# P32W Hardware Reference

The system, the bus and the cards: theory of operation

P32W — Ken Rother's 32-bit TTL computer

Generated 2026-11-13 from the repository at 3cacc22 2026-11-13

# Contents


## P32W — system architecture `docs/architecture.md`

1. The shape of the machine
2. Decisions (D50, Ken accepted all recommendations, 2026-10-24)
3. The cards
4. Physical address map (D52)

4a. The video card as software sees it (D103, D104; docs/video-card-acrtc.md)

5. The CPU on the bus
6. The order of work

## The P32W Bus `docs/bus.md`

The bus cycle

Bus mastership (DMA)

Interrupts

Slots

J1

J2

J3

Signals

## P32W Microarchitecture `docs/microarch.md`

1. Overview
2. Registers and where they live
3. Cards
4. Instructions, cycle by cycle
5. Precise exceptions
6. Microcode
7. The local link (J3, D50)
8. Timing
9. What follows

## The CPU card `docs/cpu-card.md`

1. The cycle
2. Blocks
3. The ALU module (D21)
4. Verification
5. Next

## The CPU card: theory of operation `docs/cpu-card-theory.md`

1. What the card does
2. The three buses
3. The clock: eight phases, one commit edge
4. The control store and the microword
5. The sequencer: choosing the next microaddress
6. The instruction unit
7. The register file

8. The PC
9. The immediate unit
10. The address adder (D59)
11. The comparator (D58)
12. The MDR and the store data
13. The ALU module (D21)
14. STATUS, IMASK and interrupts
15. R's sources and destinations
16. Reset, the front panel and stepping
17. Three instructions, cycle by cycle
18. Timing

## The MMU / data card `docs/mmu-card.md`

1. Blocks
2. How it works
3. Verification
4. Next

## The MMU / data card: theory of operation `docs/mmu-card-theory.md`

1. What the card does
2. The microcycle, seen from this card
3. MAR and the shadow PC
4. The TLB
5. Faults: translate(), in three GALs
6. The bus cycle
7. Load steering
8. The control registers
9. The lock and the arbiter
10. A load, end to end
11. Timing

## The memory card `docs/mem-card.md`

1. What it holds
2. Jumpers (at the card's edge, labelled)
3. How it works
4. Verification

## The memory card: theory of operation `docs/mem-card-theory.md`

1. What the card does
2. The jumpers
3. The decoding: three GALs
4. The datapath
5. A read and a write, in time
6. Fitting a card
7. Verified

## The I/O card `docs/io-card.md`

1. What it holds

2. Jumpers (at the card's edge, labelled)
3. Blocks
4. How it works
5. Verification
6. To check before layout

## The I/O card: theory of operation `docs/io-card-theory.md`

1. What the card does
2. The jumpers
3. The bus interface and the decode
4. A byte access takes two clocks (D72)
5. The UARTs and RS-232
6. The interval timer
7. The status bus and the interrupt status
8. The ports
9. The real-time clock
10. The USB modules
11. The network coprocessor (D45)
12. Interrupts
13. -ACK and -WAIT
14. Two cycles, end to end
15. Timing
16. Verified
17. Open questions

## The disk card `docs/disk-card.md`

1. What it holds
2. Jumpers (at the card's edge, labelled)
3. Blocks
4. How it works
5. Verification
6. To check before layout

## The disk card: theory of operation `docs/disk-card-theory.md`

1. What the card does
2. The ports
3. The jumpers
4. Decode and the byte lanes
5. The port access: DKSEQ, DKPSEL, DKPORT0/1
6. A register access, end to end
7. Interrupts
8. The DMA engine (D75)
9. Verified
10. Open questions

## The video card, revision 2: an HD63484 ACRTC (D103) `docs/video-card-acrtc.md`

1. What the card is
2. What is kept, and where it is

3. The chip's host side, from the manual
4. The bus side on P32W
5. The software (done 2026-11-12, D105)
6. What this card does not do

6a. The picture on the bus as well (asked 2026-11-12; decided: left off, D106)

7. Decided (D104, 2026-11-12; Ken: "agree with all your recommendations")
8. To check
9. The order of work

## The video card `docs/video-card.md`

1. What it holds
2. Jumpers (at the card's edge, labelled)
3. Blocks
4. Theory of operation
5. The font
6. Verification
7. To check before layout

## The video card: theory of operation `docs/video-card-theory.md`

1. What the card does
2. The jumpers
3. The bus interface: five GALs
4. The registers
5. The dot clock and the counters
6. A frame and a line
7. The RAM, its four slots, the text address
8. A character cell, and the VGA output
9. A bus access to the video RAM
10. PS/2
11. Interrupts
12. Timing
13. Verified
14. Open questions

## The debug card (bus tester) `docs/debug-card.md`

1. What it is for
2. Slot 0 (D64)
3. LISTEN: capture
4. DRIVE: playing the master
5. The host link and the rest
6. Size
7. Decided (Ken, 2026-10-26)
8. As built

## The debug card: theory of operation `docs/debug-card-theory.md`

1. What the card does
2. Slot 0
3. The host link

4. The local bus decode
5. The control and status registers
6. LISTEN and DRIVE
7. The slices
8. LISTEN: the capture
9. DRIVE: the drivers
10. The DMA cycle: GAL DBSEQ
11. The clock: GAL DBCLK
12. Using it
13. Timing
14. Verified
15. Open questions

## The backplane `docs/backplane.md`

1. What it holds
2. The parts no card can hold
3. Verification
4. The board (D79)

## The backplane: theory of operation `docs/backplane-theory.md`

1. What the backplane does
2. The slots
3. Bused lines and per-slot lines
4. Requests, grants and the lock
5. The pull-ups
6. The clock
7. Interrupts and the acknowledge cycle
8. Power
9. An empty slot
10. Verified
11. Open questions

## The power budget `docs/power-budget.md`

The cards
The machine
The CPU card
The MMU / data card
The memory card
The I/O card
The disk card
The video card
The debug card

## Timing `docs/timing.md`

1. Results
2. What fails
3. What is not timed, and why
4. Open-drain lines: the release

5. Method and assumptions
6. The datasheets

## Speeding up the CPU: where it can be done `docs/speed.md`

1. The clock
2. The work per cycle
3. In order of value for the effort
4. The benchmark: a clean make of the commands

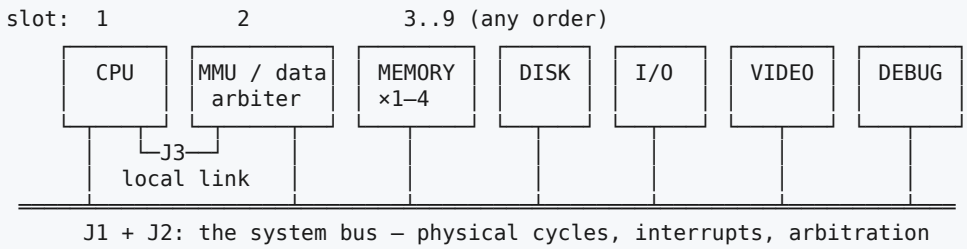
# P32W — system architecture

docs/architecture.md

**Status:** PROPOSED (2026-10-24). The frame is decided (D49: 9U × 340 mm cards, three bus connectors); the choices below are the recommendations Ken accepted the same day (D50). Numbers marked *estimate* are to be confirmed by the netlists.

The software — the ISA, the compilers, P32W Unix, Python, BASIC — runs on the microarchitecture of microarch.md (the microcode and its single internal bus); this document is the machine around it.

## 1. The shape of the machine




- **A system bus** (bus.md): the backplane carries only physical memory cycles — address, data, byte enables, read/write strobes, `-ACK` and `-WAIT` — plus interrupts, bus arbitration, the clock and reset. It is the shape of VME, Multibus or a PDP-11's bus: the CPU's microcode fields never leave its card.
- **The CPU's internals stay on the CPU card.** The one private path is **J3 in slots 1 and 2**: the local link between the CPU card and the MMU / data card, carrying the CPU's result bus R and the memory-cycle fields every microcycle (microarch.md), so they never touch the system bus.
- **The MMU / data card is the CPU's master interface**: it translates, runs the CPU's cycles on the system bus, and holds the **arbiter** that lends the bus to DMA masters.

## 2. Decisions (D50, Ken accepted all recommendations, 2026-10-24)

| # | Question | Decided |
|---|---|---|
| 1 | MMU / data | **a card of its own**, beside the CPU (slot 2), linked by J3. Logic is **GALs + TTL** (no CPLDs) |
| 2 | Clock | **3 MHz guaranteed** at the datasheets' worst case (D72); **4 MHz** with the CPU card's clock jumper, at typical delays (D74) |
| 3 | Bus timing | **synchronous**, one clock, stretched by `-WAIT` |
| 4 | Arbitration, interrupts | **radial** bus request / grant pairs, one per slot, to the arbiter in slot 2; **8 shared priority levels**, each device's level set by a jumper (D52) |
| 5 | Card addresses | **jumpers** on each card, memory and devices alike — nothing depends on the slot (D52) |
| 6 | Slots | **10** (0–9), **1.5 inches apart**, the cards' components facing the front, slot 0's end (D99; 11 at 1 inch before) |
| 7 | Disk | **2 CF + 2 IDE** ports; **PIO first**, the card able to be a DMA master |
| 8 | Video | **VGA** (640 × 480, 31 kHz), one bit a dot, drawn by an **HD63484 ACRTC** from commands (D103, D104); an 80 × 25 console on it; **PS/2 keyboard and mouse** |
| 9 | USB, network | **USB for storage** (two CH376S); **one Pico 2 W + W5500** for Ethernet and Wi-Fi (D45) |
| 10 | ISA | an atomic swap ( `SWAP` , D51) under the bus lock (D54) |

## 3. The cards

| Slot | Card | Contents | ICs (*estimate*) |
|---|---|---|---|
| 1 | **CPU** | the sequencer and control store (microcode flash, pipeline register), the instruction register and decode, immediates, the condition mux, STATUS / interrupts, the 32 × 32 register file, the PC counter, the ALU (the Tang Nano 20K FPGA module, D21), the clock (24 and 32 MHz oscillators, the CLOCK jumper: 3 or 4 MHz, D74). Front edge: RUN / HALT LEDs, RESET, STEP, the RUN / STEP switch, the probe header | ~125 |
| 2 | **MMU / data** | MAR, the split I/D TLB, the fault checks, the MDR and its byte steering, BADADDR; the system-bus master interface (drivers, `-ACK` timeout = BUSERROR, `-WAIT` → `STALL`), the **arbiter** and `-LOCK` | ~100 |
| 3+ | **Memory** (1–4) | **16 MiB** of SRAM (32 × AS6C4008, 512K × 8, 55 ns, DIP-32: eight 2 MiB banks of four byte lanes) and the boot ROM (4 × SST39SF0x0 flash). Jumpers: the card's **16 MiB window** and **ROM enable** (only one card answers for the ROM). 4 cards = 64 MiB | ~50 |
| any | **Disk** | 2 **CompactFlash** sockets (front edge, removable) and 2 **IDE** 40-pin drive ports (2 channels, master + slave each), 16-bit PIO; a DMA engine later (`-BREQ`) | ~30 |
| any | **I/O** | 4 serial ports (16C550, DIP-40; DB9 at the front), the timer (IRQ0), the RTC (DS12C887, D23), 2 USB-A (2 × CH376S, D43), Ethernet + Wi-Fi (Pico 2 W + W5500, D45; RJ45 at the front), an 8-bit input and output port | ~50 |
| any | **Video** | VGA, 640 × 480 × 1, drawn by an HD63484 ACRTC in its own 128K words of memory, none of it on the bus: the CPU sends commands (D103, D104; docs/video-card-acrtc.md, from YACC1-D's V3 design). The console is 80 × 25 characters of 8 × 16 dots on 400 of the 480 lines (Ken chose 25 rows over 30), copied by the chip from a font picture. **PS/2 keyboard and mouse** through an ATmega328P (mini-DIN at the front). Netlist done (D106): 58 ICs, 5 GALs, of which 35 ICs are YACC1-D's V3 card's, checked equal to its netlist; 50 with the character screen's sockets empty. Simulated and timed (D107). Laid out on six layers, its timing checked again with the board's tracks as delays (D108); not ordered. (Revision 0, D67: 88 ICs, 18 GALs, laid out and never built) | 58 |
| 0 | **Debug** (D64, docs/debug-card.md) | in its own slot, which every signal reaches (J3 included): **listen** (a logic analyzer: 32 K samples of all 167 lines, at the bus clock or 12 MHz) or **drive** (the bus master, by DMA at the highest priority; with the CPU and MMU / data cards pulled, the clock, reset and the CPU's side of J3, to test every other card alone), never both, and never driving a line of a CPU or MMU / data card that is in (`-CPUIN`, `-MMUIN`, D102); a two-digit status display (D100); the host link (a Pico 2) | 144 |

Every card follows D48 and the global layout rules: switches, LEDs and cable connectors at the front edge, pointing outwards; jumpers and headers at an edge; references readable with the parts in; every control labelled.

## 4. Physical address map (D52)

Every card is placed by **jumpers**, not by its slot (Ken, D52).

| Range | What | Notes |
|---|---|---|
| `0x0000_0000` – `0x000F_FFFF` | boot ROM (1 MiB) | the memory card with its ROM jumper; hides the RAM under it |
| `0x0000_0000` – `0x03FF_FFFF` | RAM, up to 64 MiB | memory cards in 16 MiB windows (window n = n × 16 MiB), 2 jumpers |
| `0x0400_0000` – `0xFDFF_FFFF` | nothing | BUSERROR |
| `0xFE00_0000` – `0xFE07_FFFF` | video memory | 64 KiB per video card, at `0xFE00_0000 + n × 64 KiB`: kept, but the video card's revision 2 has no memory on the bus (D103), so nothing answers |
| `0xFFFF_8000` – `0xFFFF_FFFF` | device registers | 8 windows of 4 KiB, window n at `0xFFFF_8000 + n × 4 KiB`, 3 jumpers per card |

- The device windows are in the top 32 KiB so that every register is one instruction away, as a sign-extended 16-bit offset from `r0`.
- **A card's registers are the top 256 bytes of its window** (`0xF00` – `0xFFF`). At `0xFF0` every card has a read-only **card ID** (1 I/O, 2 disk, 3 video; 4 was kept for the debug card, which has no registers on the bus: D101), at `0xFF1`

its revision, at `0xFF4` its **interrupt status** (a bit per device). The kernel finds the cards by reading each window's ID; an empty window is a BUSERROR.

- **Default jumpers:**

| Window | Base | Card | Registers |
|---|---|---|---|
| 7 | `0xFFFF_F000` | I/O | UART 0 `FF00`, UART 1 `FF08`, timer `FF10`–`FF18`, ports `FF20`/`FF21`, UART 2 `FF30`, UART 3 `FF38`, RTC `FF40`/`FF41`, USB 0 `FF50`/`FF51`, network `FF52`/`FF53`, USB 1 `FF54`/`FF55` |
| 6 | `0xFFFF_E000` | disk | port p at `EF00 + 16 p`: 0 CF 0, 1 CF 1, 2 IDE A, 3 IDE B |
| 5 | `0xFFFF_D000` | video | the chip's two registers at `DF00` / `DF04`, PS/2 at `DF08` / `DF0C` |
| 0–4 | | free | a second I/O card, ... (window 4 was kept for the debug card, which answers no bus address: D101) |

- **Interrupt levels** are jumpers too (devices may share one): 0 timer, 1 disk, 2 UART 0 and 2, 3 UART 1 and 3, 4 PS/2, 5 RTC, 6 USB, 7 network.

User mode traps at `0x8000_0000` and above (isa-system.md C2), so devices are supervisor-only.

## 4a. The video card as software sees it (D103, D104; docs/video-card-acrtc.md)

Revision 2: Hitachi's **HD63484 ACRTC** draws VGA, 640 × 480 at 60 Hz, one bit a dot, in its own memory (128K words; the font EPROM in the upper half of its address space). **Nothing of the picture is on the bus.** The card is a register page, window 5 by default (`0xFFFF_DF00`):

| Address | Size | Read | Write |
|---|---|---|---|
| `DF00` | halfword | the chip's status (bits 7–0) | the chip's address register |
| `DF04` | halfword | the register addressed, or the read FIFO | the register addressed, or the write FIFO |
| `DF08` | byte | card status: bit 0 FULL (a PS/2 byte waits), 1 SRC (the mouse's), 2 BUSY, 3 TEXT, 4 the chip's IRQ | a command to the PS/2 section |
| `DF0C` | byte | the PS/2 byte; reading it clears FULL | — |
| `DFF0` / `DFF1` / `DFF4` | byte / byte / word | card ID 3, revision 2, interrupt status (bit 0 the PS/2 section, 1 the chip) | — |

- **Drawing** is by the chip's commands, 16-bit words through its write FIFO (eight words): clear, copy, line, rectangle, fill, a word read or written. An access to the chip takes four clocks (three waits); a PS/2 command written, two (D106).
- **The console** (D53) is 25 rows of 80 characters on lines 40–439: a character is one graphic copy from a picture of the font under the screen; bold, underline and inverse are more pictures; a scroll is one copy by the chip, about 30 ms.
- **Two interrupts**, both level 4 by default: the PS/2 section (vector `0x50`), and the chip (`0x51`: "write FIFO empty" paces the console's output).
- **The keyboard** arrives as ASCII and VT100 sequences from the card's ATmega328P; the mouse's packets whole.

The full register list is in the programmer's guide §6.6. The earlier revisions, never built: revision 0 (D67, video-card.md), text and an overlay in 64 KiB on the bus; revision 1 (D98), a 6845 card, a description only.

## 5. The CPU on the bus

- **The microcode does not depend on the bus's speed.** A memory microcycle runs one system-bus cycle; `STALL` from the MMU / data card freezes the sequencer while the cycle waits (`-WAIT`) or the bus is lent to a DMA master. No instruction sequence changes length on a slow card.
- **Atomic swap** (D51, D54). `SWAP rd, imm(rs1)` exchanges a register with a memory word; its read and write are one bus tenure under `-LOCK`, for locks the kernel shares with DMA masters and, later, other processors.
- **Page swapping** needs no further hardware: faults are restartable (D6), the software-managed TLB tells the kernel a page is used (a TLB miss) and dirty (a write to a page mapped read-only). The per-process space (8 MiB, D34) can grow with the memory.
- **5 MHz.** The memory microcycle (MAR → TLB → physical address → memory → MDR, about 200 ns) does not fit a 200 ns clock with the control store's share. Either the MMU / data card asks for one wait state on every access (`-WAIT` on itself — no microcode change), or translation moves to the cycle that loads MAR. 3 MHz needs neither.

