product: FROB BURNER

The FROB BURNER is FROBCO's EPROM programmer card. The Preliminary Manual (1983) provides complete hardware documentation, which transitively documents portions of the Frob-Apple Card design.

9.1 BURNER hardware overview

- Apple II slot card, fits slots 1–7
- On-board edge connector (J2) accepts a cartridge adapter holding the 2732 EPROM being programmed
- On-board **25 V programming supply** with a manual ON/OFF switch in the upper-left corner of the card
- Single write-only control register accessed at any of the 16 slot I/O addresses
- 256-byte PROM-space window at $Cn00–$CnFF maps to one of 16 pages of the 4 KB EPROM
- Internal address and data latches capture the Apple's A0–A7 and D0–D7 on writes to the PROM-space window

9.2 BURNER programming sequence (from Appendix A source)

The 50 ms programming pulse is generated by the host CPU running an inline ML timing loop. Lines 402, 450, 490, 495, 500, 505 of the BURNER program execute this sequence per byte:

```
POKE DEV, $10      ; page 0, CS deasserted, 25 V off  (safe init)
POKE DEV, $30      ; page 0, CS deasserted, 25 V ON  (raise programming voltage)
POKE DEV, page+$30 ; select target page, still safe
POKE IO+low, data  ; latch low address and data byte
POKE DEV, page+$20 ; ── pulse start: CS asserted, 25 V still on
CALL WT            ;    [50 ms timing loop runs here]
POKE DEV, page+$30 ; ── pulse end: CS deasserted
```

9.3 The WT timing loop (Appendix A bytes)

The BURNER program POKEs 11 bytes of machine code into $0300–$030D before running, replacing PMOVE's normal residence with the WT routine. The code:

```
$0303  A0 1E     LDY #$1E     ; outer count = 30
$0305  A2 00     LDX #$00     ; inner count = 256
$0307  CA        DEX
$0308  D0 FD     BNE $0307    ; inner loop
$030A  88        DEY
$030B  D0 F8     BNE $0305    ; outer loop
$030D  60        RTS
```

30 × 256 × 5 cycles ≈ 38 ms at 1.023 MHz, plus instruction overhead → close enough to 50 ms for the 2732 datasheet's specification. The BURNER manual warns that any DMA or interrupt activity will perturb this timing and may damage the EPROM or the BURNER itself.

9.4 FBSAVE — EPROM read-back (Appendix F)

FBSAVE uses PMOVE's UPLOAD entry (`CALL PM+51`) to read 16 pages of EPROM via the page-window into Apple memory at $8000, then `BSAVE` s a 4 KB binary file. Parameter block: `STPAGE = 0, LNPGS = 16, LFROM = 128` .

PART II

Faithful Reproduction

10. Component Sourcing in 2026

The Frob-Apple Card and FROB BURNER were assembled almost entirely from 7400-series TTL still available as new old stock (NOS) and from manufacturers with active production runs. The list below identifies modern equivalents and notes for each part.

Original part	Function	2026 sourcing
74LS00, 02, 04, 30, 42, 74, 138, 174, 244, 374	Standard TTL	Most are still produced as 74HCT or 74LS by TI, Diodes Inc., NXP. The 74HCT-series parts are CMOS but TTL-input-compatible and a direct drop-in replacement. Available from Mouser, Digi-Key, Jameco; bulk available as NOS on eBay.

Original part	Function	2026 sourcing
M58725P (2,048 × 8 SRAM, 24-pin DIP)	4 KB Frob RAM (two needed)	Obsolete. Direct functional replacements: **HM6116** (2 KB SRAM, 24-pin DIP, widely available NOS and new). For a single-chip substitute, the **HM6264** (8 KB, 28-pin DIP) can replace both M58725Ps with adapter wiring.
2732 EPROM (4 KB, 24-pin DIP)	BURNER target; also Frob-Apple internal EPROM (empty in surviving specimen)	NOS still available. For new builds where erasability is not needed, **AT28C64B** (8 KB EEPROM, 28-pin DIP) or **SST39SF010** (Flash) are programmable with modern equipment (TL866-II, etc.).
"PPC360AN" (five 16-pin DIPs)	Hex 3-state bus driver — confirmed as 74LS367 (or relabelled Signetics 8T97 or Motorola MC6887, all with same pinout per `components.md`)	Available as 74LS367, 74HCT367, or 74F367 from TI, NXP, Diodes Inc. Drive-strength specs differ slightly between vendors; for a faithful build use NOS 74LS367 or 8T97. 74HCT367 is a drop-in CMOS replacement.
25 V programming supply (BURNER only)	EPROM programming voltage	For the BURNER, an MC34063-based step-up converter from the Apple's 12 V rail will produce 25 V at the few mA needed. Manual ON/OFF switch retained for safety.
50-pin gold edge connector	Apple II slot finger	Mating part is the .100-inch dual-row card-edge contact pattern Apple specified; PCB houses (OSH Park, JLCPCB) reproduce this trivially with ENIG finish.