  **Measured** (`p32uemu -W N`, 2026-10-25: P32W Unix booted, root logged in, `halt`; 7,490,575 instructions, on the three-bus datapath of microarch.md, D56). 40% of all microcycles are bus cycles (fetches, loads, stores), so one wait state on each adds 40% to the cycle count:

| Clock, waits | Cycles | Cycles / instruction | Speed |
|---|---|---|---|
| 3 MHz, 0 | 22,427,463 | 3.18 | 0.94 MIPS |
| 5 MHz, 1 on every bus cycle | 32,868,985 | 4.66 | **1.07 MIPS** (+14%) |
| 5 MHz, 0 (translation moved earlier) | 22,427,463 | 3.18 | 1.57 MIPS (+67%) |

  (With the address adder and the D59 instructions: 7,049,876 instructions; 47% of all microcycles are bus cycles. Before them the run was 7,490,575 instructions, 26,738,361 cycles at 3 MHz: the same work is now 1.19× faster at every clock.)

  The wait state is the safe first step (no microcode change); moving the translation is worth another 40% later.

## 6. The order of work

1. **This document and the bus** (`generators/p32bus.py`). Done.
2. **The emulators and P32W Unix on this map** (D52): jumpered device windows and card IDs, 16 MiB memory cards, four UARTs, two USB ports, the disk card's six drives, the video console (D53), `SWAP`. Done.
3. **The microcode and the card-level emulator** (`p32uemu.c`): the bus lock (D54), wait states and a DMA master; the three-bus datapath (D56), 3.57 cycles an instruction; the address adder and indexed / post-increment loads and stores and `LI32` (D59), 3.18. Done.
4. **Card netlists**, the CPU card first, then MMU / data, memory; then the backplane (radial lines, D68: docs/backplane.md), then the device cards. Netlists done; schematics and layouts next.

# The P32W Bus

docs/bus.md

**Status:** IN USE: every card's netlist and the backplane are generated from this definition and checked against it (D50, D52, D57, D59, D64, D68, D74, D76; last change 2026-11-11, D102: -CPUIN and -MMUIN on two of J3's spare pins).

*Generated by* `generators/p32bus.py` *— edit the generator, not this file.*

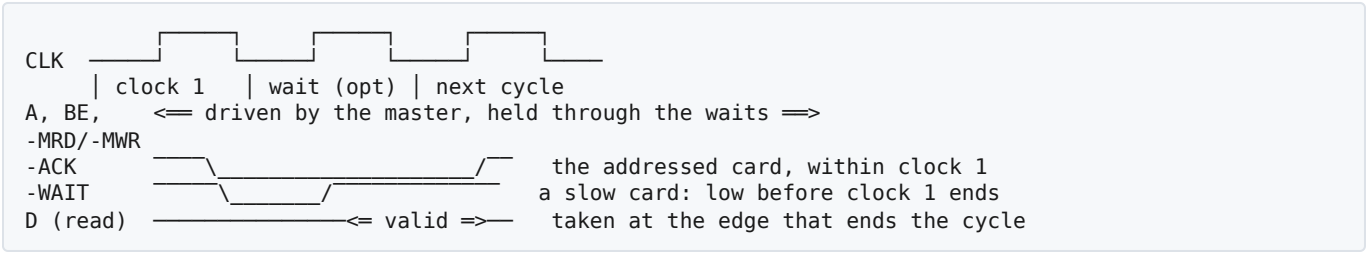
Three 96-pin DIN 41612 connectors per card (D49), down the 366.7 mm edge of a 9U × 340 mm card: **J1** the bus cycle (data, physical address, byte enables, strobes, `-ACK`, `-WAIT`), **J2** system control (clock, reset, interrupts, arbitration), **J3** the CPU ↔ MMU/data local link in slots 1 and 2 and the debug slot 0 (power only in the others). Pins 1–2 of every row are VCC, 31–32 GND; row B odd pins 3–29 are GND (a guard between rows A and C); J2's row B even pins 4–30 are VCC as well (D57), so J2 carries alternate supply pins down its middle. Active-low signals have a leading dash; `-ACK`, `-WAIT`, `-LOCK` and `-IRQn` are open-drain with backplane pull-ups. Spare pins on J1 and J2 are bused to every slot (D68); J3's eight (B28, B30, C25–C30) join slots 0, 1 and 2, as the local link does (D99): spare private lines. The backplane itself: backplane.md.

The backplane does **not** carry the CPU's microcode fields: only physical memory cycles, so any card can be a bus master (DMA), any number of memory cards can answer, and a slow card can stretch a cycle.

**Pins (per card):** 171 signal, 32 VCC, 60 GND, 25 spare (of 288).

## The bus cycle

The bus is **synchronous**: one clock (`CLK`, the CPU's microcycle: 3 MHz, or 4 MHz with the CPU card's clock jumper, D72, D74) for every card. A cycle is one clock, stretched by `-WAIT`:




1. **The master** drives `A2–A31`, `BE0–BE3` and `-MRD` or `-MWR` from the start of the clock (on a write, `D` too).
2. **The addressed card** pulls `-ACK` low within the first clock: address decoding is fast and always fits. No `-ACK` by the end of the first clock is a **BUSERROR**, so a missing card can never hang the bus.
3. **A slow card** also pulls `-WAIT` low before the clock ends. The master holds everything for another clock, and checks `-WAIT` again; when it is high at the end of a clock, that edge ends the cycle (a read's data is taken then; a write is complete).
4. **The CPU** sees a waiting cycle as `STALL` on the local link: its clock `CCLK` has no edge until the cycle ends, so nothing on the CPU side changes. The microcode does not change for slower cards.

## Bus mastership (DMA)

The **arbiter is on the MMU / data card** (slot 2), which is the CPU's own master interface. Any card in slots 3–10 may become master:

- It pulls its radial `-BREQ` low. The arbiter grants at a cycle boundary, when the CPU's next microcycle does not use the bus (most do not: about two in six are memory cycles), and holds the CPU (`STALL`) only when the CPU wants the bus while a device has it.
- `-BGNT` low means the card is master **from the next clock**. It runs as many cycles as it likes and gives the bus back by releasing `-BREQ`; the arbiter then drops `-BGNT`.
- **Priority:** fixed, among the requests present at the boundary: slot 0 (the debug card, D64) first, then the higher slot first.
- `-LOCK`: a master that holds `-LOCK` low keeps the bus for the next cycle too; the CPU uses it for an atomic read-modify-write (`LOCKREQ` on the local link), and the arbiter grants nothing while it is low.
- A DMA master addresses **physical** memory; it has no MMU. The kernel gives it physical buffers (and pins their pages while the transfer runs).

## Interrupts

- `-IRQ0`–`-IRQ7`, open-drain levels into the CPU's `IMASK` / `IPEND` logic; the lowest-numbered pending, enabled level is taken first (`IRQ0`, the timer, has the highest priority).
- Each card's **jumpers** choose the level each of its devices drives (D52). Nothing about an interrupt depends on the slot a card is in.
- **Vectored (D57).** When the CPU takes level *n*, the MMU / data card runs an **interrupt acknowledge cycle**: a bus cycle like a read, with `-IACK` low instead of `-MRD` and `A4:A2` = *n* (the other address bits and `BE0`–`BE3` don't matter). The card driving level *n* pulls `-ACK` and puts its **vector** on `D7:D0`: its window number (D52) × 16 + the number of its device that is asking (its interrupt status register's bit; the lowest pending one on that level). A level is driven by **one card** — the jumpers of two cards must not share one — though one card may drive several. No `-ACK` (no card, or one without vectors) is not a bus error: the vector is `0xFF`, and the kernel reads the cards' interrupt status registers instead. The CPU puts the vector into `CAUSE` bits 15:8.

## Slots

| Slot | Card | J3 |
|---|---|---|
| 0 | the debug card (D64): every signal, to listen or drive | the local link, a third drop |
| 1 | CPU | the local link |
| 2 | MMU / data (and the arbiter) | the local link |
| 3–10 | any: memory, disk, I/O, video | power only |

**The backplane has ten positions, slots 0–9, 1.5 inches apart** (D99, docs/backplane.md): the bus's slot 10 has no socket there, and its request pair is unused. Cards go in with their components facing slot 0's end, the front.

**Slot 0** carries everything: J1 and J2 as every slot, the radial `-BREQ3–10` / `-BGNT3–10` as slot 2 sees them (to watch or play every requester), its own pair `-BREQ0` / `-BGNT0` (the arbiter's highest priority), and J3. The debug card there can watch a running CPU's local link, or, with the CPU card pulled, play the CPU's side of it.

Every card is placed by its **jumpers**, not its slot (D50, D52): memory cards choose a 16 MiB window, device cards a 4 KiB window among the device registers, and their interrupt levels. Apart from slots 1 and 2 (the local link), any card works in any slot; the slot only sets its bus-request priority.

## J1

| Pin | Row A | Row B | Row C |
|---|---|---|---|
| 1 | VCC | VCC | VCC |
| 2 | VCC | VCC | VCC |
| 3 | D0 | GND | D28 |
| 4 | D1 | A26 | D29 |
| 5 | D2 | GND | D30 |
| 6 | D3 | A27 | D31 |
| 7 | D4 | GND | A2 |
| 8 | D5 | A28 | A3 |
| 9 | D6 | GND | A4 |
| 10 | D7 | A29 | A5 |
| 11 | D8 | GND | A6 |
| 12 | D9 | A30 | A7 |
| 13 | D10 | GND | A8 |
| 14 | D11 | A31 | A9 |
| 15 | D12 | GND | A10 |
| 16 | D13 | BE0 | A11 |
| 17 | D14 | GND | A12 |
| 18 | D15 | BE1 | A13 |
| 19 | D16 | GND | A14 |
| 20 | D17 | BE2 | A15 |
| 21 | D18 | GND | A16 |
| 22 | D19 | BE3 | A17 |
| 23 | D20 | GND | A18 |
| 24 | D21 | -MRD | A19 |
| 25 | D22 | GND | A20 |
| 26 | D23 | -MWR | A21 |
| 27 | D24 | GND | A22 |
| 28 | D25 | -ACK | A23 |
| 29 | D26 | GND | A24 |
| 30 | D27 | -WAIT | A25 |
| 31 | GND | GND | GND |
| 32 | GND | GND | GND |

## J2

| Pin | Row A | Row B | Row C |
|---|---|---|---|
| 1 | VCC | VCC | VCC |

| Pin | Row A | Row B | Row C |
|---|---|---|---|
| 2 | VCC | VCC | VCC |
| 3 | CLK | GND | -BREQ3 |
| 4 | -CLK | VCC | -BREQ4 |
| 5 | -RES | GND | -BREQ5 |
| 6 | PWROK | VCC | -BREQ6 |
| 7 | -LOCK | GND | -BREQ7 |
| 8 | -IRQ0 | VCC | -BREQ8 |
| 9 | -IRQ1 | GND | -BREQ9 |
| 10 | -IRQ2 | VCC | -BREQ10 |
| 11 | -IRQ3 | GND | -BGNT3 |
| 12 | -IRQ4 | VCC | -BGNT4 |
| 13 | -IRQ5 | GND | -BGNT5 |
| 14 | -IRQ6 | VCC | -BGNT6 |
| 15 | -IRQ7 | GND | -BGNT7 |
| 16 | -BREQ | VCC | -BGNT8 |
| 17 | -BGNT | GND | -BGNT9 |
| 18 | -IACK | VCC | -BGNT10 |
| 19 | -FAST | GND | -BREQ0 |
| 20 | XCLK | VCC | -BGNT0 |
| 21 | -XCLKEN | GND | SPARE_J2C21 |
| 22 | SPARE_J2A22 | VCC | SPARE_J2C22 |
| 23 | SPARE_J2A23 | GND | SPARE_J2C23 |
| 24 | SPARE_J2A24 | VCC | SPARE_J2C24 |
| 25 | SPARE_J2A25 | GND | SPARE_J2C25 |
| 26 | SPARE_J2A26 | VCC | SPARE_J2C26 |
| 27 | SPARE_J2A27 | GND | SPARE_J2C27 |
| 28 | SPARE_J2A28 | VCC | SPARE_J2C28 |
| 29 | SPARE_J2A29 | GND | SPARE_J2C29 |
| 30 | SPARE_J2A30 | VCC | SPARE_J2C30 |
| 31 | GND | GND | GND |
| 32 | GND | GND | GND |

## J3

| Pin | Row A | Row B | Row C |
|---|---|---|---|
| 1 | VCC | VCC | VCC |
| 2 | VCC | VCC | VCC |
| 3 | LR0 | GND | LR28 |

| Pin | Row A | Row B | Row C |
|---|---|---|---|
| 4 | LR1 | FAULT | LR29 |
| 5 | LR2 | GND | LR30 |
| 6 | LR3 | FC0 | LR31 |
| 7 | LR4 | GND | RSRC0 |
| 8 | LR5 | FC1 | RSRC1 |
| 9 | LR6 | GND | RSRC2 |
| 10 | LR7 | FC2 | RSRC3 |
| 11 | LR8 | GND | RDST0 |
| 12 | LR9 | FC3 | RDST1 |
| 13 | LR10 | GND | RDST2 |
| 14 | LR11 | -WSTB | MEM0 |
| 15 | LR12 | GND | MEM1 |
| 16 | LR13 | STALL | MEM2 |
| 17 | LR14 | GND | SIZE0 |
| 18 | LR15 | LOCKREQ | SIZE1 |
| 19 | LR16 | GND | SIGNED |
| 20 | LR17 | CCLK | CRSEL0 |
| 21 | LR18 | GND | CRSEL1 |
| 22 | LR19 | PCINC | CRSEL2 |
| 23 | LR20 | GND | SUPER |
| 24 | LR21 | MARPC | TLBW |
| 25 | LR22 | GND | SPARE_J3C25 |
| 26 | LR23 | MARINC | SPARE_J3C26 |
| 27 | LR24 | GND | SPARE_J3C27 |
| 28 | LR25 | -CPUIN | SPARE_J3C28 |
| 29 | LR26 | GND | SPARE_J3C29 |
| 30 | LR27 | -MMUIN | SPARE_J3C30 |
| 31 | GND | GND | GND |
| 32 | GND | GND | GND |

## Signals

| Signal | Pin | Wired to | Driven by | Read by | Meaning |
|---|---|---|---|---|---|
| `D0` | J1 A3 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D1` | J1 A4 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D2` | J1 A5 | every slot | the master (write) or the addressed card (read) | all | data bus |

| Signal | Pin | Wired to | Driven by | Read by | Meaning |
|---|---|---|---|---|---|
| `D3` | J1 A6 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D4` | J1 A7 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D5` | J1 A8 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D6` | J1 A9 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D7` | J1 A10 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D8` | J1 A11 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D9` | J1 A12 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D10` | J1 A13 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D11` | J1 A14 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D12` | J1 A15 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D13` | J1 A16 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D14` | J1 A17 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D15` | J1 A18 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D16` | J1 A19 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D17` | J1 A20 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D18` | J1 A21 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D19` | J1 A22 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D20` | J1 A23 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D21` | J1 A24 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D22` | J1 A25 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D23` | J1 A26 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D24` | J1 A27 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D25` | J1 A28 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D26` | J1 A29 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D27` | J1 A30 | every slot | the master (write) or the addressed card (read) | all | data bus |

| Signal | Pin | Wired to | Driven by | Read by | Meaning |
|---|---|---|---|---|---|
| `D28` | J1 C3 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D29` | J1 C4 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D30` | J1 C5 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `D31` | J1 C6 | every slot | the master (write) or the addressed card (read) | all | data bus |
| `A2` | J1 C7 | every slot | the master | all | physical address bits 31:2 |
| `A3` | J1 C8 | every slot | the master | all | physical address bits 31:2 |
| `A4` | J1 C9 | every slot | the master | all | physical address bits 31:2 |
| `A5` | J1 C10 | every slot | the master | all | physical address bits 31:2 |
| `A6` | J1 C11 | every slot | the master | all | physical address bits 31:2 |
| `A7` | J1 C12 | every slot | the master | all | physical address bits 31:2 |
| `A8` | J1 C13 | every slot | the master | all | physical address bits 31:2 |
| `A9` | J1 C14 | every slot | the master | all | physical address bits 31:2 |
| `A10` | J1 C15 | every slot | the master | all | physical address bits 31:2 |
| `A11` | J1 C16 | every slot | the master | all | physical address bits 31:2 |
| `A12` | J1 C17 | every slot | the master | all | physical address bits 31:2 |
| `A13` | J1 C18 | every slot | the master | all | physical address bits 31:2 |
| `A14` | J1 C19 | every slot | the master | all | physical address bits 31:2 |
| `A15` | J1 C20 | every slot | the master | all | physical address bits 31:2 |
| `A16` | J1 C21 | every slot | the master | all | physical address bits 31:2 |
| `A17` | J1 C22 | every slot | the master | all | physical address bits 31:2 |
| `A18` | J1 C23 | every slot | the master | all | physical address bits 31:2 |
| `A19` | J1 C24 | every slot | the master | all | physical address bits 31:2 |
| `A20` | J1 C25 | every slot | the master | all | physical address bits 31:2 |
| `A21` | J1 C26 | every slot | the master | all | physical address bits 31:2 |
| `A22` | J1 C27 | every slot | the master | all | physical address bits 31:2 |
| `A23` | J1 C28 | every slot | the master | all | physical address bits 31:2 |
| `A24` | J1 C29 | every slot | the master | all | physical address bits 31:2 |
| `A25` | J1 C30 | every slot | the master | all | physical address bits 31:2 |
| `A26` | J1 B4 | every slot | the master | all | physical address bits 31:2 |
| `A27` | J1 B6 | every slot | the master | all | physical address bits 31:2 |
| `A28` | J1 B8 | every slot | the master | all | physical address bits 31:2 |
| `A29` | J1 B10 | every slot | the master | all | physical address bits 31:2 |
| `A30` | J1 B12 | every slot | the master | all | physical address bits 31:2 |
| `A31` | J1 B14 | every slot | the master | all | physical address bits 31:2 |
| `BE0` | J1 B16 | every slot | the master | all | byte enables (lanes 0-3; lane n = D[8n+7:8n]) |
| `BE1` | J1 B18 | every slot | the master | all | byte enables (lanes 0-3; lane n = D[8n+7:8n]) |
| `BE2` | J1 B20 | every slot | the master | all | byte enables (lanes 0-3; lane n = D[8n+7:8n]) |
| `BE3` | J1 B22 | every slot | the master | all | byte enables (lanes 0-3; lane n = D[8n+7:8n]) |

| Signal | Pin | Wired to | Driven by | Read by | Meaning |
|---|---|---|---|---|---|
| `-MRD` | J1 B24 | every slot | the master | all | read cycle: the addressed card drives D |
| `-MWR` | J1 B26 | every slot | the master | all | write cycle: the addressed card takes D (enabled lanes) |
| `-ACK` | J1 B28 | every slot | the addressed card | the master | open-drain: a card decodes A, in the cycle's first clock; none -> BUSERROR |
| `-WAIT` | J1 B30 | every slot | the addressed card | the master | open-drain: stretch this cycle by one clock while low (a wait state) |
| `CLK` | J2 A3 | every slot | CPU | all | the system clock (the microcycle) and its inverse |
| `-CLK` | J2 A4 | every slot | CPU | all | the system clock (the microcycle) and its inverse |
| `-RES` | J2 A5 | every slot | CPU | all | reset |
| `PWROK` | J2 A6 | every slot | backplane | CPU | power good (the backplane's +5 V supervisor, D78); low = reset, falling = power failing |
| `-LOCK` | J2 A7 | every slot | the master | arbiter | open-drain: keep the bus for the next cycle too (an atomic read-modify-write) |
| `-IRQ0` | J2 A8 | every slot | devices (and the debug card in DRIVE, D101) | CPU | open-drain interrupt request levels (IRQ0 = timer, the highest priority: the lowest-numbered is taken first) |
| `-IRQ1` | J2 A9 | every slot | devices (and the debug card in DRIVE, D101) | CPU | open-drain interrupt request levels (IRQ0 = timer, the highest priority: the lowest-numbered is taken first) |
| `-IRQ2` | J2 A10 | every slot | devices (and the debug card in DRIVE, D101) | CPU | open-drain interrupt request levels (IRQ0 = timer, the highest priority: the lowest-numbered is taken first) |
| `-IRQ3` | J2 A11 | every slot | devices (and the debug card in DRIVE, D101) | CPU | open-drain interrupt request levels (IRQ0 = timer, the highest priority: the lowest-numbered is taken first) |
| `-IRQ4` | J2 A12 | every slot | devices (and the debug card in DRIVE, D101) | CPU | open-drain interrupt request levels (IRQ0 = timer, the highest priority: the lowest-numbered is taken first) |
| `-IRQ5` | J2 A13 | every slot | devices (and the debug card in DRIVE, D101) | CPU | open-drain interrupt request levels (IRQ0 = timer, the highest priority: the lowest-numbered is taken first) |
| `-IRQ6` | J2 A14 | every slot | devices (and the debug card in DRIVE, D101) | CPU | open-drain interrupt request levels (IRQ0 = timer, the highest priority: the lowest-numbered is taken first) |
| `-IRQ7` | J2 A15 | every slot | devices (and the debug card in DRIVE, D101) | CPU | open-drain interrupt request levels (IRQ0 = timer, the highest priority: the lowest-numbered is taken first) |
| `-IACK` | J2 A18 | every slot | the master (the MMU / data card for the CPU) | devices | interrupt acknowledge cycle (D57): instead of -MRD, with A4:A2 = the level; the card driving that level answers -ACK and puts its vector on D7:0 |
| `XCLK` | J2 A20 | every slot | the debug card (slot 0) | CPU | an external oscillator for the CPU card (D76): the debug card's 12 MHz, or one edge per host STEP; the CPU card takes it while -XCLKEN is low |
| `-XCLKEN` | J2 A21 | every slot | the debug card (slot 0) | CPU | low: the CPU card's oscillator is XCLK (D76); with no debug card the CPU card's pull-up holds it high |
| `-FAST` | J2 A19 | every slot | the CPU card's clock jumper (fitted = low); the debug card alone (D101) | CPU, I/O | low: the 4 MHz clock (D74), the I/O card's timer dividing by 4; open (the backplane's pull-up): 3 MHz. Set with the power off |
| `-BREQ` | J2 A16 | slots 0, 3–10 (radial) | the card | arbiter | this slot asks for the bus (radial) |
| `-BGNT` | J2 A17 | slots 0, 3–10 (radial) | arbiter | the card | this slot is master from the next clock (radial) |
| `-BREQ3` | J2 C3 | slots 2 and 0 (radial) | slot n's card | arbiter | slot n's -BREQ |
| `-BREQ4` | J2 C4 | slots 2 and 0 (radial) | slot n's card | arbiter | slot n's -BREQ |

| Signal | Pin | Wired to | Driven by | Read by | Meaning |
|---|---|---|---|---|---|
| `-BREQ5` | J2 C5 | slots 2 and 0 (radial) | slot n's card | arbiter | slot n's -BREQ |
| `-BREQ6` | J2 C6 | slots 2 and 0 (radial) | slot n's card | arbiter | slot n's -BREQ |
| `-BREQ7` | J2 C7 | slots 2 and 0 (radial) | slot n's card | arbiter | slot n's -BREQ |
| `-BREQ8` | J2 C8 | slots 2 and 0 (radial) | slot n's card | arbiter | slot n's -BREQ |
| `-BREQ9` | J2 C9 | slots 2 and 0 (radial) | slot n's card | arbiter | slot n's -BREQ |
| `-BREQ10` | J2 C10 | slots 2 and 0 (radial) | slot n's card | arbiter | slot n's -BREQ |
| `-BGNT3` | J2 C11 | slots 2 and 0 (radial) | arbiter (the debug card alone, D101) | slot n's card | slot n's -BGNT |
| `-BGNT4` | J2 C12 | slots 2 and 0 (radial) | arbiter (the debug card alone, D101) | slot n's card | slot n's -BGNT |
| `-BGNT5` | J2 C13 | slots 2 and 0 (radial) | arbiter (the debug card alone, D101) | slot n's card | slot n's -BGNT |
| `-BGNT6` | J2 C14 | slots 2 and 0 (radial) | arbiter (the debug card alone, D101) | slot n's card | slot n's -BGNT |
| `-BGNT7` | J2 C15 | slots 2 and 0 (radial) | arbiter (the debug card alone, D101) | slot n's card | slot n's -BGNT |
| `-BGNT8` | J2 C16 | slots 2 and 0 (radial) | arbiter (the debug card alone, D101) | slot n's card | slot n's -BGNT |
| `-BGNT9` | J2 C17 | slots 2 and 0 (radial) | arbiter (the debug card alone, D101) | slot n's card | slot n's -BGNT |
| `-BGNT10` | J2 C18 | slots 2 and 0 (radial) | arbiter (the debug card alone, D101) | slot n's card | slot n's -BGNT |
| `-BREQ0` | J2 C19 | slots 2 and 0 (radial) | the debug card (slot 0) | arbiter | slot 0's -BREQ: the highest priority (D64) |
| `-BGNT0` | J2 C20 | slots 2 and 0 (radial) | arbiter | the debug card (slot 0) | slot 0's -BGNT |
| `LR0` | J3 A3 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR1` | J3 A4 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR2` | J3 A5 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR3` | J3 A6 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR4` | J3 A7 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR5` | J3 A8 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |

| Signal | Pin | Wired to | Driven by | Read by | Meaning |
|---|---|---|---|---|---|
| `LR6` | J3 A9 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR7` | J3 A10 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR8` | J3 A11 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR9` | J3 A12 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR10` | J3 A13 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR11` | J3 A14 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR12` | J3 A15 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR13` | J3 A16 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR14` | J3 A17 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR15` | J3 A18 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR16` | J3 A19 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR17` | J3 A20 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR18` | J3 A21 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR19` | J3 A22 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR20` | J3 A23 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR21` | J3 A24 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR22` | J3 A25 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |

| Signal | Pin | Wired to | Driven by | Read by | Meaning |
|---|---|---|---|---|---|
| `LR23` | J3 A26 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR24` | J3 A27 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR25` | J3 A28 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR26` | J3 A29 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR27` | J3 A30 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR28` | J3 C3 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR29` | J3 C4 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR30` | J3 C5 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `LR31` | J3 C6 | slots 0–2 only | CPU or MMU/data (RSRC) | CPU, MMU/data | the CPU's result bus R, extended to the MMU / data card: addresses to MAR, store data out, load data (steered), BADADDR and the MMU's control registers in |
| `RSRC0` | J3 C7 | slots 0–2 only | CPU | MMU/data | R's source, encoded (microword RSRC) |
| `RSRC1` | J3 C8 | slots 0–2 only | CPU | MMU/data | R's source, encoded (microword RSRC) |
| `RSRC2` | J3 C9 | slots 0–2 only | CPU | MMU/data | R's source, encoded (microword RSRC) |
| `RSRC3` | J3 C10 | slots 0–2 only | CPU | MMU/data | R's source, encoded (microword RSRC) |
| `RDST0` | J3 C11 | slots 0–2 only | CPU | MMU/data | R's destination, encoded (microword RDST) |
| `RDST1` | J3 C12 | slots 0–2 only | CPU | MMU/data | R's destination, encoded (microword RDST) |
| `RDST2` | J3 C13 | slots 0–2 only | CPU | MMU/data | R's destination, encoded (microword RDST) |
| `MEM0` | J3 C14 | slots 0–2 only | CPU | MMU/data | memory cycle type (microword MEM) |
| `MEM1` | J3 C15 | slots 0–2 only | CPU | MMU/data | memory cycle type (microword MEM) |
| `MEM2` | J3 C16 | slots 0–2 only | CPU | MMU/data | memory cycle type (microword MEM) |
| `SIZE0` | J3 C17 | slots 0–2 only | CPU | MMU/data | access size and sign-extension (from IR) |
| `SIZE1` | J3 C18 | slots 0–2 only | CPU | MMU/data | access size and sign-extension (from IR) |

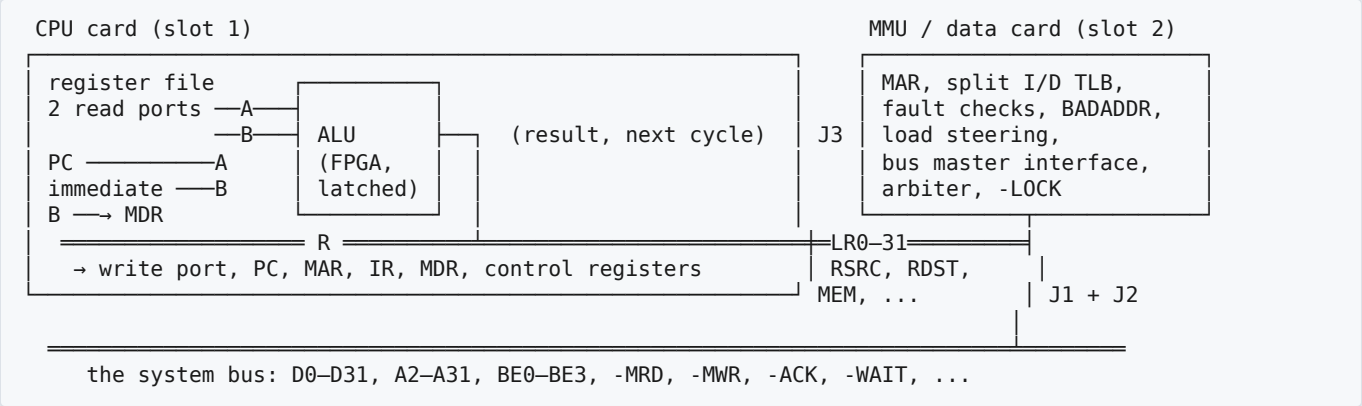
| Signal | Pin | Wired to | Driven by | Read by | Meaning |
|---|---|---|---|---|---|
| SIGNED | J3 C19 | slots 0–2 only | CPU | MMU/data | access size and sign-extension (from IR) |
| SUPER | J3 C23 | slots 0–2 only | CPU | MMU/data | STATUS.S: 1 = supervisor (untranslated) |
| TLBW | J3 C24 | slots 0–2 only | CPU | MMU/data | write TLBHI/TLBLO into the TLB |
| CRSEL0 | J3 C20 | slots 0–2 only | CPU | MMU/data | hardware control register for RSRC / RDST = CRHW |
| CRSEL1 | J3 C21 | slots 0–2 only | CPU | MMU/data | hardware control register for RSRC / RDST = CRHW |
| CRSEL2 | J3 C22 | slots 0–2 only | CPU | MMU/data | hardware control register for RSRC / RDST = CRHW |
| FAULT | J3 B4 | slots 0–2 only | MMU/data | CPU | this memory cycle faulted: abort it, enter trap |
| FC0 | J3 B6 | slots 0–2 only | MMU/data | CPU | fault cause |
| FC1 | J3 B8 | slots 0–2 only | MMU/data | CPU | fault cause |
| FC2 | J3 B10 | slots 0–2 only | MMU/data | CPU | fault cause |
| FC3 | J3 B12 | slots 0–2 only | MMU/data | CPU | fault cause |
| -WSTB | J3 B14 | slots 0–2 only | CPU | MMU/data | write strobe: gates every SRAM write |
| STALL | J3 B16 | slots 0–2 only | MMU/data | CPU | hold the microcycle: the bus is not yet ours (arbitration) or the cycle is waiting (-WAIT) |
| LOCKREQ | J3 B18 | slots 0–2 only | CPU | MMU/data | this access and the next are one atomic pair (-LOCK) |
| PCINC | J3 B22 | slots 0–2 only | CPU | MMU/data | the PC's +4 and MAR <- PC: the MMU / data card keeps a shadow PC (it loads from R with the PC and counts with it), so the next fetch's address needs no R cycle |
| MARPC | J3 B24 | slots 0–2 only | CPU | MMU/data | the PC's +4 and MAR <- PC: the MMU / data card keeps a shadow PC (it loads from R with the PC and counts with it), so the next fetch's address needs no R cycle |
| MARINC | J3 B26 | slots 0–2 only | CPU | MMU/data | MAR += 4 (MAR is a counter): LI32's value word, then the next fetch (D59) |
| CCLK | J3 B20 | slots 0–2 only | CPU | MMU/data | the CPU's clock: CLK's cycles in which the CPU commits (none while STALL, halted or between steps) |
| -CPUIN | J3 B28 | slots 0–2 only | the CPU card: the pin tied to its ground | the debug card (slot 0) | low: a CPU card is in slot 1 (D102). While it is low the debug card drives nothing alone: no CLK, no -RES, none of its groups |
| -MMUIN | J3 B30 | slots 0–2 only | the MMU / data card: the pin tied to its ground | the debug card (slot 0) | low: an MMU / data card is in slot 2 (D102). While it is low the debug card alone leaves that card's lines to it: D, the bus cycle's lines and the grants |

# P32W Microarchitecture

docs/microarch.md

**Status:** PROPOSED (2026-10-25, D55, D56), running in the microcode-level emulator. How the ISA (isa-v0.md, isa-system.md) is built from 74HCT logic, GALs, SRAM, flash and the FPGA ALU module (D21), on the two cards that make up the processor: the **CPU card** (slot 1) and the **MMU / data card** (slot 2), joined by the J3 local link (D49, D50; architecture.md, bus.md). The microcode is generated from Python tables that share the ISA's single source of truth (`microcode/p32ucode.py`, which is the authority for everything below).

## 1. Overview




- **Three buses. A** carries the first operand (register-file read port A, or the PC), **B** the second (read port B, or the immediate unit), **R** the result (the ALU, memory data, the immediate unit, MAR, the store data, CAUSE, BADADDR or a hardware control register). Each has one driver a cycle, chosen by an encoded field decoded where its sources sit, so contention is impossible by construction.
- **The register file reads two registers and writes one every cycle.** R goes to one destination a cycle: the register file's write port, PC, MAR, PC and MAR together, IR, MDR or a hardware control register.
- **The ALU is pipelined.** A word that names an ALU function latches A, B and the function at the clock edge; the result is on R from the next cycle, and the comparison of the latched operands (EQ, LT, LTU) is a condition from that edge. So the ALU takes new operands every cycle while the previous result is being used.
- **An address adder (AGU, D59)** puts A + an offset (the IR's 16- or 8-bit offset, or B shifted by 0–3 for an index) on R in the cycle the operands are read, so a load or store's address reaches MAR without the ALU's extra cycle: loads and stores take 3 cycles. MAR (on the MMU / data card) is also a counter (`MARINC`), for `LI32`'s value word.
- **Fetch overlaps the end of the instruction before.** Every instruction starts with MAR = PC; its first step reads it into IR and adds 4 to the PC; its last step sets MAR to the next PC. The generator checks the rule on every path.
- **R is the local link.** It runs on the CPU card and is extended to the MMU / data card over J3 (`LR0`–`LR31`) with the fields that say who drives it and who takes it. The CPU reaches memory only through the MMU / data card, which translates and puts the **physical** address on the system bus (D12). A load's data comes back on R, already steered and extended; a store's data goes out on R from MDR, replicated.
- **One microcycle = one CPU clock (CCLK).** Sources drive during the cycle; destinations latch at the edge that ends it. A bus cycle that a slow card stretches (`-WAIT`), or that waits for a DMA master, has no CCLK edge until it ends (`STALL`); nothing else changes.