11. PCB Design Notes

11.1 Source for layout

The Bradley Bell KiCad 9 schematic (`retroabandon/apple2-re/frob/kicad/a2board.kicad_sch` , April 2025) is the canonical electrical reference for replica builds. The retroabandon project also includes a PCB layout, though as of that date it has not been fabricated and tested. Treat the layout as a starting point that may need correction during bring-up.

11.2 Layout philosophy

The original P.P.C. 1982 boards were hand-cut, single-sided traces. A modern two-layer board with a ground pour gives much better signal integrity at no cost premium and is recommended for any new build.

11.3 Critical signals

- **/DEVSEL** (Apple slot pin 41) — active low when the CPU accesses the slot's $C0n0–$C0nF range. Gates writes to the control register and reads from the status register.
- **/IOSEL** (Apple slot pin 1) — active low when the CPU accesses the slot's $Cn00–$CnFF range. Gates the SRAM page-window access.
- **R/W** (Apple slot pin 18) — distinguishes Apple read cycles from write cycles.
- **Q3** (Apple slot pin 37, 2 MHz asymmetric clock) — used by the Frob-Apple Card for timing. Note that the canonical Apple-card timing reference Φ0 (pin 40) and Φ1 (pin 38) are *not* connected on this board. The retroabandon `pinout.md` flags this as an unusual design choice but the schematic and our specimen are consistent on this point.

11.4 Address decoder

The Frob-Apple Card decodes A0 (LSB of the slot's I/O nibble) to split the $C0n0–$C0nF range into two registers: $C0n0 (control/status) and $C0n1 (bidirectional data). A2 is also decoded to enable the per-register strobes (control write vs status read vs bidir write vs bidir read), through the U2 74LS138 and the U12 74LS42 in the schematic's "main" sheet.

11.5 VCS-side board

The cartridge-side board that plugs into the VCS slot is minimal: a 24-pin DIP socket for either a 2716/2732 EPROM or the ribbon cable to the Frob-Apple Card, plus a single 74LS04 with one of its six inverters used to convert the cart's A12/ENB line (active-high) to the EPROM /CE (active-low). The other five inverter gates are unused. This minimalism is essentially preserved in the modern reimplementation (see §17).

12. Operating Procedure

To use a reproduced Frob-Apple Card with a surviving Apple II:

1. Power down the Apple. Insert the Frob-Apple Card into any slot 1–7. Power up.
2. Boot a copy of FROB2629.DSK (or FROBDV13.DSK).

3. Run AMON (typed as `RUN AMON` at the Applesoft prompt, or selected from the disk's HELLO menu).
4. AMON prompts for the slot number. Enter 1–7 to match the card's installed slot.
5. AMON prompts to load a file. Either decline (Y/N → N) to enter the monitor cold, or enter the filename of an existing target binary.
6. AMON prompts to load FMON. Answer Y to upload FMON to the target via PMOVE; the target's reset vector will then point at FMON.
7. Power on the target console. The target's reset jumps into FMON, which immediately handshakes with AMON via the bidirectional port.
8. The user is now at AMON's command prompt and can edit memory, set breakpoints, single-step, etc., on the running target.