## 2. Registers and where they live

### 2.1 Register file: 32 × 32-bit fast SRAM (CPU card)

Two read ports and one write port: two copies of the file (each four 8-bit fast SRAMs, ≤ 15 ns), written together from R, read one per port. Only 32 words are used:

| Physical | Contents |
|---|---|
| 0 | r0 — holds 0 (see below) |
| 1–14 | r1–r14 |
| 15 | r15 in **user** mode = **USP** |
| 16 | `EPC` |
| 17 | `CAUSE` |
| 18 | `BADADDR` |
| 19 | `VBR` |
| 20–27 | **T0–T7**: microcode scratch, invisible to programs |
| 28 | **write sink**: writes to r0 land here (see below) |
| 29–30 | spare |
| 31 | r15 in **supervisor** mode = **SSP** |

- **Register addresses** (5 bits each, for port A, port B and the write port) are formed on the CPU card: each port's select picks IR field A, B or C, the word's constant, the CR decode table, or SP; then the **bank rule** maps register 15 to physical 31 when S = 1 (D6's banked SP in one gate). The three ports share the word's one constant.
- **r0 = 0:** when a write's register address comes from an IR field equal to 0, the address logic sends physical **28** (the write sink) instead; the reset microcode writes 0 into physical 0 once (D4, D16).
- **Control registers stored in SRAM** (`EPC`, `CAUSE`, `BADADDR`, `VBR`, and `USP` = physical 15) cost no hardware. `MFCR`/`MTCR` find them through a small **CR decode table** (CR number → SRAM address, or a hardware register, or illegal), generated from `p32isa.py`.

### 2.2 Hardware registers

| Register | Card | Why it is not in SRAM |
|---|---|---|
| `IR` (32 bits) | CPU | its fields drive decode every cycle |
| `PC` (bits 31:2) | CPU | a **counter**: +4 at the clock edge, no ALU cycle |
| `MDR` (store data) + replication | CPU | loaded from B (or R); replicates a byte or halfword onto every lane for a store |
| `STATUS` (S, IE, PS, PIE) | CPU | S and IE gate hardware every cycle |
| `IMASK`, `IPEND` | CPU | interrupt priority logic |
| ALU operand latches, function, result | CPU (the ALU module) | at the FPGA |
| `ASID`, `TLBHI`, `TLBLO` | MMU / data | used by the TLB hardware |
| `MAR` (virtual address) | MMU / data | addresses every memory access |
| load steering | MMU / data | aligns and extends load data onto R |
| `BADADDR` latch | MMU / data | holds the faulting address until trap entry copies it (D17) |

The hardware registers on the MMU / data card are read and written over the local link: `CRSEL` names the register for `RSRC` / `RDST = CRHW`.

## 3. Cards

| Card | Contents |
|---|---|
| **CPU** (slot 1) | control-store flash + pipeline register, sequencer, IR, instruction decode, immediate unit, condition logic, STATUS, interrupt priority / IMASK, trap logic, CR decode table; the dual-read-port register file, the PC counter, MDR; the ALU module (FPGA, D21: operand latches, every ALU function incl. multiply / divide, compare outputs); the clock, reset and the front panel |
| **MMU / data** (slot 2) | MAR, split I/D TLB, ASID / TLBHI / TLBLO, alignment and upper-half checks, the BADADDR latch, load steering; the bus master interface (physical address and data drivers, `-ACK` timeout = BUSERROR, `-WAIT` → `STALL`), the arbiter and `-LOCK`. Raises MISALIGNED, PRIVILEGE (upper half), TLB miss, PROTECT, BUSERROR on `FAULT` + `FC0`–`FC3` |
| **Memory** (1–4), **I/O**, **disk**, **video**, **debug** | architecture.md §3 |

## 4. Instructions, cycle by cycle

One line per microcycle; `R[x]` the register file. Step 0 is the fetch, shared by every instruction; MAR already holds the PC.

**Fetch:** `IR ← M[MAR]` (R = memory), `PC += 4` — commits only if no fault.

**ADD rd, rs1, rs2** — 3 cycles:

```
0  fetch
1  A = R[rs1], B = R[rs2], ALU ← ADD
2  R[rd] ← ALU, MAR ← PC, end
```

**LW rd, imm(rs1)** — 3 cycles (the address adder, D59):

```
0  fetch
1  A = R[rs1]; MAR ← AGU (A + sext(imm))
2  R[rd] ← M[MAR] (steered), MAR ← PC, end     — may fault: then nothing is written
```

**SW rs, imm(rs1)** — 3 cycles:

```
0  fetch
1  A = R[rs1]; MAR ← AGU (A + sext(imm)); MDR ← B = R[rs]
2  M[MAR] ← MDR (R = MDR, replicated), MAR ← PC, end   — may fault: the store is the only effect
```

**LWX rd, rs1, rs2, scale** — 3 cycles; `SWX` 4 (base, index and data are three registers, and the register file has two read ports):

```
0  fetch
1  A = R[rs1], B = R[rs2]; MAR ← AGU (A + B << s)
2  R[rd] ← M[MAR], MAR ← PC, end
```

**LWP rd, off(rs1), inc** — 4 cycles; `SWP` 4:

```
0  fetch
1  A = R[rs1]; MAR ← AGU (A + sext(off8)); B = sext(inc8), ALU ← ADD
2  R[rd] ← M[MAR]                          — may fault: nothing written yet
3  R[rs1] ← ALU, MAR ← PC, end             (after the access: restartable)
```

**LI32 rd, value** — 3 cycles (`LUI` + `ORI`: 5):

```
0  fetch (MAR still at this instruction, the PC past it)
1  MAR += 4                                (MAR = the value's word = PC)
2  R[rd] ← M[MAR] (a fetch: execute permission), PC += 4, MAR += 4, end
```

**BEQ ra, rb, label** — 3 cycles not taken, 4 taken:

```
0  fetch
1  A = R[ra], B = R[rb], ALU ← SUB; the condition: EQ of the latched operands
2  not taken: MAR ← PC, end   |   taken: A = PC, B = offset, ALU ← ADD
3  (taken) PC, MAR ← ALU, end
```

**CALL label** — 7 cycles, nothing visible before the store (commit after the access, D6):

```
0  fetch
1  A = SP, B = 4, ALU ← SUB
2  MAR ← ALU (SP − 4); A = PC, ALU ← PASSA
3  MDR ← ALU (the return address); A = SP, B = 4, ALU ← SUB
4  M[MAR] ← MDR                        — may fault: SP and PC still unchanged
5  SP ← ALU (SP − 4); A = PC, B = offset, ALU ← ADD
6  PC, MAR ← ALU, end
```

Every instruction (from `p32ucode.py --stats`): ALU and immediate ops 3, `LUI` 2, `LI32` 3, loads and stores 3 (indexed stores and post-increment 4), `SWAP` 5, branches 3 (not taken) / 4 (taken), `JMP` 3, `JR` 4, `CALL` 7, `CALLR` 10, `RET` 6, `RTE` 3, `MFCR` / `MTCR` 3–5, the rest 2.

**Measured** (`p32uemu`, P32W Unix booted, root logged in, `halt`: 7,490,575 instructions, the same run on both datapaths):

| Datapath | Cycles | Cycles / instruction | 3 MHz | 5 MHz, a wait state on every bus cycle | 5 MHz, no wait |
|---|---|---|---|---|---|
| one internal bus | 47,030,356 | 6.28 | 0.48 MIPS | 0.65 MIPS | 0.80 MIPS |
| three buses | 26,738,361 | 3.57 | 0.84 MIPS | 1.00 MIPS | 1.40 MIPS |
| **three buses, the address adder and the D59 instructions** | **22,427,463** | **3.18** (7,049,876 instructions) | **0.94 MIPS** | **1.07 MIPS** | **1.57 MIPS** |

The three-bus datapath does the same work in 57% of the cycles (1.76× at the same clock); the address adder and the D59 instructions take another 16% off (the same boot in 84% of the cycles, 6% fewer instructions: 2.10× the single bus in all). MIPS here counts the instructions each version ran; for the same work the last row is 1.19× the one above it.

With loads and stores shorter, bus cycles are now **47%** of all cycles (one wait state on each adds 10.4 million), so memory and translation fast enough for 5 MHz without a wait state is the next lever.

## 5. Precise exceptions

The ISA promises that a faulting instruction has **no effect** (D10). The hardware gets there by **microcode discipline plus one abort mechanism**:

1. **Commit after access.** In every instruction's microcode, all writes to programmer-visible state (registers, PC, SP, memory, STATUS) come **after the instruction's last possibly-faulting memory access**, or in the access cycle itself; earlier results go to the scratch registers T0–T7 or wait in the ALU. The generator **checks this rule statically** on every path of every opcode and refuses to build images that break it.
2. **Abort.** A fault found by the MMU / data card (MISALIGNED, PRIVILEGE, TLB miss, PROTECT) or a missing `-ACK` on the system bus (BUSERROR) is detected **in the cycle of the access** and signalled on the local link (`FAULT`, the cause on `FC0`–`FC3`); the CPU card suppresses that cycle's loads and forces the next microaddress to the **trap-entry sequence**, latching the cause (D17).
3. **Trap entry** (6 cycles): `CAUSE` ← the cause latch; `EPC` ← PC (fetch fault) or PC − 4 (fault after fetch); `BADADDR` ← the MMU / data card's latch (0 for traps the CPU card raises itself); the STATUS mode bits in one step (PS←S, PIE←IE, S←1, IE←0); PC and MAR ← `VBR` (bits 31:2). Nothing is written to memory (D10).
4. **Interrupts** are checked by the sequencer at the end of each instruction: if `STATUS.IE` and `IPEND & IMASK`, the next microaddress is the interrupt-entry sequence (5 cycles) instead of the fetch (D11, D20).

# 6. Microcode

## 6.1 Address

```
microaddress = { SEQ (1) | OPCODE (8) | STEP (4) | COND (1) }  = 14 bits, 16K words
```

- `SEQ` = 0: the instruction in IR (OPCODE = IR[31:24]); `SEQ` = 1: an internal sequence (reset, trap entry, interrupt entry).
- `STEP` counts microcycles within the sequence (up to 16); `COND` is the condition the previous word chose, so every word has a taken / not-taken twin.
- Privileged opcodes in user mode, and illegal opcodes, are caught by a decode table (from `p32isa.py`) in step 1, the first after the fetch, whose effects are suppressed; trap entry follows (D18). Step 1 never starts a memory cycle.

## 6.2 Word format (56 bits, 7 flash chips)

| Field | Bits | Purpose |
|---|---|---|
| `ASRC` | 2 | A's source: none, register port A, PC |
| `BSRC` | 2 | B's source: none, register port B, immediate |
| `RSRC` | 4 | R's source: none, ALU, memory, immediate, MAR, MDR (store data), CAUSE, BADADDR, hardware CR, the address adder (D59) |
| `RDST` | 3 | R's destination: none, register write port, PC, MAR, PC and MAR, IR, MDR, hardware CR |
| `MDRB` | 1 | MDR ← B |
| `MARPC` | 1 | MAR ← PC (the next fetch's address) |
| `PCINC` | 1 | PC += 4 |
| `ALUGO`, `ALU` | 1 + 5 | latch A, B and this ALU function |
| `RSA`, `RSB`, `RSW` | 3 + 3 + 3 | each port's register address: IR.A, IR.B, IR.C, constant, CR table, SP |
| `RCONST` | 5 | the constant register address (T0–T7, EPC, …) |
| `IMM` | 3 | immediate unit: sext16, zext16, lui, branch, jump, 4, 0, inc8 (IR 7:0) |
| `MEM`, `LOCK` | 3 + 1 | cycle type: none, fetch, load, store, load word, store word, alignment check; `LOCK`: this load word and the store word after it are one bus tenure (`-LOCK`, D54) |

| Field | Bits | Purpose |
|---|---|---|
| `SEQ` | 2 | next step, end (→ fetch or interrupt), trap |
| `COND` | 3 | the condition for the next step: EQ, LT, LTU, CR illegal, CR hardware, fetch fault, MMU fault |
| `MODE` | 3 | trap-enter, RTE, TLBW, HALT, WAIT |
| `RAISE` | 4 | microcode-raised cause (SYSCALL, BREAK, ILLEGAL) |
| `AOFF` | 2 | the address adder's offset: sext16, off8 (IR 15:8), B << IR 1:0 (D59) |
| `MARINC` | 1 | MAR += 4 (D59) |

### 6.3 Store and pipeline

- **Seven 5 V parallel flash chips** (55 ns, DIP-32, D18); the outputs go through a **pipeline register** clocked on the opposite edge, so no ROM glitch reaches the datapath. The decode tables (opcode, control register) are generated from `p32isa.py` into their own flash images (D17).
- The generator (`microcode/p32ucode.py`) imports `p32isa.py`, emits the images, checks every word (R's source and destination, a memory read's data on R, a store's data from MDR, MAR not loaded twice, …), every END (MAR left at the next PC), commit after access and the lock; and **feeds the microcode-level emulator** `emulator/p32uemu.c`, which runs the real images over a model of the two cards and is compared with the instruction-level reference emulator (D7) at every instruction on the whole test suite (emulators.md).

## 7. The local link (J3, D50)

The microcode's datapath crosses between the two cards on J3 in slots 1 and 2 (bus.md, "J3"):

| Group | Direction | Signals |
|---|---|---|
| R: `LR0`–`LR31` | either (`RSRC` picks the driver) | 32 |
| `RSRC`, `RDST` (encoded) | CPU → MMU / data | 7 |
| Memory cycle `MEM0`–`MEM2`, size `SIZE0`–`SIZE1`, `SIGNED` (from IR) | CPU → MMU / data | 6 |
| `CRSEL0`–`CRSEL2` (hardware control register), `SUPER` (STATUS.S), `TLBW` | CPU → MMU / data | 5 |
| `-WSTB` (the write strobe), `LOCKREQ`, `CCLK` (the CPU's clock) | CPU → MMU / data | 3 |
| `PCINC`, `MARPC`: the MMU / data card's **shadow PC** loads from R with the PC and counts with it, so `MAR ← PC` is local to that card and needs no R cycle | CPU → MMU / data | 2 |
| `MARINC`: MAR counts by 4 (`LI32`, D59) | CPU → MMU / data | 1 |
| `FAULT` + cause `FC0`–`FC3` | MMU / data → CPU | 5 |
| `STALL` (the bus is not yet ours, or the cycle is waiting) | MMU / data → CPU | 1 |
| **Total** | | **62** |

A and B, the register addresses and the ALU function stay on the CPU card. The system bus (J1, J2) carries only physical memory cycles, interrupts, arbitration, the clock and reset.

## 8. Timing

Critical paths at 74HCT speeds (≈ 15–25 ns per gate level, 15 ns SRAM, 55 ns memory), for a 200 ns (5 MHz) microcycle:

- **Operand cycle:** pipeline register → register addresses → SRAM read → A, B → ALU latch setup ≈ 90 ns.

- **Address cycle (D59):** register read → A (and B for an index) → offset select / index shift → 32-bit adder → R → J3 → MAR setup on the MMU / data card ≈ 90 + 15 + 60 + 30 ns ≈ 195 ns with a carry-lookahead adder (74F381s with 74F182 lookahead): the tightest path at 5 MHz, to be confirmed in the card's gate-level simulation. (A ripple-carry 74HCT283 chain, ≈ 160 ns alone, fits only 3 MHz.)
- **Result cycle:** the ALU's result (from the latches, inside the FPGA) → R → the register file's write (in the cycle's last part, `-WSTB`) or MAR ≈ 120 ns.
- **Memory cycle:** MAR → TLB SRAM → tag compare → physical address on the system bus → 55 ns memory → data back on R, steered → destination setup ≈ 200 ns: one wait state at 5 MHz until translation moves earlier.

3 MHz (a 15 MHz oscillator divided by 5, D17) needs no wait state; the system bus is designed for 5 MHz (D50).

## 9. What follows

1. The bus definition's J3 (bus.md, `generators/p32bus.py`): done with this datapath; the rest of the bus is redesigned to use all three connectors (D55).
2. The card netlists, starting with the CPU card; gate-level simulation of the CPU card's netlist comes with that netlist.

# The CPU card

docs/cpu-card.md

**Status:** NETLIST DONE AND VERIFIED (2026-10-26); layout to do. Slot 1, 9U × 340 mm (D49); every part through-hole (D21); 6 layers (D77); the layout rules of D48 and D55. The design: D56 (the three-bus datapath), D58 (its choices), D59 (the address adder), D61 (this card). The netlist is `generators/card_cpu.py`; the card runs in the gate-level simulator through `generators/sim_cpu.py`.

How it works, block by block: the theory of operation.

**195 ICs, 35 of them GALs** (ATF22V10C), 9 flash chips, 8 SRAMs, the ALU module; 429 parts in all (`python3 generators/card_cpu.py --bom`).

**Faster parts on the critical paths (D72):** to meet 3 MHz at the datasheets' worst case (docs/timing.md), the register file's address muxes (74ACT157), write latches and the pipeline register (74ACT574), its read buffers and the address adder onto R (74ACT541), IR (74ABT377: no 74ACT or 74F 377 is made in DIP), the opcode mux (74F153), R's destination decode (74F138), the local link's gates (74ACT08), the pending-interrupt ORs (74ACT32) and the `-IRQ` synchroniser (74ACT574); the comparator in GALs. Same pinouts as the 74HCT parts they replace.

## 1. The cycle

A 24 MHz oscillator; the GAL **CLKGEN** counts **eight phases** a microcycle (3 MHz). Every clock on the card is a register output of CLKGEN on the same oscillator edge, so none can glitch.

**The clock jumper (D74).** A second oscillator, 32 MHz, in a socket; GAL **CKSEL** puts one of the two on OSC (D76, below). The **CLOCK** jumper (JP1, at a board edge, labelled `3 MHz` open / `4 MHz` fitted) selects it and pulls the bus line `-FAST` low, so the I/O card's timer divides by 4 instead of 3 and still ticks at 1 MHz. 3 MHz is guaranteed over the datasheets' worst case; 4 MHz meets every path at the parts' typical delays with about 20% in hand (timing.md), which most boards at room temperature will run, but which is not guaranteed. Move the jumper only with the power off: a change while running could put a short pulse on OSC.