PART III

Modern Reimplementation

13. Architectural Translation

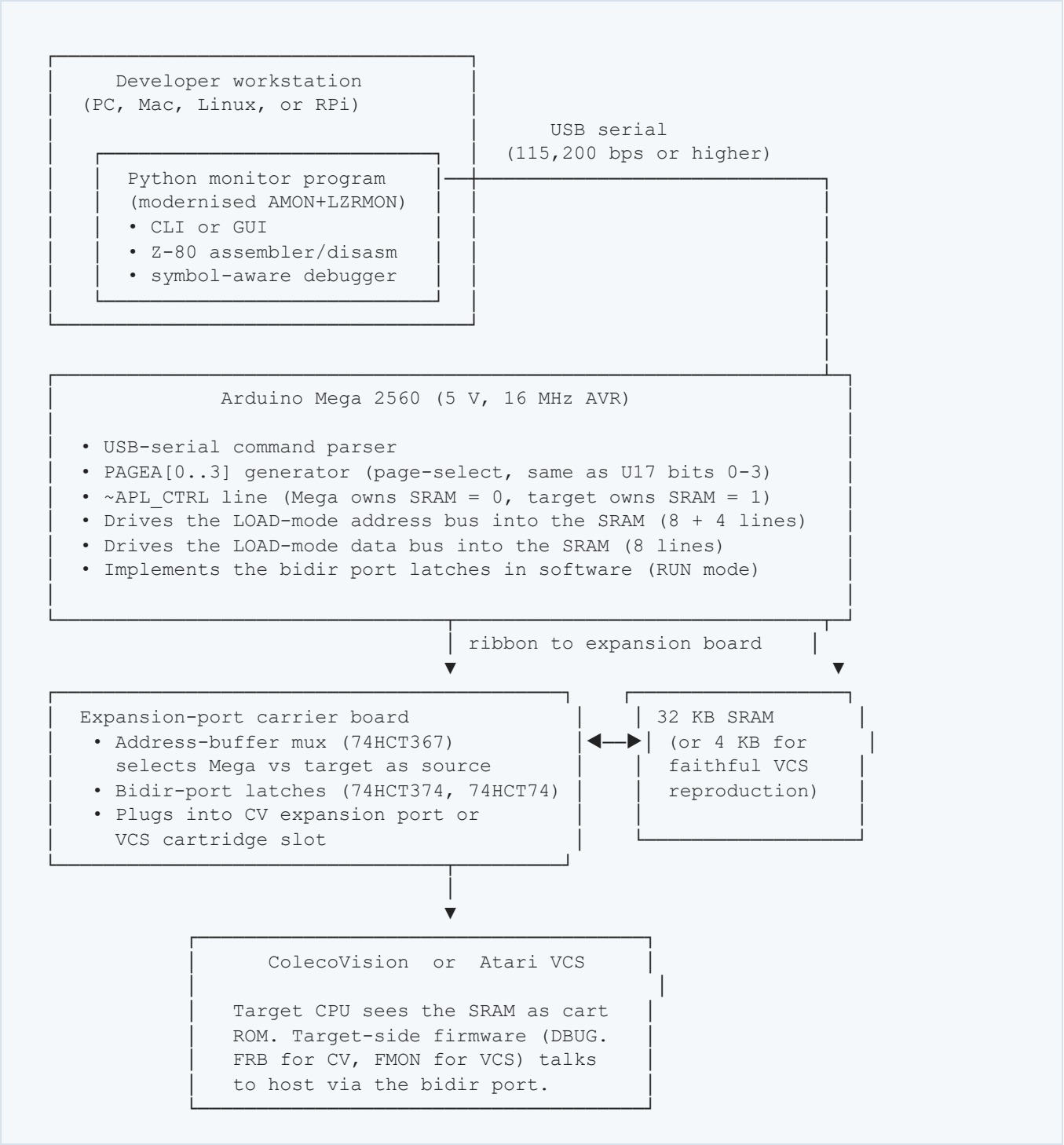
The modernised system replaces the Apple II host with a two-board host: a Raspberry Pi (or PC) running the developer UI, and an Arduino Mega 2560 acting as the bus-control microcontroller. The target-side hardware is unchanged in concept — a board that plugs into the ColecoVision (or VCS) expansion port and emulates the cartridge with on-board SRAM — but is realised in modern components.

13.1 Translating the original architecture

The original Frob-Apple Card achieves cart-RAM emulation through a beautifully simple mechanism: a 4 KB SRAM whose address-bus inputs are switched between two source buffers by a single control bit. The Apple's bus drives the SRAM during code upload (LOAD mode); the VCS's bus drives the SRAM while the target runs (RUN mode). No /BUSREQ, no bus arbitration in the conventional sense — just a one-bit toggle of which set of 74LS367 buffers is enabled.

The modern reimplementation preserves this exact mechanism. The Arduino Mega 2560 replaces the Apple II as the LOAD-mode bus master; the on-board SRAM and its dual address-buffer set are otherwise identical to the original. The Python program on the PC or Raspberry Pi replaces AMON / FLOAD / FSAVE.

13.2 System block diagram



13.3 Protocol preservation

The original FROBCO protocols are preserved **exactly** on the target side: the bidirectional port still appears at the same memory or I/O addresses the original used (top-of-cart $FFF0/$FFF1/$FFF2 on the VCS; Z-80 I/O ports $7C/$7D for the Frob-Coleco target), with the same status flag bit assignments (bit 7 = Write OK, bit 6 = Read OK) and the same FMON command set ($10 read, $20 write, $30 go, $40 jump-indirect). A target-side debugger written

for the original Frob-Apple Card will work without change on the modern Arduino-emulated card, and vice versa.

14. Host-Side Software (Python on PC or RPi)

14.1 Architecture

A single Python program implements the modernised monitor. It communicates with the Arduino Mega over USB serial at a high baud rate (typically 115,200 or higher). The program exposes either a CLI or a GUI; both call into the same protocol layer.

```
frobmon/
├── frobmon.py          — entry point, CLI parser, REPL
├── protocol.py         — USB-serial framing; command/response packets
├── target.py           — high-level operations: read_mem, write_mem,
│                         set_breakpoint, get_registers, go, single_step
├── monitor.py          — interactive REPL: address parsing, hex display,
│                         disassembly, symbol table
├── assemble.py         — Z-80 assembler integration (e.g., via z80asm)
├── disasm.py           — Z-80 disassembler (e.g., via z80dasm or built-in)
├── symtab.py           — symbol table; tracks labels from assembly
├── images.py           — load/save target image files (.bin / .col / etc.)
└── ui_gui.py           — optional Qt or Textual GUI front end
```

14.2 Command parity with AMON

The Python monitor implements every AMON command, plus a modern superset:

AMON command	Python equivalent	Modern superset
`<addr>.` (read memory)	`m <addr>` or `read <addr> [count]`	Symbol-name addresses; supports ranges and structured types
`<addr><` (write memory)	`w <addr> <bytes>`	Accepts hex, decimal, ASCII strings, file contents
`R` (registers)	`regs`	Watches; auto-refresh on each break
`P` / `K` (set/kill breakpoint)	`bp <addr>` / `bd <n>`	Conditional breakpoints; symbol-named
`G [addr]` (go)	`go [addr]`	—
`L [addr]` (disassemble)	`dis <addr> [count]`	Z-80 disassembly with symbols and comments

AMON command	Python equivalent	Modern superset
`I` (load file)	`load <path> [@addr]`	Supports .col / .bin / .hex / Intel HEX; with optional symbol .sym
(none)	`step` / `next`	Single-step (DBUG.FRB cooperatively inserts a one-shot breakpoint)
(none)	`watch <expr>`	Memory or register watch with break-on-change
(none)	`trace`	Trace execution to a log file

14.3 USB-serial protocol

The host ↔ Arduino link uses a simple framed protocol so the Arduino can multiplex bus operations, debugger events, and trace output:

```
Frame: [STX] [LEN] [TYPE] [PAYLOAD…] [CRC16]    STX = 0x7E

Types (host → Arduino):
  0x01  READ_MEM   payload: u16 addr, u16 count
  0x02  WRITE_MEM  payload: u16 addr, u16 count, then count bytes
  0x03  GO         no payload
  0x04  HALT       no payload (forces /BUSREQ)
  0x05  RESET      payload: 1 byte (0 = release, 1 = assert)
  0x06  GET_REGS   no payload
  0x07  SET_BP     payload: u16 addr
  0x08  CLR_BP     payload: u16 addr
  0x09  STEP       no payload

Types (Arduino → host):
  0x81  READ_DATA      payload: requested bytes
  0x82  ACK            payload: 1 byte status
  0x83  BREAK_EVENT    payload: register block (A, F, BC, DE, HL, IX, IY, SP, PC)
  0x84  TRACE_EVENT    payload: 1 byte cycle info + u16 PC
  0xFF  ERROR          payload: 1 byte error code, optional message
```

DESIGN — the framing is a recommendation; implementations may choose SLIP, COBS, or any other reliable framing. The important property is that each frame is independently parseable.

15. Microcontroller Firmware (Arduino Mega 2560)

15.1 Pin budget

Following the original Frob-Apple architecture, the Arduino does *not* need to bus-master the target's CPU bus. Instead it owns the SRAM during LOAD mode and releases it to the target during RUN mode, controlled by a single GPIO. This dramatically reduces the required pin count compared to a full bus-mastering design.