**Stepping the clock itself (D76).** Besides STEP (one microcycle or one instruction, the oscillator running on), the machine's clock can be stopped and advanced **one oscillator edge at a time: one of the eight phases** a press, so `CLK`, `CCLK`, `-WSTB` and every bus cycle can be watched phase by phase. GAL **CKSEL** (the -7 grade) picks OSC:

| Source | When |
|---|---|
| the debug card's `XCLK` (bus, J2) | `-XCLKEN` low: the debug card's host asks (CTRL1 bit 7): its 12 MHz (a 1.5 MHz machine), or one edge per host STEP |
| the **CLOCK STEP** button | the **CLOCK** switch at STEP (front edge, labelled RUN / STEP at its ends) |
| the 32 MHz oscillator | the speed jumper fitted (`-FAST` low) |
| the 24 MHz oscillator | otherwise |

The CLOCK STEP button is debounced as RESET and STEP are (an RC, 10 ms, into a 74HCT14), so a press is one clean edge. CKSEL is a plain multiplexer: **change the source with RESET held**; a change in the middle of a phase can cut it short. With no debug card, the card's own resistors hold `XCLK` low and `-XCLKEN` high. Everything is static, so the

machine waits indefinitely between edges; the devices with their own clocks (the video card, the UARTs' baud rate, the clock chip, the USB modules, the Pico, the drives) run on in real time, which every interface allows (they wait or are polled), but a UART receiving at full speed will overrun.

| Phase | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| `CLK` (the bus clock, J2) | high | high | high | high | low | low | low | low |
| `CCLK` (the commit, J3) | rises at 0 unless held | | | | falls | | | |
| register file | write address | **write** ( `-WSTB` ) | write address | read | read | read | read | read |
| `-IRQ` synchroniser | | | | | samples ( `-CLK` rises) | | | |

- **One commit edge.** At phase 0's start `CCLK` rises: the pipeline register takes the next microword and every destination takes R (IR, PC, MDR, STATUS, IMASK, the cause latch, the flags; on the MMU / data card MAR and its shadow PC). `ACLK` rises with it in an ALUGO word: the ALU's operand and function latches.
- **The next microaddress is worked out during the cycle**, from the cycle's own signals, so the microcode's meaning ("the condition on the state after the edge") holds with one edge: a branch is tested on the **live** comparison of A and B in the word that latches them, and on the latched flags after; the opcode of step 1 comes from R while the instruction is being fetched (the opcode mux); a fault, a decode trap or an interrupt picks the internal sequence.
- **The register file is written late:** R, the write address and the enable are latched at the commit and written in phases 0–2 of the next cycle (the strobe in phase 1, the address steady either side), before that cycle's reads (phases 3–7). A register written by one word and read by the next needs no bypass.
- **Held cycles:** `STALL` (the MMU / data card: arbitration, `-WAIT`), WAIT with nothing pending, HALT, the front panel's STEP mode: no `CCLK` edge; the cycle repeats with the same word, and its late write again (the same data to the same register).
- **Timing at 3 MHz** (333 ns; 74HCT ≈ 15–25 ns a level, flash 55 ns, SRAM 15 ns, 74F283 ≈ 8 ns a stage), *estimates to check against the datasheets*: reads begin at phase 3 (125 ns); A and B ≈ 40 ns later; the address adder to MAR ≈ 110 ns more (≈ 275 ns); the comparison through the condition, the flash and the pipeline register's setup ≈ 150 ns more (≈ 315 ns: the tightest path). The fetch's opcode path (memory data on R, the opcode mux, the flash) fits if the MMU / data card has the data back by ≈ 220 ns. **5 MHz does not fit this card as designed**: it needs the reads to start earlier (a separate write port), faster parts on the condition path and translation moved earlier (backlog).

## 2. Blocks

| Block | Parts | ICs |
|---|---|---|
| **Clock and control** | 24 MHz oscillator, a 32 MHz one in a socket, GAL CKSEL picking OSC from them, the CLOCK STEP button or the debug card's XCLK (D74, D76); GAL CLKGEN; GAL PANEL (reset from PWROK and the RESET button, RUN / STEP, MICRO / INSTR stepping, HALT, the LEDs); 74HCT14 (debounce, the ALU's GO toggle) | 6 |
| **Control store** | 7 × SST39SF010A (the 56-bit microword, D59), 7 × 74ACT574 pipeline register | 14 |
| **Sequencer** | 74HCT574 microaddress register (sequence, step, internal sequence id); GAL NEXT (the next step and sequence, the opcode source); 4 × 74F153 opcode mux (IR, R, the sequence id); GAL COND (the condition: live or latched compare, CR table, fault flags); GAL EVENTS (abort, trap / interrupt entry, ALUGO's commit, WAIT, HALT, the cause source); 74HCT138 step decode | 9 |
| **Instruction unit** | 4 × 74HCT377 IR; 2 × SST39SF010A decode tables (opcode: legal, privileged, zero class, size, signed; CR: illegal, hardware, address); 4 × 74HCT4002 must-be-zero groups; GAL DECODE (the decode trap, D18) | 11 |
| **Register addresses** | per port (A, B, write): 2 × 74HCT153 (the IR field A / B / C or SP) and a GAL (the bank rule, the constant, the CR table's address; the write port's r0 sink) | 9 |
| **Register file** (two copies, D58) | 8 × CY7C199; 74HCT574 write address and enable; 4 × 74HCT157 read / write address; 8 × 74HCT574 write data; 8 × 74HCT541 onto A and B; a 74HCT32 gate ( `-WE` ) | 29 |

| Block | Parts | ICs |
|---|---|---|
| **PC** | 8 × 74HCT161 (bits 31:2: count, load from R); 4 × 74HCT541 onto A | 12 |
| **Immediate unit** | 6 GALs IMM0–5 onto B (sext16, zext16, LUI, branch, jump, 4, 0, inc8) | 6 |
| **Address adder** (D59) | 4 × 74HCT257 and 2 × 74HCT541 (the IR's 16- or 8-bit offset); 4 GALs IDX0–3 (B << IR[1:0]); 8 × 74F283; 4 × 74HCT541 onto R | 22 |
| **Comparator** (D58, D72) | GALs CMP0–CMP7 (each nibble's GT and LT; CMP7 also signed), CMPH (bits 16–31), CMPF (bits 0–15, then EQ, LTU, LT): three GAL levels, about 40 ns where the 74HCT85 tree took 150 | 10 |
| **MDR** (store data) | 8 × 74HCT157 (from B or R), 4 × 74HCT377; onto R replicated by size: 74HCT541, 4 × 74HCT257, 4 × 74HCT253 | 21 |
| **ALU** (D21) | Tang Nano 20K; 8 × 74LVC574 operands, 74LVC574 function + GO, 4 × 74LVC574 result onto R; 74LVC244 (its reset); LD1117V33 | 16 |
| **Status and interrupts** | GAL STATUS (S, IE, PS, PIE; TLBW); GAL IMASK; 74ACT574 `-IRQ` synchroniser; 3 × 74ACT32 (pending); GAL ANYP (any pending, D72); 74HCT148 priority (the level's number); 2 × 74HCT153 cause mux; 74HCT377 cause latch; onto R: 3 × 74HCT541 (CAUSE, STATUS), 2 × 74HCT540 (IPEND, IMASK) | 15 |
| **R's other sources and destinations** | 2 × 74HCT138 (RSRC); GAL RDEC (LUI, zero fill, CPU CRs); 4 × 74HCT541 zero fill; 2 × 74HCT541 LUI; GAL ABDEC (A / B enables, size, sign bits); 74HCT138 (RDST, gated by the abort); 2 × 74HCT08, 74HCT00 (the local link's gated lines, PC and MDR loads) | 14 |
| **Front panel** (front edge, labelled, D48) | RESET and STEP buttons, RUN / STEP and INSTR / MICRO switches, RUN and HALT LEDs; the CLOCK switch (RUN / STEP) and CLOCK STEP button (D76); the POWER LED | — |
| **Pull-downs** | 12 × 10k networks on R, A, B | — |

**The abort:** reset, a decode trap (step 1) or a fault from the MMU / data card suppresses the cycle — R's destination decoder is disabled, the local link's RDST, PCINC, MARPC and MARINC are gated off, ALUGO's latches don't clock, MODE has no effect — and the sequence goes to trap entry with the cause latched.

**The local link (J3)** carries R itself (`LR0–31`, the card's R bus), RSRC, RDST (gated), MEM, SIZE / SIGNED (from the opcode table), CRSEL, SUPER, TLBW, `-WSTB`, LOCKREQ, CCLK, PCINC / MARPC / MARINC (gated), and back FAULT, FC and STALL.

**"A CPU card is in":** `-CPUIN` (J3 B28, D102). The pin is tied to the card's ground plane, with no part. On the backplane the line runs to the debug card in slot 0, which pulls it up: low there means this card is in its slot, and the debug card then refuses to make `CLK`, `-RES` or the local link's lines itself (debug-card-theory §9). Nothing on this card reads it. J3 B30 is the MMU / data card's line, `-MMUIN`, not connected here.

## 3. The ALU module (D21)

The operand latches take A and B at `ACLK`; the function latch takes the 5-bit function and toggles **GO**. The FPGA sees GO change, reads the four operand halves over its 16-bit `FD` bus (one latch pair's outputs at a time, `SELAL_N` … `SELBH_N`), computes, and writes the result halves into the result latches (`CKLO`, `CKHI`), which drive R when RSRC = ALU — well inside the next microcycle at 3 MHz (a division, 18 fclk = 167 ns, D21). The module's 34 GPIOs: 29 used. **Which module pin is which GPIO is to verify against Sipeed's pin diagram before layout** (the netlist's pads are provisional); the FPGA's side of this protocol is still to be written (`fpga/alu`: the core exists).

## 4. Verification

- **Every GAL** is generated from a behavioural spec (`minimize`, or a hand equation for the wide ones) and checked exhaustively against it, with the 22V10's product-term limits (`p32gal.py`).
- **The netlist** passes the electrical checks (`p32net.check`: every pin connected or declared unused, no floating input, no two drivers, power).

- **The card as wired runs programs** (`generators/sim_cpu.py`): every chip simulated, the real GAL equations, the control store and decode tables in its flash chips; the MMU / data card, memory and the devices are `p32uemu`'s own model (`emulator/p32busmodel.c`) on J3. All 17 test programs of the cross-check (`hello`, `echo`, `isa_test`, `d59_test`, `fib`, `primes`, `trap_test`, `kernel_hello`, `kernel_fault`, `irq_test`, `wait_nowake`, `mt_demo`, `vm_test`, `vm_demo`, the exception tests) run with **the state at every instruction identical to `p32emu`'s and the cycle count identical to `p32uemu`'s**; `make test` runs five of them. About 300 microcycles a second: too slow to boot Unix.
- **Not simulated:** timing (the simulator is zero-delay: §1's estimates are to be checked against the datasheets), the FPGA's FD-bus protocol, STALL from a real MMU / data card.

## 5. Next

The schematic and the board (the `p32kicad_*` generators, placement by the layout rules, 4 or 6 layers), the module's pin map, the FPGA's bus protocol; then the MMU / data card, designed the same way, against the same model.

# The CPU card: theory of operation

docs/cpu-card-theory.md

**Status:** DESCRIBES THE NETLIST AS BUILT AND VERIFIED (2026-10-26), `generators/card_cpu.py` (every chip and GAL equation comes from there), running the microcode of `microcode/p32ucode.py`. The card's summary, its parts list by block and its verification are in the CPU card. The datapath it implements is the microarchitecture (D56). This chapter explains how the circuit works, block by block, and then follows three instructions through it cycle by cycle.

## 1. What the card does

The CPU card is the whole processor **except memory access**. It holds:

- the microcode, and the sequencer that steps through it;
- the instruction register and decode;
- the 32 registers and the PC;
- the ALU (a Tang Nano 20K FPGA module, D21);
- the address adder, the comparator, the immediate unit;
- STATUS, IMASK and the interrupt logic;
- the clock and the front panel.

Everything to do with memory (MAR, translation, faults, the bus cycle, the byte steering of loads) is on the **MMU / data card** in slot 2. The two cards talk over **J3, the local link**, which exists only in slots 1 and 2 (and slot 0, where the debug card listens; one more J3 pin, B28, is tied to this card's ground: `-CPUIN`, by which the debug card knows a CPU card is in and will not drive its lines, D102). The CPU card tells the MMU / data card what each cycle needs (`MEM`, `SIZE`, `RSRC`, `RDST`, `PCINC` …). The MMU / data card answers with load data on R, `FAULT` and its cause, and `STALL`.

## 2. The three buses

The card is built around three 32-bit buses (D56):

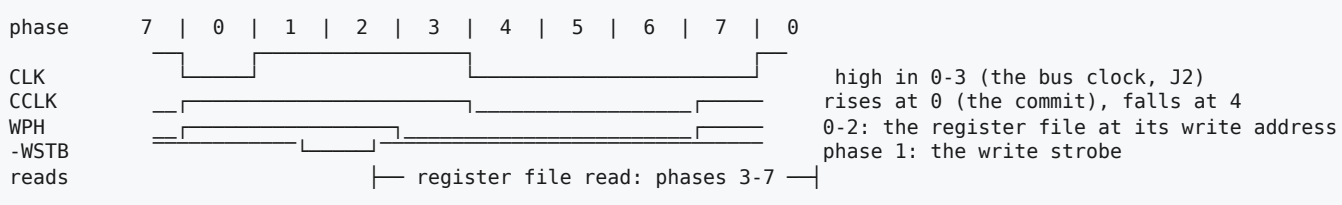
| Bus | Nets | Driven by | Read by |
|---|---|---|---|
| **A** | `AB0–31` | register file copy A; the PC | the ALU's operand A latches; the address adder; the comparator |
| **B** | `BB0–31` | register file copy B; the immediate unit | the ALU's operand B latches; the address adder's index; the comparator; the MDR |
| **R** (the result) | `LR0–31` | the ALU result latches; the address adder; the MMU / data card (load data, MAR, BADADDR, its CRs); the immediate unit's LUI and zero fill; the store data lanes; CAUSE, STATUS, IPEND, IMASK | the register file (write); the PC; the IR; the MDR; STATUS, IMASK; the MMU / data card (MAR, the shadow PC, its CRs, store data) |

R is the same net as J3's `LR0–31`: the local link carries the result bus itself. Each bus has 10k pull-downs (twelve 8-resistor networks), so in a cycle that drives nothing a bus reads 0 rather than floating.

**Who drives a bus** is decided by the microword: `ASRC` (none, register, PC), `BSRC` (none, register, immediate) and `RSRC` (16 codes). Decoders turn each field into one active-low enable. Exactly one source drives a bus in a cycle, and the simulator checks every cycle for two drivers fighting.

## 3. The clock: eight phases, one commit edge

A **24 MHz oscillator** clocks **GAL CLKGEN**, a 3-bit phase counter (`PH0–2`) whose other outputs are all registered on the same oscillator edge. None of them can glitch, because none is decoded combinationally. With the **CLOCK** jumper fitted (D74) GAL CKSEL passes a 32 MHz oscillator to CLKGEN instead: the same eight phases, a 4 MHz microcycle, and the bus line `-FAST` low so the I/O card's timer still ticks at 1 MHz. CKSEL can also take the CLOCK STEP button or the debug card's `XCLK`, one edge at a time (D76): each edge is one phase, and nothing moves between.




One microcycle is eight phases of 41.7 ns, which is **333 ns at 3 MHz**.

- `CCLK`**, the commit**, rises at the start of phase 0, and everything that holds state changes on that one edge: the pipeline register (the next microword), the microaddress register, IR, PC, MDR, STATUS, IMASK, the cause latch, the comparison flags, the register file's write latches, and, on the MMU / data card, MAR and its shadow PC. `ACLK` rises with it, but only in a word with `ALUGO`: the ALU's operand and function latches.
- **A held cycle** has no `CCLK` edge. CLKGEN suppresses it when `STALL` (from the MMU / data card: a DMA master has the bus, or a slow card holds `-WAIT`) or `HOLD` (from GAL PANEL: WAIT with nothing pending, HALT, the front panel's STEP mode) is high at the end of phase 7. The same microword then runs again. Its effects are idempotent: the late write writes the same value to the same register again.
- `CLK`, the backplane's bus clock, is high in phases 0–3. Every card on the bus works from it (the bus).

## 4. The control store and the microword

The microword is **56 bits** in **seven SST39SF010A flash chips** (128 K × 8, 55 ns), one byte each. Their outputs go into **seven 74HCT574s, the pipeline register**, clocked by `CCLK`. So the word that controls a cycle is the one the flash produced during the previous cycle, latched at the commit. Its fields (`p32ucode.py`, low bit first):

| Field | Bits | What it controls |
|---|---|---|
| `ASRC` | 2 | A's source: none, the register file, the PC |
| `BSRC` | 2 | B's source: none, the register file, the immediate unit |
| `RSRC` | 4 | R's source: none, ALU, MEM (the MMU / data card), IMM, MAR, MDRS (store data), CAUSE, BADADDR, CRHW (a hardware CR), AGU (the address adder) |
| `RDST` | 3 | R's destination at the commit: none, REG, PC, MAR, PC+MAR, IR, MDR, CRHW |
| `MDRB` | 1 | the MDR loads from B instead of R |
| `MARPC`, `PCINC`, `MARINC` | 1 each | MAR ← the shadow PC; PC += 4; MAR += 4 (D59) |
| `ALUGO` | 1 | latch A, B and the function into the ALU at this commit |
| `ALU` | 5 | the ALU function |
| `RSA`, `RSB`, `RSW` | 3 each | each register port's address source: the IR's field A, B or C, SP, a constant, the CR table |
| `RCONST` | 5 | the constant register address (EPC, CAUSE, scratch …) |
| `IMM` | 3 | the immediate unit's mode |

| Field | Bits | What it controls |
|---|---|---|
| `MEM` | 3 | the memory cycle: none, FETCH, LOAD, STORE, LOADW, STOREW, CHECKW, IACK |
| `LOCK` | 1 | `LOCKREQ`: this access and the next are atomic (SWAP, D51, D54) |
| `SEQ` | 2 | NEXT step, END of the instruction, TRAP |
| `COND` | 3 | the condition that selects between a step's two words |
| `MODE` | 3 | none, TRAPENTER, RTE, TLBW, HALT, WAIT |
| `RAISE` | 4 | the cause a `SEQ = TRAP` word raises |
| `AOFF` | 2 | the address adder's offset: the 16-bit offset, the 8-bit one, the index |

**The microaddress** is 14 bits on the flash chips' address lines:

| Address bit | 0 | 1–4 | 5–12 | 13 |
|---|---|---|---|---|
| | **the condition** | **the step** (0–15) | **the opcode**, or the internal sequence's number | **USEQ**: an instruction (0) or an internal sequence (1) |

So each instruction owns 16 steps of two words: a word for "condition false" and a word for "condition true". A step that does not depend on a condition has the same word in both. Step 0 of every opcode is the same **FETCH** word. The internal sequences (USEQ = 1) are RESET, TRAP and INTR.