Function	Arduino Mega pins	Count	Notes
SRAM data bus (RAMD0–RAMD7)	PORTA pins 22–29	8	Bidirectional; tristate when target owns SRAM
SRAM address low (RAMA0–RAMA7)	PORTC pins 30–37	8	Drive only when ~APL_CTRL = 0
SRAM address high (RAMA8–RAMA11)	Pins 38–41	4	Drive only when ~APL_CTRL = 0
SRAM /WE, /OE, /CS	Pins 42, 43, 44	3	Drive only when ~APL_CTRL = 0
~APL_CTRL (SRAM ownership)	Pin 45	1	0 = Arduino owns SRAM, 1 = target owns SRAM
BIDIR_EN (bidir port enable)	Pin 46	1	Gates the bidir latches on the carrier board
Bidir port D0–D7 (shared GPIO)	PORTL pins 49–42	(shared with SRAM data)	Time-multiplexed; the same PORTA can serve both
Bidir port status / strobes	Pins 47, 48	2	Read APL_RD_OK and APL_WR_OK flags from the carrier board's 74LS374-equivalent
Target /RESET	Pin 49	1	Open-drain; asserted while loading code, released to start target
Total used		**~30 of 54**	Plenty of headroom for debug LEDs, status, additional features

15.2 Loading code into the SRAM (LOAD mode)

The Arduino writes a byte into the cartridge SRAM by driving RAMA0–RAMA11 and RAMD0–RAMD7, then strobing /WE. No interaction with the target CPU is needed; the target is held in reset and the carrier-board mux directs the SRAM's address pins to the Arduino's drivers.

```
void sram_write(uint16_t addr, uint8_t data) {
  PORTC = addr & 0xFF;            // RAMA0-RAMA7
  PORTL = (addr >> 8) & 0x0F;     // RAMA8-RAMA11
```

```
  DDRA = 0xFF;                    // RAMD as outputs
  PORTA = data;                   // data to SRAM
  digitalWrite(SRAM_CS, LOW);
  digitalWrite(SRAM_WE, LOW);
  // ~70 ns access time on a 6116; 1 us delay is enormously conservative
  delayMicroseconds(1);
  digitalWrite(SRAM_WE, HIGH);
  digitalWrite(SRAM_CS, HIGH);
}

void load_cart(const uint8_t *image, size_t len) {
  // Phase 1: ensure target is held in reset and we own the SRAM
  digitalWrite(TARGET_RESET, LOW);
  digitalWrite(APL_CTRL, LOW);    // Arduino owns SRAM
  // Phase 2: write the image
  for (size_t i = 0; i < len; i++) sram_write(i, image[i]);
  // Phase 3: hand off
  DDRA = 0x00;                    // tristate Arduino's SRAM data drivers
  digitalWrite(APL_CTRL, HIGH);   // target now owns SRAM
  digitalWrite(BIDIR_EN, HIGH);   // open the comm channel
  digitalWrite(TARGET_RESET, HIGH); // release the target
}
```

15.3 Bidirectional port emulation (RUN mode)

Once the target is running, the Arduino's interaction is entirely through the bidirectional port. Two 74LS374s on the carrier board hold the in-flight bytes; two 74LS74s hold the handshake flags. The Arduino reads and writes these via dedicated GPIO, exactly as the Apple II did via its data-bus reads of $C0n0 and $C0n1.

```
uint8_t bidir_status(void) {
  // Read APL_RD_OK and APL_WR_OK from the carrier board
  return (digitalRead(APL_RD_OK) << 6) | (digitalRead(APL_WR_OK) << 7);
}

void bidir_send(uint8_t b) {
  while (!(bidir_status() & 0x80)) { /* spin until target read */ }
  // Drive byte onto the carrier's "A2-to-VCS" U4 latch input,
  // then pulse the clock to capture it
  DDRA = 0xFF; PORTA = b;
  digitalWrite(A2_TO_VCS_CLK, HIGH);
  delayMicroseconds(1);
  digitalWrite(A2_TO_VCS_CLK, LOW);
  DDRA = 0x00;
}

uint8_t bidir_recv(void) {
  while (!(bidir_status() & 0x40)) { /* spin until target wrote */ }
  // Enable the "VCS-to-A2" U14 latch's output, read the byte, disable
  digitalWrite(VCS_TO_A2_OE, LOW);
  delayMicroseconds(1);
  DDRA = 0x00;
```

```
  uint8_t b = PINA;
  digitalWrite(VCS_TO_A2_OE, HIGH);
  return b;
}
```

15.4 Cartridge capacity

For faithful 1:1 replacement of the original Frob-Apple Card on a VCS or in Frob-26 mode, 4 KB SRAM (e.g., HM6116 × 2 or HM6264 × 1) is sufficient. For ColecoVision development with the larger 8–32 KB CV ROMs the original Frob-Coleco targeted, use a 32 KB SRAM (AS6C62256, 32K × 8, 28-pin DIP, 5 V) with the extra address bits (RAMA12–RAMA14) sourced from the CV's A12–A14.