## 5. The sequencer: choosing the next microaddress

The next microaddress is **worked out during the cycle**, from the cycle's own signals, and latched at the same `CCLK` edge as everything else (**74HCT574 microaddress register**: USEQ, the step, the internal sequence's number). The microcode's convention is "the condition is tested on the state **after** this cycle's edge". The hardware gets that with one edge by computing the condition from the signals that will be latched.

**GAL NEXT** computes the next sequence, step and opcode source:

| This cycle | Next microaddress |
|---|---|
| RESET | USEQ = 1, step 0, sequence RESET |
| a trap being entered (`GOTRAP`) | USEQ = 1, step 0, sequence TRAP |
| an interrupt being taken (`GOINTR`) | USEQ = 1, step 0, sequence INTR |
| `SEQ = NEXT` | the same sequence, step + 1 |
| `SEQ = END` | USEQ = 0, step 0: the FETCH word (the same for every opcode) |

**The opcode mux** (four 74HCT153s) supplies address bits 5–12 from one of three places (`OPS`):

- **IR[31:24]**, normally;
- **R[31:24]** in the step after FETCH. The instruction is still on R, arriving from memory, and has not reached IR yet, so its opcode comes straight off the bus. This saves a cycle on every instruction.
- **the internal sequence's number**, for RESET, TRAP and INTR.

**GAL COND** computes address bit 0, the condition (`NCOND`), only in a `SEQ = NEXT` word:

| COND | Condition |
|---|---|
| `EQ`, `LT`, `LTU` | the comparison of A and B. **In the word that latches them** (`ALUGO`) it is the comparator's **live** output, the comparison of what is on the buses now; in a later word, the flags COND **latched** at that ALUGO's commit (`EQF`, `LTF`, `LTUF`) |

| COND | Condition |
|---|---|
| CRILL, CRHW | from the control-register decode table: the CR number is illegal; it is a hardware CR, not one kept in the register file |
| FFAULT, MMUF | for the trap sequence: the fault came from a fetch; it came from the MMU / data card (D17) |
| ZERO | 0 |

COND also keeps the two fault flags, latched when a trap is entered.

**GAL EVENTS** decides what kind of cycle this is:

- ABORT_N: the cycle is cancelled when there is a reset, a decode trap (DECTRAP) or a fault from the MMU / data card (FAULT). R's destination decoder is disabled, and the local link's RDST, PCINC, MARPC and MARINC are gated off. ALUGO, MODE and the MDR load have no effect. Nothing the cycle would have changed changes, so an instruction that faults can be restarted (commit after access, D14).
- GOTRAP: a decode trap, a fault, or a SEQ = TRAP word.
- GOINTR: an END word, not aborted, with STATUS.IE set and an enabled interrupt pending, and not the end of trap entry itself (D20).
- SEQNEXT, ALUGOK (an ALUGO that commits), WAITHOLD (MODE = WAIT with nothing pending: GAL PANEL holds the clock), HALTMODE.
- **The cause latch's source** (CSEL, below) and its load (CLOAD_N).

A **74HCT138** decodes the step into FIRST_N (step 0, the fetch) and FIRST1_N (step 1, when the decode trap is checked).

## 6. The instruction unit

- **IR**: four 74HCT377s loaded from R at the commit when RDST = IR (the FETCH word).
- **The opcode decode table**: an SST39SF010A addressed by IR[31:24]. For each opcode it gives LEGAL, PRIV (privileged), a 3-bit **zero class** (which bits of the instruction must be 0), SIZE (byte, halfword, word) and SIGNED. SIZE and SIGNED go to the MMU / data card on J3.
- **The control-register decode table**: an SST39SF010A addressed by IR[19:16] (an MFCR / MTCR's CR number). It gives CRILL (no such CR), CRHW (a hardware CR: STATUS, IPEND, IMASK, ASID, TLBHI, TLBLO), CRSEL (which one; it goes to the MMU / data card for its three) and, for CRs kept in the register file (EPC, CAUSE, BADADDR, VBR, USP), the register address.
- **The must-be-zero check** (D18): four 74HCT4002 (dual 4-input NOR) detect seven groups of IR bits that are all zero. **GAL DECODE** compares them with the opcode's zero class. In step 1 it raises DECTRAP for an illegal opcode or a must-be-zero field that is not zero (cause ILLEGAL), or a privileged opcode in user mode (cause PRIVILEGE). The step-1 word is then aborted and the trap entered.

## 7. The register file

**32 physical registers** of 32 bits:

| Address | Register |
|---|---|
| 0 | r0: written 0 by the reset microcode, then never written again |
| 1–14 | r1–r14 |

| Address | Register |
|---|---|
| 15 | r15 in user mode: the user stack pointer (USP) |
| 16–19 | EPC, CAUSE, BADADDR, VBR (control registers kept here) |
| 20–27 | T0–T7: the microcode's scratch registers |
| 28 | the sink: a write to r0 lands here |
| 31 | r15 in supervisor mode: the supervisor stack pointer (SSP, D6) |

**Two copies** (D58), each four **CY7C199** (32 K × 8, 15 ns; only five address bits used): copy A drives bus A, copy B drives bus B. Both are written together with the same data. Together they give **two read ports and one write port** from single-ported SRAM.

**The port addresses.** For each port (A, B, write) two 74HCT153s pick an IR field: field A (IR[23:20]), B (IR[19:16]), C (IR[15:12]) or 15 (SP), under `RSA` / `RSB` / `RSW`. Then a GAL (**REGA, REGB, REGW**) makes the 5-bit physical address:

- **the bank rule**: field value 15 in supervisor mode becomes 31 (SSP);
- or the microword's **constant** (`RCONST`: EPC, a scratch register …);
- or the **CR table's** address (`MFCR` / `MTCR` of a CR kept in the file);
- and for the write port only, register 0 becomes the **sink, 28**. r0 stays 0 without any gating.

**The late write.** A result is on R at the end of its cycle, but the register file is written **in the next cycle's first three phases**:

1. At the commit, a 74HCT574 latches the write address and its enable (`RDST = REG`), and eight 74HCT574s (four per copy) latch R itself.
2. In phases 0–2 (`WPH` high), 74HCT157s switch both copies' address lines from the read address to the latched write address. The SRAMs' outputs are off (`-OE = WPH`), and the latched data drives their data pins.
3. `-WSTB` (phase 1) ORed with the latched enable makes `-WE`: the write happens in the middle of the window, with the address steady on either side of it.
4. In phases 3–7 the address lines carry the read addresses. The SRAMs drive their 74HCT541s onto A and B, which GAL ABDEC enables outside `WPH`.

So a register written by one word and read by the next one **needs no bypass**: the write happens before that cycle's reads begin. The cost is that reads start at phase 3 (125 ns into the cycle), and that is what limits the clock (§17).

## 8. The PC

The PC is bits 31:2 (instructions are word-aligned): **eight 74HCT161 counters**, a carry chain for `PCINC` (PC += 4 at the commit), and a load from R for `RDST = PC` or `PC+MAR`. Four 74HCT541s put it on A (bits 1:0 as 0) for branch targets, `CALL`'s return address and EPC.

The MMU / data card keeps a **shadow PC** that loads and counts with this one, so the next fetch's address (`MARPC`: MAR ← PC) costs no cycle on R.

## 9. The immediate unit

Six GALs (**IMM0–5**) drive B from the IR, under `IMM`, when `BSRC = IMM`:

| Mode | Value on B |
|---|---|
| `SEXT16` | IR[15:0] sign-extended |
| `ZEXT16` | IR[15:0] zero-extended |
| `LUI` | IR[15:0] << 16 |
| `BR` | IR[15:0] sign-extended, × 4 (a branch offset) |
| `J` | IR[23:0] sign-extended, × 4 (a jump offset) |
| `FOUR`, `ZERO` | 4, 0 |
| `INC8` | IR[7:0] sign-extended (post-increment, D59) |

Each GAL covers a group of B's bits and has as inputs only the IR bits that can reach them, generated from `imm_value` (the same function `p32uemu` uses).

Two more R sources use the IR: **LUI** puts IR[15:0] on R[31:16] (two 74HCT541s, with zero fill below), and the **zero fill** (four 74HCT541s with grounded inputs) fills the bytes of R that a narrower source (CAUSE, STATUS, IPEND, IMASK, an 8-bit immediate) does not drive. **GAL RDEC** decides which bytes.

## 10. The address adder (D59)

Loads and stores compute their address in the cycle the base register is read, without the ALU. That makes a load three cycles, not four.

- **The offset**, `AOFF`:
  - `SEXT16`: the 16-bit offset, IR[15:0], through four 74HCT257s, with IR[15] spread to bits 31:16 by two 74HCT541s;
  - `OFF8`: the 8-bit offset IR[15:8], through the same 74HCT257s (post-increment, D59);
  - `INDEX`: B shifted left by IR[1:0] (scale 1, 2, 4, 8), from four GALs **IDX0–3**.
- **The adder**: eight **74F283** (4-bit, fast carry) add A and the offset.
- **Onto R**: four 74HCT541s when `RSRC = AGU`. Usually `RDST = MAR`, so the address goes to the MMU / data card at the commit.

## 11. The comparator (D58)

Fourteen **74HCT85** magnitude comparators in a tree compare A with B directly, without waiting for the ALU:

- eight compare bits 3:0 … 31:28 (A > B, A < B per nibble);
- one more compares bits 31:28 with both sign bits inverted (GAL ABDEC inverts them), for the signed result;
- two compare the eight nibble results in two halves (bits 15:0, 31:16), and one more does the signed upper half;
- the last two give `CMPEQ`, `CMPLTU` (unsigned) and `CMPLT` (signed).

Branches use them through GAL COND (§5). So does `SLT`, which latches them as flags.

## 12. The MDR and the store data

- **The MDR** (four 74HCT377s) loads at the commit from R or, with `MDRB`, from B, through eight 74HCT157s. A store's data is usually register `rd` on B, loaded in the same cycle as the address.

- **Store data onto R** (`RSRC = MDRS`): the bytes are **replicated by size** so that each byte lane of the bus carries the right byte whatever the address. Lane 0 is always MDR[7:0]. Lane 1 is MDR[7:0] for a byte, else MDR[15:8] (74HCT257). Lane 2 is MDR[7:0] for a byte or halfword, else MDR[23:16]. Lane 3 is MDR[7:0], MDR[15:8] or MDR[31:24] by size (74HCT253). The MMU / data card's byte enables then pick the lanes that are written.

## 13. The ALU module (D21)

The ALU is an FPGA on a **Tang Nano 20K** module, at 3.3 V behind 74LVC latches (5 V tolerant):

1. At `ACLK` (an ALUGO word's commit), **eight 74LVC574s** take A and B, and one more takes the 5-bit function and **toggles GO** (a 74HCT14 inverter feeds GO's complement back into its own D input).
2. The FPGA sees GO change. It reads the four operand halves over its 16-bit **FD bus**, enabling one latch pair at a time (`SELAL_N`, `SELAH_N`, `SELBL_N`, `SELBH_N`), computes, and writes the result halves into **four 74LVC574 result latches** (`CKLO`, `CKHI`).
3. The result latches drive R when `RSRC = ALU`, in the next cycle or later. A division takes 18 FPGA clocks, about 167 ns, so even the slowest result is ready well inside the next 333 ns cycle.

The module's reset comes from `-RES` through a 74LVC244, and an LD1117 makes 3.3 V for the latches. The FPGA's side of the FD-bus protocol is still to be written. **The module's pins** (D70) are as Sipeed's datasheet v1.3 shows them: 5 V and GND at the USB end of the right row, 3.3 V and GND at the far end of the left row and GND, 3.3 V in the right row; the rows 20.32 mm apart. Which FPGA pin carries which signal is in `fpga/alu/p32alu_nano.cst` (generated from the card's pin map): the FD bus on pins that otherwise reach only the unused LCD connector, the microSD slot and the HDMI EDID lines; the latch selects and result clocks on the six LED pins (the LEDs show the ALU at work); the function on the audio amplifier's inputs and the WS2812's; GO and the reset on two of the BL616's high-speed SPI lines (HSPI_DIN2, HSPI_DIN3), the only pins left; that its debugger firmware leaves them alone is to confirm on the bench. **Do not plug a USB cable into the module while the card is powered** unless the module's 5 V pin is diode-isolated from its USB-C (to check on Sipeed's schematic): the backplane's 5 V and the USB's would meet.

## 14. STATUS, IMASK and interrupts

**GAL STATUS** holds S (supervisor), IE (interrupts enabled), PS and PIE (their previous values). At the commit:

- `MTCR STATUS` writes it from R[3:0];
- **trap entry** (`MODE = TRAPENTER`) saves S and IE into PS and PIE, then sets S = 1 and IE = 0;
- `RTE` restores S and IE from PS and PIE.

Reset sets S = 1 and IE = 0. STATUS also makes `TLBW` for the MMU / data card (`MODE = TLBW`) and drives `SUPER` on J3.

**GAL IMASK** holds the interrupt mask **inverted** (`IMN`: 1 = disabled), so reset (all registers 0 in the GAL, the outputs inverted) masks everything.

**The interrupt path:**

1. A **74HCT574 synchroniser** samples `-IRQ0–7` at `-CLK`'s rise (phase 4), so an asynchronous request can never be seen half-changed.
2. Three 74HCT32s OR each synchronised line with its `IMN` bit: a low output means "pending and enabled".

3. A **74HCT148** priority encoder gives `ANYP_N` (something is pending and enabled) and the **lowest-numbered** such level (IRQ0, the timer, is the highest priority).
4. At an `END` word with IE set, GAL EVENTS raises `GOINTR`, and the next microaddress is the INTR sequence.

**The cause latch** (a 74HCT377 with a 74HCT153 mux) loads at trap or interrupt entry. It takes one of four sources: the microcode's `RAISE` (SYSCALL, BREAK, …), the MMU / data card's fault cause (`FC`), the decode trap's cause (ILLEGAL / PRIVILEGE), or the interrupt level, with bit 4 marking an interrupt (CAUSE bit 31). It drives R for the trap sequence's first word, which stores it in the CAUSE register (address 17). Further 74HCT541 / 540 buffers put CAUSE, STATUS, IPEND and IMASK on R for `MFCR`.

**The interrupt sequence** (INTR, six words) puts CAUSE in MAR and saves the PC in EPC. It runs the **acknowledge cycle** (`MEM = IACK`, D57): the MMU / data card puts `-IACK` on the bus, and the card driving that level answers with its vector, which comes back on R in CAUSE bits 15:8 and goes into the CAUSE register. Then BADADDR = 0, PC = MAR = VBR, and trap entry switches STATUS. **The trap sequence** is the same without the acknowledge cycle, with EPC chosen by whether the fault was in the fetch, and BADADDR from the MMU / data card.

## 15. R's sources and destinations

- **Sources**: two 74HCT138s decode `RSRC` into enables for the ALU, IMM, MDRS, CAUSE, CRHW and AGU. The MMU / data card decodes the same field on J3 for MEM, MAR, BADADDR and its own CRs. GAL RDEC adds LUI, the zero fill and which of the CPU's hardware CRs to drive.
- **Destinations**: a 74HCT138 decodes `RDST` with `ABORT_N` **as its enable**, so an aborted cycle writes nothing: REG (the late-write latch), PC, PC+MAR, IR, MDR, CRHW (STATUS, IMASK; the MMU / data card's CRs). 74HCT08s gate the local link's `RDST`, `PCINC`, `MARPC` and `MARINC` the same way, and the PC's and MDR's loads.

## 16. Reset, the front panel and stepping

**GAL PANEL** (on the 24 MHz clock) turns the front panel and the events into the clock's `HOLD`:

| Control | Effect |
|---|---|
| **RESET** button, or `PWROK` low | `RESET` → `-RES` on the backplane; the sequencer goes to the RESET sequence (boot §1–2) |
| **RUN / STEP** switch | RUN: free running. STEP: `HOLD` except when STEP allows a commit |
| **STEP** button | MICRO mode: one commit (one microcycle). INSTR mode: commits up to the next instruction boundary (`NBOUND`, from GAL NEXT) |
| **INSTR / MICRO** switch | what one STEP press does |
| HALT instruction | `HALTED` latched at its commit; `HOLD` until reset |
| WAIT with nothing pending | `WAITHOLD` from EVENTS: the WAIT word repeats until an enabled interrupt arrives |
| **RUN** LED | lit while the clock commits (not `HOLD`) |
| **HALT** LED | lit after a `HALT` |

The buttons are debounced by an RC and a 74HCT14 Schmitt trigger.

## 17. Three instructions, cycle by cycle

Each row is one 333 ns microcycle. "At the commit" is the `CCLK` edge that ends it.

`ADD r1, r2, r3` (3 cycles, 1 µs):

| Cycle | Word | What happens | At the commit |
|---|---|---|---|
| 1 | FETCH | `MEM = FETCH`: the MMU / data card reads the word at MAR, and it arrives on R. The opcode mux takes R[31:24] for the next microaddress. | IR ← R; PC += 4 (and the shadow PC) |
| 2 | step 1 | A = r2 (field B), B = r3 (field C); `ALUGO`, function ADD. DECODE checks the opcode (legal, no stray bits). | the ALU latches A, B, ADD; the FPGA starts |
| 3 | step 2, END | R = the ALU's result; `RDST = REG`, write port = field A (r1). `MARPC`: MAR ← the PC. No interrupt pending. | the write latched; MAR = PC; next microaddress: FETCH |
| (4) | the next FETCH | phases 0–2: r1 written | |

`LW r1, 8(r2)` (3 cycles):

| Cycle | Word | What happens | At the commit |
|---|---|---|---|
| 1 | FETCH | as above | IR, PC |
| 2 | step 1 | A = r2; the address adder adds the 16-bit offset 8; `RSRC = AGU`, `RDST = MAR` | MAR ← r2 + 8 |
| 3 | step 2, END | `MEM = LOAD`, `SIZE` = word: the MMU / data card translates MAR, runs the bus cycle and puts the word on R; `RDST = REG` (r1); `MARPC` | r1's write latched; MAR = PC |

If the load **faults** (a TLB miss, say), the MMU / data card raises `FAULT` in cycle 3. `ABORT_N` cancels the cycle's register write and its `MARPC`. The cause latch takes the fault's cause, and the next microaddress is the TRAP sequence. EPC will be this instruction, so it runs again after the handler.