16. Target-Side Debugger (DBUG.FRB for Z-80)

The Frob-Coleco analog of FMON has not been recovered as a binary, but the recovered FMON.T source on FROB2629.DSK provides the structural template. DBUG.FRB must be Z-80 code, sized ~512 bytes, residing in the top of the cartridge-emulation RAM at $FE00–$FFFF (or wherever the CV-side Frob conventions place it). It implements the same command set ($10, $20, $30, $40) on the same Bit-7/Bit-6 handshake protocol.

16.1 Z-80 source skeleton

```
; DBUG.FRB - Frob-Coleco target-side debugger
; Modeled on FMON.T (Ken Clements, FROBCO, 1/26/83)
; Z-80 port assignments:
PSTAT   EQU 7CH             ; status port
PDATA   EQU 7DH             ; data port (same address, read or write)

        ORG 0FE00H          ; conventional top-of-cart location

BREAK:  ; entry on RST/INT/breakpoint; save state and notify host
        EX  AF,AF'
        EXX
        PUSH AF
        PUSH BC
        PUSH DE
        PUSH HL
        PUSH IX
        PUSH IY
        ; …push remaining state, then send to host
        CALL TXSTATE
        ; fall through to command loop

CI:     CALL RBA            ; read command byte
```

```
        CP  10H
        JR  Z,RM
        CP  20H
        JR  Z,WM
        CP  30H
        JP  Z,GO
        CP  40H
        JR  Z,GOI
        JR  CI              ; unknown; resync

RM:     CALL RBA            ; hi addr
        LD  H,A
        CALL RBA            ; lo addr
        LD  L,A
        CALL RBA            ; count
        LD  B,A
RM1:    LD  A,(HL)
        CALL WBA
        INC HL
        DJNZ RM1
        JR  CI

WM:     CALL RBA            ; hi addr
        LD  H,A
        CALL RBA            ; lo addr
        LD  L,A
        CALL RBA            ; count
        LD  B,A
WM1:    CALL RBA
        LD  (HL),A
        INC HL
        DJNZ WM1
        JR  CI

; …GO and GOI as in FMON, adapted to Z-80 register restore

; Communication helpers — direct ports analog of FMON.T's WBA/RBA

RBA:    IN  A,(PSTAT)       ; wait for read-OK (bit 6)
        AND 40H
        JR  Z,RBA
        IN  A,(PDATA)
        RET

WBA:    PUSH AF             ; save the byte to send
WBA1:   IN  A,(PSTAT)       ; wait for write-OK (bit 7)
        AND 80H
        JR  Z,WBA1
        POP AF
        OUT (PDATA),A
        RET
```

DESIGN — this skeleton is a direct port of FMON.T with 6502-isms replaced by Z-80 idioms. The full break-state save needs the entire Z-80 register set (AF, BC, DE, HL, IX, IY, AF', BC',

DE', HL', SP, PC, I, R), which is more state than FMON saves on the 6502. The host-side `BREAK_EVENT` frame (§14.3) is sized to receive this complete state.

16.2 Breakpoints on the Z-80

The standard mechanism is to replace the target instruction byte with `RST 38H` (one byte, $FF) or any other RST that vectors into DBUG.FRB's BREAK entry. The host saves the original byte in a shadow table; on resume, the original byte is restored before the JR back into target code. This is the same pattern as FMON's 6502 BRK-trap.

17. ColecoVision Expansion Port Interface

The ColecoVision exposes the Z-80 bus through a 60-pin edge connector (Coleco's "expansion port"). The pinout below is the developer-targeted subset most relevant for a Frob-style interface; the full pinout is documented in the Frob User's Manual Appendix A.4.7.

Signal	Direction (CV side)	Pin (subject to verification)	Notes
D0–D7	I/O	(8 pins)	Z-80 data bus
A0–A15	OUT	(16 pins)	Z-80 address bus
/RD, /WR	OUT	(2 pins)	Memory and I/O strobes
/MREQ, /IORQ	OUT	(2 pins)	Memory vs I/O cycle discriminator
/M1	OUT	(1 pin)	Z-80 opcode fetch indicator
/BUSREQ	IN	(1 pin)	Card asserts to halt the Z-80 and take the bus
/BUSACK	OUT	(1 pin)	Z-80 confirms bus has been released
/RESET	IN	(1 pin)	Bidirectional reset line
/NMI, /INT	IN	(2 pins)	Interrupt sources
/WAIT	IN	(1 pin)	Wait-state insertion
+5 V, +12 V, –5 V, GND	POWER	(several pins)	Power rails

Signal	Direction (CV side)	Pin (subject to verification)	Notes
SOUND	IN	(1 pin)	External sound mixing (optional)

The Arduino Mega's 5 V logic levels are directly compatible with the Z-80's TTL levels — no level shifting is required. This is one of the cleanest aspects of the modernised design.

PART IV

Roadmap and Open Items

18. Implementation Roadmap

A practical sequence of milestones for building out the modernised system, in dependency order:

Phase	Milestone	Dependencies	Deliverable
1	USB-serial framing and ACK round-trip	Arduino Mega + USB cable	Working host-↔-Arduino protocol layer
2	32 KB SRAM cartridge-emulator board (passive)	SRAM, expansion port connector, address decoder	Static CV cart-replacement, no debugger
3	Phase 2 + Arduino bus-master access	Phase 1 + 2	Upload cart image from PC to SRAM via Arduino, run unmodified CV cartridge
4	Bidirectional port latches on the breakout PCB	Phase 3 + two 74LS374 + address decoder	CV-side `IN/OUT` to $7C/$7D works
5	DBUG.FRB skeleton on the target	Phase 4 + Z-80 assembler	Target accepts $10/$20/$30/$40 commands
6	Python monitor command set (read/write/go)	Phase 5	Functional debugger at AMON parity

Phase	Milestone	Dependencies	Deliverable
7	Breakpoints and register save/restore	Phase 6	Single-step + register inspection
8	Symbol table, disassembler, GUI	Phase 7	Modern superset features
9	Faithful Frob-Apple Card replica	Independent track; for Apple II owners	Period-correct dev workflow on real Apple II hardware

19. Open Investigations

Several v0.2 investigations have been closed by the Bradley Bell schematic and the retroabandon hwdoc material. The remaining open items are listed below.

19.1 PPC360AN component identification — CLOSED in v0.3

Per `hwdoc/components.md`, the five PPC360AN parts are 74LS367 hex 3-state bus drivers (possibly relabelled Signetics 8T97 or Motorola MC6887, which share the same pinout). Their roles on the board are now documented in §5.1.

19.2 Full bit map of DEV+0 (Frob-Apple) — CLOSED in v0.3

The control register bit map is now CONFIRMED from `memory-map.md` and the U17 connections in the Bradley Bell schematic. See §6.2.

19.3 EPROM socket contents OPEN

The empty 24-pin DIP socket on the photographed Frob-Apple specimen would have held a 2716 or 2732 EPROM. The retroabandon schematic does not wire this socket as part of the active circuit, suggesting it may have been intended for a future feature that was never deployed (a slot-boot PROM was a common Apple II card pattern). Recovery requires locating another card with an EPROM still in the socket, or confirmation from Ken Clements.

19.4 Frob-Coleco card register layout OPEN

It is not yet confirmed whether the Frob-Coleco card uses the identical Frob-Apple register layout or adds functional bits beyond bits 4/5 (e.g., for Z-80-specific addressing modes, 32 KB cartridge addressing, or different page sizes). A photograph of a known-original Frob-Coleco

card would resolve this. The Frob-Coleco User's Manual (FROBCO, 1984) Appendix D and partial Appendix E may provide further detail when fully recovered.

19.5 DBUG.FRB target-side binary OPEN

The Frob User's Manual references a ~512-byte Z-80 binary called DBUG.FRB that runs on the ColecoVision. This binary has not been recovered. The §16 skeleton is a direct structural translation from FMON.T and would serve as a starting point for a modern reimplementation, but recovery of the original binary would be valuable for historical fidelity.

19.6 Bidirectional port latch wiring — CLOSED in v0.3

The exact wiring of U1, U3, U4, and U14 to implement the bidir port's status, write, and read paths is documented in the Bradley Bell schematic, `bidir.kicad_sch` sheet. See §5.2 for the resulting structure.

19.7 Q3 vs Φ0 timing source — OPEN, low priority

The Frob-Apple Card draws timing from Q3 (Apple slot pin 37, 2 MHz asymmetric clock) rather than the more typical Φ0 (pin 40). Both the retroabandon `pinout.md` and the photographed specimen confirm this is the actual wiring, not an error. The motivation is unclear — Q3 has a different phase relationship to memory access than Φ0/Φ1 — but the card demonstrably works as-is, so this is more a historical curiosity than an active investigation.