`BEQ r1, r2, label`**, taken** (4 cycles; 3 when not taken):

| Cycle | Word | What happens | At the commit |
|---|---|---|---|
| 1 | FETCH | as above | IR, PC |
| 2 | step 1 | A = r1, B = r2; `ALUGO` SUB; `COND = EQ`. The comparator's **live** EQ goes to the flash's address bit 0, so step 2's two words are chosen now. | flags latched; the next word is step 2's "true" word |
| 3 | step 2 (true) | A = the PC, B = the branch offset (IMM `BR`); `ALUGO` ADD | the ALU starts PC + offset |
| 4 | step 3, END | R = the ALU; `RDST = PC+MAR` | PC = MAR = the target |

Not taken, step 2's "false" word is `MARPC`, `END`: three cycles.

## 18. Timing

At 3 MHz a cycle is 333 ns. The estimates below are from typical part delays and are **to be checked against the datasheets** (the simulator has zero delay):

| Path | Approximate time |
|---|---|
| the register file's read address valid | phase 3, 125 ns |
| A and B valid (SRAM 15 ns + 541) | ≈ 165 ns |
| the address adder (74F283 chain) onto R and to MAR's set-up | ≈ 275 ns |
| **the comparison → COND → the flash (55 ns) → the pipeline register's set-up** | **≈ 315 ns: the tightest path** |
| a fetch: the MMU / data card's data on R by ≈ 220 ns → the opcode mux → the flash | fits |

5 MHz (200 ns) does not fit this design as it stands. It would need the reads to start earlier (a separate write port), faster parts on the condition path, and translation moved earlier (backlog).

# The MMU / data card

docs/mmu-card.md

**Status:** NETLIST DONE AND VERIFIED with the CPU card (2026-10-26); layout to do. Slot 2, 9U × 340 mm (D49), through-hole (D21). The CPU card's partner on the local link (J3) and its way onto the system bus (J1, J2): D12, D17, D50, D52, D54, D57, D59, D62. The netlist is `generators/card_mmu.py`.

How it works, block by block: the theory of operation.

**"An MMU / data card is in":** `-MMUIN` (J3 B30, D102): the pin is tied to the card's ground plane, with no part. The debug card in slot 0 pulls the line up and reads it: while it is low, the debug card alone leaves D, the bus cycle's lines and the grants to this card, whose address drivers, strobes and arbiter drive them (debug-card-theory §9). Nothing on this card reads it; J3 B28, the CPU card's `-CPUIN`, is not connected here.

**130 ICs, 8 of them GALs**; 269 parts (`python3 generators/card_mmu.py --bom`).

It has no clock of its own: `CCLK` (the commit), `CLK` (the bus) and `-WSTB` (the CPU card's write strobe, phase 1) come from the CPU card.

## 1. Blocks

| Block | Parts | ICs |
|---|---|---|
| **MAR** | 8 × 74F161 (D72; bits 31:2: load, +4 for LI32's MARINC), 8 × 74HCT157 (from R or the shadow PC); bits 1:0 and the controls in GAL MMUCTL | 17 |
| **Shadow PC** (D56) | 8 × 74HCT161 (loaded with the PC from R, +4 with PCINC): `MAR ← PC` needs no R cycle | 8 |
| **TLB** (split I / D, 64 + 64, direct-mapped) | 7 × CY7C199 (an entry: VPN, ASID, PFN, V, W, X); 2 × 74ACT157 (the index: MAR[17:12] and the side, or TLBHI's for TLBW); 7 × 74ACT541 (the write data from TLBHI / TLBLO); 4 × 74F521 (VPN and ASID against MAR and ASID; the '688's pinout, D72); a 74HCT32 gate (`-WE`) | 21 |
| **Faults** (as `translate()`) | GAL FAULTA (the access: kind, misaligned, privilege, translated?), GAL FAULTB (miss, protection, the cause; TLBHI takes the fault; whether the bus cycle runs), GAL FAULT2 (bus error: no `-ACK`; FAULT and its cause to the CPU; STALL) | 3 |
| **The physical address** | 5 × 74HCT157 (the PFN, or MAR in supervisor mode); 74HCT157 (A4:A2 = the level in the acknowledge cycle); 4 × 74ACT541 onto A2–A31 (D72) | 10 |
| **The bus cycle** | GAL STROBE (`-MRD`, `-MWR`, `-IACK`, BE0–3; off the bus while a DMA master has it); 4 × 74HCT541 (the store data onto D) | 5 |
| **Load steering** (as `load_extract()`) | GAL STEER; 4 × 74HCT253 (byte 0: the addressed lane), 4 × 74HCT253 (byte 1: the next lane, the vector, the fill), 4 × 74HCT257 (bytes 2–3: the data or the fill) | 13 |
| **Hardware CRs** | TLBHI: 8 × 74HCT157 (from R, or the fault's address, side and ASID) and 4 × 74ABT377; TLBLO 3 × 74ABT377 (D72); ASID 74HCT377; BADADDR: 5 × 74HCT157 (MAR, or the physical address for a bus error) and 4 × 74HCT377 | 25 |
| **R's sources** | 2 × 74HCT138 (MEM, MAR, BADADDR; which CR); 74HCT138 (CR writes); onto R: MAR, BADADDR, ASID, TLBHI, TLBLO (20 × 74HCT541) | 23 |
| **Lock and arbiter** | GAL LOCKG (the lock: a locked LOADW to its STOREW, D54; the TLB tag match); GAL ARB (slot 0, the debug card, then slots 3–10: the highest request at a free, unlocked edge, kept while asked; D64); 74HC05 (`-LOCK`, open drain) | 3 |
| **Glue, pull-ups** | 74HCT08, 74HCT00, 74HCT32; 10k on `-BREQ3–10` (empty slots) and on D0–7 (no vector reads FF) | 2 |

## 2. How it works

- **MAR** loads at the commit from R (RDST = MAR), from R with the PC (PCMAR), or from the shadow PC (MARPC), and counts by 4 (MARINC). The CPU card gates all of these off in a cycle that aborts.
- **Translation.** In supervisor mode the physical address is MAR. In user mode the TLB is read in the same cycle (its index MAR[17:12], its side the fetch / data): VPN and ASID must match and V be set, else a TLB miss (TLBHI takes the faulting page, side and ASID); a store needs W, a fetch X, else PROTECT; an address in the upper half is PRIVILEGE. Misalignment is checked first, in either mode.
- **The bus cycle** runs only when no fault was found before it: the strobes for the microcycle, the store data from R (already replicated by the CPU card), the byte enables from SIZE and MAR[1:0]. No `-ACK` is a BUSERROR (BADADDR = the physical address); in the acknowledge cycle it is vector FF.
- **TLBW is written late,** like the register file: latched at the commit and written at the next cycle's `-WSTB` — that cycle is always a fetch in supervisor mode, which does not use the TLB.
- **STALL** holds the CPU while `-WAIT` is low in its cycle or a DMA master has the bus.

## 3. Verification

- Every GAL against its spec, exhaustively; FAULTA followed by FAULTB against `translate()`'s order for every input (`check_faults()`). The netlist's electrical checks.
- **Both cards together at gate level** (`generators/sim_sys.py`): the two netlists joined at J1–J3 as the backplane joins them; only memory and the devices are a model (`p32uemu`'s, as bus slaves on J1). The cross-check programs, the TLB's (`vm_test`, `vm_demo`) and interrupts' (`irq_test`: the acknowledge cycle's vector) included, run with **the state at every instruction identical to `p32emu`'s and the cycles to `p32uemu`'s**.
- Not yet exercised: STALL (no slow slave, no DMA master in the simulation), the arbiter and the lock against a DMA master.

## 4. Next

The schematic and board; a DMA master and a slow slave in the simulation (the arbiter, the lock, STALL); timing against the datasheets — the translation and the bus cycle inside one 333 ns microcycle (MAR at the commit, the TLB 15 ns, the comparators, the address on the bus, a 55 ns memory, the data back through the steering by ≈ 220 ns).

# The MMU / data card: theory of operation

docs/mmu-card-theory.md

**Status:** DESCRIBES THE NETLIST AS BUILT AND VERIFIED (2026-10-26), `generators/card_mmu.py`. The card's summary, its parts by block and its verification are in the MMU / data card. The behaviour it implements is `p32uemu`'s model of it (`translate()`, `mem_cycle()`, `load_extract()`), which the gate-level simulation checks it against. The architecture it serves: `isa-system.md` (virtual memory, faults), D12, D13, D17, D51, D54, D57, D59, D62.

## 1. What the card does

The MMU / data card stands between the CPU card and the system bus. In every microcycle it does what the CPU card's microword asks for over J3, the local link:

- it keeps **MAR**, the memory address register, and a **shadow PC**;
- it **translates** MAR from a virtual to a physical address through the **TLB**, and checks the access (alignment, privilege, a TLB miss, protection);
- it runs the **bus cycle**: the address, the strobes and the byte enables on the backplane; the store data out; the load data back, **steered** to the bottom of R and sign- or zero-extended;
- it reports a **fault** (and its cause) in the same cycle, so the CPU card can abort it;
- it holds **TLBHI, TLBLO, ASID and BADADDR**, the control registers of the memory system;
- it runs the **interrupt acknowledge cycle** that fetches a vector (D57);
- it is the bus's **arbiter** for DMA masters, and holds the bus **lock** for `SWAP` (D51, D54).

It has **no clock of its own**. Its registers change at the CPU card's `CCLK` (the commit), its arbiter at the bus's `CLK`, and its TLB is written at the CPU card's `-WSTB`.

## 2. The microcycle, seen from this card

| When | What happens here |
|---|---|
| **The commit** (`CCLK` rising, phase 0) | MAR, the shadow PC, TLBHI / TLBLO / ASID, BADADDR and the lock take their new values. A TLB write requested by the word just finished (`TLBW`) becomes *pending*. |
| **Phases 0–2** | A pending TLB write happens, at `-WSTB` (phase 1). |
| **During the cycle** | The new word's `MEM`, `SIZE`, `RSRC` arrive on J3. MAR goes through the TLB and the comparators; the fault GALs decide; the physical address goes onto the bus with the strobes; the addressed card answers. |
| **By the end of the cycle** | Load data is steered onto R, or `FAULT` is up with its cause. The CPU card takes R, or aborts the cycle, at the next commit. |

Everything from MAR to R happens **within one microcycle**: one memory access per cycle, with no wait state at 3 MHz.

## 3. MAR and the shadow PC

**MAR** is bits 31:2 in **eight 74HCT161 counters**, plus bits 1:0 in **GAL MMUCTL**. At the commit it:

- loads **from R** when `RDST = MAR` (an address the CPU computed: the address adder, the ALU, CAUSE for the acknowledge cycle);

- loads **from the shadow PC** when `MARPC` is set (the next instruction's fetch), through eight 74HCT157s. Bits 1:0 become 0. With `RDST = PC+MAR` (a jump), R goes to both the PC and MAR.
- **counts by 4** when `MARINC` is set (`LI32` steps from the instruction to its value word and past it, D59).

**The shadow PC** is a second set of **eight 74HCT161s**. It loads from R whenever the CPU card's PC does (`RDST = PC` or `PC+MAR`) and counts with it (`PCINC`). It is an exact copy of the PC, kept here so that "MAR ← PC" for the next fetch needs neither a cycle nor the R bus (D56).

The CPU card gates `RDST`, `MARPC`, `MARINC` and `PCINC` with its abort, so an aborted cycle changes neither register.

## 4. The TLB

**The organisation** (D13, isa-system.md): **split** into an instruction half and a data half of **64 entries each**, **direct-mapped**. A virtual page's entry is at index VPN[5:0] (MAR[17:12]) in the half the access uses. An entry holds:

| Bits | Field |
|---|---|
| 0–19 | the VPN, MAR[31:12] (the tag) |
| 20–27 | the ASID |
| 28–47 | the PFN, the physical page |
| 48, 49, 50 | V (valid), W (writable), X (executable) |

51 bits, in **seven CY7C199** SRAMs (32 K × 8, 15 ns) side by side. Only 7 of their address lines are used: the 6-bit index and the half, `ISFETCH`.

**A lookup** happens in every cycle, whether or not the access is translated. MAR[17:12] and `ISFETCH` (from GAL FAULTA: this is a fetch) address the SRAMs. Their 56 outputs (the entry, `E0–E50`) feed:

- **four 74HCT688 identity comparators**: the entry's VPN against MAR[31:12], and its ASID against the ASID register. GAL LOCKG combines their four outputs into `MATCH`;
- **the physical address**: five 74HCT157s choose the entry's PFN (user mode) or MAR[31:12] (supervisor mode: untranslated, `SUPER` from the CPU card) for address bits 31:12. Bits 11:2 are MAR's, untranslated in either mode;
- **the fault GALs**: V, W and X.

**A TLB write** (the `TLBW` instruction) is written **late**, like the CPU card's register file:

1. At `TLBW`'s commit, GAL MMUCTL registers `TLBWP`, a write pending.
2. In the next cycle, two 74HCT157s switch the SRAMs' index from MAR to TLBHI's (bits 17:12 and the I bit, 8). The SRAMs' outputs turn off (`-OE = TLBWP`), and seven 74HCT541s drive the new entry from TLBHI and TLBLO onto their data pins.
3. `-WSTB` (the CPU card's phase-1 strobe) ORed with `TLBWPN` makes `-WE`: the write happens in the middle of the window.

The cycle after a `TLBW` is always the next instruction's fetch in supervisor mode (`TLBW` is privileged and ends its instruction). That fetch is untranslated and does not need the TLB, so borrowing the SRAMs for one cycle costs nothing.

## 5. Faults: translate(), in three GALs

The checks are made in the order `p32uemu`'s `translate()` makes them, and `check_faults()` proves that for every combination of inputs.

**GAL FAULTA** decodes the access from `MEM`, `SIZE`, MAR[1:0], MAR[31] and `SUPER`:

| Output | Meaning |
|---|---|
| `MIS` | **misaligned**: a halfword at an odd address, a word not at a multiple of 4 (any mode, any kind of access) |
| `PRV` | **privilege**: a user-mode access to the upper half, `0x8000_0000` and above (device registers, the kernel's addresses) |
| `XL` | **translated**: a user-mode access that is neither of those; the TLB decides |
| `ISFETCH`, `ISSTORE` | the kind of access (the TLB half; W or X) |
| `BUSK` | the cycle is a bus kind (fetch, load, store, the locked pair, the acknowledge cycle) |

**GAL FAULTB** adds the TLB's verdict:

| Order | Fault | Cause | TLBHI loaded? |
|---|---|---|---|
| 1 | misaligned | 3 MISALIGNED | no |
| 2 | user access to the upper half | 2 PRIVILEGE | no |
| 3 | translated, and not (`MATCH` and V) | 8 TLBMISS_X (fetch), 10 TLBMISS_W (store), 9 TLBMISS_R (load) | **yes** |
| 4 | translated, a store without W or a fetch without X | 11 PROTECT | **yes** |

Its outputs are the cause (`C3–C0`), `PRE` (a fault found before the bus), `TLBF` (TLBHI takes the fault) and `BUSCYC`: the bus cycle runs only for a bus kind with no fault found (and not during reset). A faulting access never reaches the bus, so a store that faults writes nothing.

**GAL FAULT2** adds what the bus says. A bus cycle that gets **no** `-ACK` (no card at that address, a write to the ROM, a wrong size for a device register) is a **bus error**, cause 4. The exception is the acknowledge cycle, where no answer means vector `0xFF` (D57). FAULT2 drives `FAULT` and `FC3–FC0` to the CPU card on J3, and `BEF` (a bus error: BADADDR takes the *physical* address).

**What a fault leaves behind**, at the commit:

- **BADADDR** (four 74HCT377s, loaded when `FAULT` is up) takes the faulting address. Bits 11:0 are MAR's. Bits 31:12 are MAR's, or the physical page for a bus error (five 74HCT157s choose).
- **TLBHI**, for a TLB miss or a protection fault, takes the faulting page's VPN, the I bit (a fetch) and the current ASID, through eight 74HCT157s. The refill handler then only has to write TLBLO and execute `TLBW` (isa-system.md).

## 6. The bus cycle

**The address** goes onto A2–A31 through four 74HCT541s:

- bits 31:12: the physical page (§4);
- bits 11:5: MAR;
- bits 4:2: MAR, or, in the acknowledge cycle, MAR[2:0], the **interrupt level** being acknowledged (a 74HCT157). MAR holds CAUSE then, whose low bits are the level.

**GAL STROBE** drives the strobes for the whole cycle when `BUSCYC` is set: `-MRD` for a fetch, load or `LOADW`; `-MWR` for a store or `STOREW`; `-IACK` for the acknowledge cycle. It also drives the **byte enables BE0–3** from `SIZE` and MAR[1:0]: one lane for a byte, two for a halfword, all four for a word, lane 0 alone for the acknowledge cycle.

**Store data** reaches the bus through four 74HCT541s from R. The CPU card has already replicated the bytes into every lane (cpu-card-theory §12), so the data is right whichever lanes the byte enables select.

**The answer:** the addressed card pulls `-ACK` low within the first clock (no answer is the bus error above). A slow card also pulls `-WAIT`. FAULT2 then raises `STALL`: the CPU card holds its commit, and the cycle repeats with everything unchanged until `-WAIT` is released.

## 7. Load steering

A load's data comes back on D0–31 in the lanes it was read from. The CPU wants it **right-aligned on R**, sign- or zero-extended (`load_extract()`). **GAL STEER** sets up three groups of multiplexers onto R:

| R's byte | Comes from | Through |
|---|---|---|
| 0 (R7:0) | the addressed lane: MAR[1:0]'s for a byte, lane 0 or 2 for a halfword, lane 0 for a word | four 74HCT253 (dual 4:1) |
| 1 (R15:8) | lane 1 (a halfword at 0, a word), lane 3 (a halfword at 2), the fill (a byte), or **the vector** D7:0 (the acknowledge cycle) | four 74HCT253 |
| 2–3 (R31:16) | lanes 2–3 (a word), or the fill | four 74HCT257 |

**The fill** is 0, or for a signed load the sign bit (R7 for a byte, R15 for a halfword), which STEER reads back from R itself.