19.8 Frob-26 vs Frob-Coleco PCB differences — OPEN

Whether the Frob-Coleco used the same physical PCB as the Frob-26 (i.e., the same Frob-Apple Card) with only differences in the cartridge-side board and the target firmware, or whether a separate "Frob-Apple-Coleco" PCB existed, is unresolved. The retroabandon README mentions no surviving Frob-Coleco hardware. If the Frob-26 hardware is reusable for Coleco development with only a different cable-end adapter, that would simplify our Part III modernization significantly.

20. References

1. **Bell, B. (2025).** Frob-Apple Card reverse-engineered KiCad 9 schematic, 12 April 2025. Hosted at `gitlab.com/retroabandon/apple2-re/-/tree/main/frob/kicad`.
2. **Sampson, C., et al. (2025).** retroabandon `apple2-re/frob` hardware documentation: `hwdoc/components.md`, `hwdoc/memory-map.md`, `hwdoc/pinout.md`; documentation

transcriptions: `doc/brochure.md`, `doc/burner.md`, `doc/manual.md`, `doc/frob-52.md`.

3. **retroabandon (2025).** MAME emulation of the FROB, GitHub PR #13596 (pending merge as of v0.3 compilation).
4. **FROB BURNER Preliminary Documentation.** FROBCO, 1983. 30 pp. Includes appendices A–F with BURNER, FBSAVE, and PMOVE source listings. Archived as `frob_burner_document.pdf` on Internet Archive.
5. **Frob User's Manual.** FROBCO, January 1984, 122 pp. Scanned by Bruce Tomlin (Tursi), hosted at atari7800.net/files/frob.pdf.
6. **The Frob-Coleco User's Manual: A True Connoisseur's Guide.** FROBCO, January 1984. Chapters 1–3, Appendices A–C, and partial Appendix E.
7. **Appendix D. ColecoVision Monitor Disassembly.** FROBCO, 90 pp.
8. **The Frob: A Connoisseur's Guide.** FROBCO, 1982. Frob-26 deeper technical manual.
9. **FROB 1.3 Release Notes.** FROBCO, 1983. 2 pp. Notes on a new release of the FROB software.
10. **FROB-26 brochure.** FROBCO, 1982. Marketing brochure; transcribed by retroabandon as `doc/brochure.md`. Lists FROB-26 at $995, FROB BURNER + 2 adapters at $495.
11. **Inside the Atari VCS.** FROBCO, 23 pp. Description of the workings of the 2600.
12. **FROB2629.DSK, FROBDV13.DSK, FRB26211.DSK, FRB26227.DSK, FROBBR10.DSK, FROBBR13.DSK.** Apple II DOS 3.3 disk images, recovered.
13. **FROBCO source listings.** PMOVE.pdf, FMON.pdf, AMON.pdf, EXPLORER.pdf, XCONTROL.pdf, EXPLOR.pdf, FLOAD.pdf, FSAVE.pdf. Re-assembled March 2025.
14. **Cherryhomes, T. (2025).** Frobble: a player-graphics editor for the Frob, dated 03/11/2025. Independent reimplementation used here for cross-validation. Cherryhomes is also the present owner of the only widely-known surviving FROB hardware.
15. **Goodman, D. (1983-08).** "Do-it-yourself games", Radio-Electronics, p. 14. Period magazine description of the FROB-26 (cites $495 price, half the brochure list).
16. **VCF Forums.** "The Frob: It needs some rev-eng love." Thread containing the original .lst and .baa source listings that retroabandon and this project both ingested.
17. **Tomlin, B. (Tursi).** AtariAge community thread on ColecoVision development, November 2006. Source for the Frob PDF scans and the xi6.com toolchain pointers.
18. **Zaks, R.** Programming the Z80 (Sybex, 1981). Reference for Z-80 instruction encoding and bus cycles used in the §16 skeleton.
19. **Mitsubishi Electric.** M58725P datasheet (2K × 8 static RAM).
20. **Hitachi.** HM6116 datasheet (2K × 8 static RAM, M58725P-compatible).
21. **Texas Instruments.** SN54LS366A datasheet (74LS367 hex bus driver, 3-state). Equivalent to PPC360AN.
22. **Intel.** 2732 datasheet (4 KB UV EPROM, 24-pin DIP).

23. **Apple Computer.** Apple II Reference Manual, 1979. Source for the slot peripheral card pinout (§17).

This document is part of the ColecoVisionADAM.com / AdamArchive.org preservation effort. Contributions, corrections, and additional artifacts are welcomed. Direct any feedback or new evidence to the archive maintainer.