**In the acknowledge cycle** R carries **MAR with the vector in byte 1**: MAR's drivers come on for bytes 0, 2 and 3, and byte 1 takes D7:0. The INTR microcode writes that word into CAUSE, so CAUSE bits 15:8 hold the vector (D57). With no card answering, ten-kilohm pull-ups on D0–7 make the vector read `0xFF`.

## 8. The control registers

| Register | Parts | Written | Read (`RSRC = CRHW`, CRSEL) |
|---|---|---|---|
| **TLBHI** (bits 31:12, 8, 7:0) | 4 × 74HCT377, 8 × 74HCT157 | `MTCR` from R, or by a TLB fault (§5) | 4 × 74HCT541 |
| **TLBLO** (bits 31:12, 2:0) | 3 × 74HCT377 | `MTCR` | 4 × 74HCT541 |
| **ASID** (8 bits) | 74HCT377 | `MTCR` (the kernel at every context switch) | 74HCT541 |
| **BADADDR** | 4 × 74HCT377, 5 × 74HCT157 | by a fault | 4 × 74HCT541 |
| **MAR** (`RSRC = MAR`) | — | — | 4 × 74HCT541 |

The CPU card's control-register decode table sends **CRSEL** for the hardware CRs. One 74HCT138 decodes the writes (`RDST = CRHW`), another the reads. A further 74HCT138 decodes the `RSRC` codes this card serves: `MEM`, `MAR`, `BADADDR`.

## 9. The lock and the arbiter

**The bus lock** (D51, D54): `SWAP` reads with `LOADW` and `LOCKREQ`, then writes with `STOREW`, and no DMA master may come between the two. GAL LOCKG registers `LOCKED` at the locked `LOADW`'s commit and clears it at the `STOREW`'s, or at a fault (a `SWAP` that faults is restarted whole). A 74HC05 puts it on the backplane's open-drain `-LOCK`.

**The arbiter** (D50, D64), **GAL ARB**, clocked by `CLK`. Each slot from 3 to 10, and the debug card's slot 0, has a radial pair, `-BREQn` in and `-BGNTn` out. Ten-kilohm pull-ups keep an empty slot's request off.

- At a `CLK` edge with no grant out and `-LOCK` high, the requesting slot of the **highest priority** gets its grant: slot 0 first (the debug card can always borrow the bus), then the highest-numbered slot.
- A master **keeps the bus as long as it asks**. When it releases `-BREQ`, its grant drops at the next edge.
- While a grant is out (`DMAOWN`), this card's address drivers, strobes and store data are all off the bus. If the CPU wants a bus cycle meanwhile, FAULT2's `STALL` holds it.

## 10. A load, end to end

`LW r1, 8(r2)` in user mode, the page in the TLB, the second word of the instruction (cpu-card-theory §17):

1. **The commit before:** MAR ← r2 + 8 (`RDST = MAR` from the CPU card's address adder).
2. **During the cycle:** the word on J3 says `MEM = LOAD`, `SIZE = word`, `RSRC = MEM`. MAR[17:12] and `ISFETCH = 0` index the data half of the TLB; the 688s compare the VPN and ASID; `MATCH` and V are set. FAULTA says aligned, user, translated; FAULTB finds no fault and raises `BUSCYC`. The physical address (PFN, MAR[11:2]) goes onto A2–A31 with `-MRD` and all four byte enables.
3. The memory card in that window answers `-ACK`, and its SRAMs put the word on D0–31. STEER passes the lanes straight through to R.
4. **The commit:** the CPU card latches R for r1.

Had the page been missing, FAULTB would have raised `PRE` with TLBMISS_R, `BUSCYC` would have stayed low (no bus cycle), and FAULT2 would have raised `FAULT`. At the commit, BADADDR and TLBHI would have taken the address and page, and the CPU card would have aborted the cycle and entered the trap.

## 11. Timing

At 3 MHz the whole path above must fit one 333 ns cycle (*estimates, to be checked against the datasheets*):

| Step | Approximately |
|---|---|
| MAR valid after the commit (74HCT161) | 25 ns |
| the TLB's SRAMs (15 ns) | 40 ns |
| the 688s, LOCKG's `MATCH`, FAULTB's `BUSCYC` (GALs ≈ 15 ns each) | 90 ns |
| the address and strobes on the backplane | 110 ns |
| the memory card: decode, a 55 ns SRAM, the transceivers | ≈ 200 ns |
| the steering onto R | ≈ 220 ns |
| the CPU card's set-up before the commit | ample |

The tightest part is the fault decision before the strobes. It is also why 5 MHz would need translation moved earlier (backlog).

# The memory card

docs/mem-card.md

**Status:** NETLIST DONE AND VERIFIED with the CPU and MMU / data cards (2026-10-26); layout to do. Any slot from 3 to 10, 9U × 340 mm (D49), through-hole (D21). D50, D52, D60, D63. The netlist is `generators/card_mem.py`.

How it works, block by block: the theory of operation.

**48 ICs, 3 GALs** (`python3 generators/card_mem.py --bom`).

## 1. What it holds

- **16 MiB of static RAM in eight banks of 2 MiB** (D60): each bank is four AS6C4008 (512K × 8, 55 ns), one per byte lane. Fit the banks in order from bank 0: a card holds 2, 4, … 16 MiB.
- **The 1 MiB boot ROM** (four SST39SF020A, 256K × 8, one per lane) on the card in window 0, over the first MiB (the RAM under it is hidden).

## 2. Jumpers (at the card's edge, labelled)

Fitted = 0, open = 1.

| Jumpers | Meaning |
|---|---|
| `WIN1 WIN0` | the card's window: n × 16 MiB (D52); the first card is window 0, both fitted |
| `BANKS2 BANKS1 BANKS0` | the number of banks fitted, minus 1: all open = 8 banks (16 MiB); all fitted = 1 bank (2 MiB) |
| `ROMEN` | open: this card has the boot ROM (window 0's card); fitted on the others |

## 3. How it works

- **GAL MSEL** selects the card: A31:A26 zero, A25:A24 = the window.
- **GAL MBANK**: the ROM region (the first MiB, if the ROM is fitted) or a fitted bank (A23:A21 ≤ the jumpers); `-ACK` (open drain) only there — and not for a write to the ROM, so that is a bus error, as are addresses in unfitted banks. The data transceivers' enable and direction.
- **A 74HCT138** picks the bank's four chips; **GAL MWE** gives each byte lane its `-WE` (the enabled lanes only, in the bus clock's second half, the address and data settled since the first) and the read's `-OE`.
- 74HCT541 address buffers and 74HCT245 data transceivers: one load per line on the backplane.
- 55 ns RAM, 70 ns ROM: no wait state at 3 MHz.

## 4. Verification

Its GALs against their specs, exhaustively; the netlist's electrical checks; and **with the CPU and MMU / data cards at gate level** (`generators/sim_sys.py --mem`): the ROM's program read through its four lanes, the RAM through the 32 SRAMs, a write to the ROM a bus error — the programs run with every instruction's state as `p32emu`'s and every cycle count as `p32uemu`'s. `make test` runs one.

The emulators take each window's size (`-M 6,0,4`), and the kernel probes in 2 MiB steps and uses every bank, holes included (D60; `make test` boots on 6 + 0 + 4 MiB). The boot ROM's monitor keeps its stack in the first bank. To do: the layout.

# The memory card: theory of operation

docs/mem-card-theory.md

**Status:** DESCRIBES THE NETLIST AS BUILT AND VERIFIED (2026-10-26), `generators/card_mem.py`. The card's summary and jumpers are in the memory card. D50, D52, D60, D63.

## 1. What the card does

A memory card is a **bus slave**: it never starts a cycle, it only answers the ones addressed to it. It holds:

- **up to 16 MiB of static RAM**, in **eight banks of 2 MiB**. A bank is four AS6C4008 chips (512 K × 8, 55 ns), one per byte lane. Banks are fitted in order from bank 0, so a card can hold 2, 4, … 16 MiB (D60).
- optionally **the 1 MiB boot ROM**: four SST39SF020A flash chips (256 K × 8, 70 ns), one per lane. Only the card in window 0 has them fitted and enabled.

Up to four memory cards share the bus, each in its own **16 MiB window** chosen by jumpers (D52), not by its slot. The physical address map:

| Physical address | What |
|---|---|
| `0x0000_0000`–`0x000F_FFFF` | the boot ROM (on window 0's card, with ROMEN open), hiding the RAM under it; a write is a bus error |
| `0x0000_0000`–`0x00FF_FFFF` | window 0's card |
| `0x0100_0000`–`0x01FF_FFFF` | window 1's card |
| `0x0200_0000`–`0x02FF_FFFF` | window 2's |
| `0x0300_0000`–`0x03FF_FFFF` | window 3's |

Within its window a card answers only in its fitted banks. An access to a bank that is not fitted, or to a window with no card, gets no `-ACK`, and the MMU / data card turns that into a **bus error**. That is how the kernel finds the memory at boot (boot §5.2).

## 2. The jumpers

Six jumpers at the card's edge. **Fitted reads 0, open reads 1**: each jumper shorts its line to ground, and a 10k network pulls it up.

| Jumpers | Meaning |
|---|---|
| `WIN1 WIN0` | the window, 0–3 (both fitted: window 0) |
| `BANKS2 BANKS1 BANKS0` | the number of banks fitted **minus one**: all fitted = 1 bank (2 MiB), all open = 8 banks (16 MiB) |
| `ROMEN` | open: this card has the boot ROM; fitted on every other card |

## 3. The decoding: three GALs

**GAL MSEL**, *is this cycle for this card?*:

- `CARDRD` = A31:A26 all zero, A25:A24 = the window jumpers, and `-MRD` low;
- `CARDWR` the same with `-MWR`.

Eight address lines and two jumpers go in, and nothing else on the card looks at A24–A31.

**GAL MBANK**, *where in the card, and does it answer?*:

- `ROMR`: the ROM is fitted (`ROMEN` open) and A23:A20 = 0, the first MiB;
- `FIT`: the bank A23:A21 is fitted, i.e. bank number ≤ `BANKS`;
- `ACK`:
  - a read of the ROM region;
  - a read or write of a fitted bank outside the ROM region;
  - **not** a write to the ROM, so that is a bus error, as `p32uemu` models it;
- `ROMCSN` (the ROM chips' chip select), `RAMENN` (enables the bank decoder), `RAMWR`, and the data transceivers' enable (`XOEN`, on exactly when the card answers) and direction (`DIRW`, for a write).

**A 74HCT138** decodes A23:A21 into the eight banks' chip selects (`BANK0_N`–`BANK7_N`), enabled by `RAMENN`. Only the addressed bank's four chips are selected.

**GAL MWE**, *the strobes*:

- each lane's `-WE`: a RAM write (`RAMWR`) with that lane's byte enable (`BE0`–`BE3`), **only while `CLK` is low**, the bus clock's second half. By then the address and data have been on the bus and settled for half a cycle, so a write can never land at a wrong address while they change;
- `-OE` for every chip: a read of this card (`CARDRD`).

A 74HC05 open-drain gate puts `ACK` on the backplane's `-ACK`.

## 4. The datapath

- **Address buffers:** three 74HCT541s take A2–A20 off the backplane (one load per line) onto the card's `MA0–MA18`. A RAM chip uses all 19 lines (512 K words of 4 bytes = 2 MiB per bank); a ROM chip uses 18 (256 K words = 1 MiB). Bytes are not addressed individually: each chip is one byte lane, and the byte enables pick lanes.
- **Data transceivers:** four 74HCT245s between the backplane's D0–D31 and the card's `MD0–MD31`, one per lane. They are enabled only when the card answers, pointing onto the bus for a read and onto the card for a write.
- Each RAM chip's eight data pins are on its lane of `MD`. Its `-CE` is its bank's select, its `-OE` the read enable, its `-WE` its lane's write strobe. The ROM chips share the lanes; their `-CE` is `ROMCSN`, and their `-WE` is tied high (on the bus they can only be read; they are programmed off the card: writing them through the debug card was dropped, D101).

## 5. A read and a write, in time

One bus cycle is one `CLK` period, 333 ns at 3 MHz (CLK high in the first half).

**A read** (a load, or a fetch):

1. From the start of the cycle the MMU / data card drives the address, the byte enables and `-MRD` (mmu-card-theory §6). Its translation takes most of the first ≈ 110 ns.
2. MSEL and MBANK decode (two GAL delays, ≈ 30 ns). `-ACK` goes low, the bank decoder selects four chips, `-OE` goes low, and the transceivers turn on toward the bus.
3. The SRAMs answer within **55 ns** (the ROM's flash within 70 ns), and the data crosses the 245s onto D0–D31 at ≈ 200 ns.

4. The MMU / data card steers it onto R, and the CPU card takes it at the commit that ends the cycle.

**A write** (a store):

1. The address, the byte enables, `-MWR` and the data come from the start of the cycle.
2. `-ACK` as for a read; the transceivers point onto the card.
3. When `CLK` falls (mid-cycle), MWE drives `-WE` low for each enabled lane. It rises again with `CLK` at the end of the cycle, and the SRAMs take the data on that rising edge. The bus holds the data until after the edge.

At 3 MHz the 55 ns RAM leaves well over 100 ns to spare, so the card **never needs a wait state** and does not use `-WAIT`.

## 6. Fitting a card

- Fit RAM banks **in order from bank 0** and set `BANKS` to the number fitted minus one.
- Window 0's card must have **at least bank 0** (the monitor's stack and the kernel live in the first 2 MiB, D60), and its ROM chips with `ROMEN` open.
- Every other card: `ROMEN` fitted, its own window. Two cards jumpered to the same window would both answer, and fight on D0–31.
- The kernel finds what is fitted by itself: no software setting is needed.

## 7. Verified

GAL MSEL, MBANK and MWE are checked against their specs for every input. The card runs at gate level with the CPU and MMU / data cards (`generators/sim_sys.py --mem`): the boot program comes out of its four ROM chips, its RAM is 32 simulated SRAMs, a write to the ROM is a bus error, and the programs' state and cycles match the emulators exactly. `make test` runs one such program.

# The I/O card

docs/io-card.md

**Status:** NETLIST DONE AND VERIFIED with the CPU, MMU / data and memory cards (2026-10-26); layout to do. Any slot from 3 to 10, 9U × 340 mm (D49), through-hole (D21). D23, D43, D45, D50, D52, D57, D65. The netlist is `generators/card_io.py`.

**86 ICs, 16 of them GALs**, plus a Raspberry Pi Pico 2 W and a WIZ850io module; 255 parts (`python3 generators/card_io.py --bom`).

## 1. What it holds

The devices `emulator/p32io.h` models for the I/O card, in its 4 KiB window (window 7, `0xFFFF_F000`, by default; D52), registers in the top 256 bytes:

| Address | Device | Level (default) |
|---|---|---|
| `FF00`–`FF07`, `FF08`–`FF0F`, `FF30`–`FF37`, `FF38`–`FF3F` | **four UARTs**, TL16C550 (1.8432 MHz clock), each on a **DB9 plug at the front edge** through a MAX232A (TXD, RXD, RTS, CTS at RS-232 levels, wired as a PC's COM port) | 2, 3, 2, 3 |
| `FF10` TCOUNT, `FF14` TRELOAD, `FF18` TCTRL | **the timer**: 32 bits, 1 MHz, reload, pending (words only) | 0 |
| `FF20`, `FF21` | **the input and output ports**, 8 bits each, on 10-pin IDC headers | — |
| `FF40`, `FF41` | **the real-time clock**, DS12C887 (D23): register number, register | 5 |
| `FF50`/`51`, `FF54`/`55` | **two USB ports**: CH376S modules in 2 × 8 sockets (D43) | 6, 6 |
| `FF52`/`53` | **the network coprocessor's mailbox** (D45): a Pico 2 W and a WIZ850io (W5500, RJ45) | 7 |
| `FFF0`, `FFF1`, `FFF4` | card ID (1), revision (0), interrupt status (a word: bit per device) | — |

Byte registers answer byte accesses only, the timer and `FFF4` words only. Anything else in the window gets no `-ACK`, which is a bus error, exactly as `p32io.h` behaves.

## 2. Jumpers (at the card's edge, labelled)

Fitted = 0, open = 1 (10k pull-ups). **30 jumpers:**

| Jumpers | Meaning | Default |
|---|---|---|
| `WIN2 WIN1 WIN0` | the card's window, 0–7 | 7: all open |
| `L0_2 L0_1 L0_0` | the timer's interrupt level | 0 |
| `L1_*` … `L4_*` | UARTs 0–3 | 2, 3, 2, 3 |
| `L5_*` | the clock | 5 |
| `L6_*`, `L7_*` | USB 0, USB 1 | 6, 6 |
| `L8_*` | the network | 7 |

Each device's level is its own three jumpers (D52). Devices may share a level, and the acknowledge cycle still tells them apart.

## 3. Blocks

| Block | Parts | ICs |
|---|---|---|
| **Bus interface** | 2 × 74HCT541 (A2–A7, BE0–3, `-MRD`, `-MWR`, `-IACK`, CLK, `-RES`: one load per line); 2 × 74HCT688 (A31–A16 all ones); GALs IOSEL (the window), IOBE (size, lane), IODEC1 (the byte devices' selects), IODEC2 (the other registers), IOSTB (strobes, byte lanes, `-ACK`); 4 × 74HCT245 (D lanes ↔ the byte bus ID0–7) | 13 |
| **UARTs** | 4 × TL16C550, the 1.8432 MHz oscillator; 4 × MAX232A; 4 × DB9 (right angle, front edge) | 9 |
| **Timer** | 8 × 74HCT163 (~TCOUNT, counting up); 4 × 74HCT374 (~TRELOAD, on the load bus LD); 12 × 74HCT540 (D → LD, LD → D, count → D, each inverting); GALs TIMDEC (the paths), TIMCTL (enable, pending, prescaler) | 26 |
| **Status word** | 74HCT541, 74HCT125 (the devices' requests → the status bus SB), 4 × 74HCT541 (SB → D: TCTRL, `FFF4`); pull-downs | 6 |
| **Ports** | 74HCT541 in, 74HCT574 out, 74HCT541 read-back | 3 |
| **Clock** | DS12C887 | 1 |
| **USB** | two CH376S modules in sockets | — |
| **Network** | 74HCT574 request latch, 74LVC244 (to the Pico at 3.3 V), 74HCT574 reply latch, 74HCT244 (NSTAT), CD74HC4050 (TXE, RXF to the Pico), GALs NSTB (strobes) and MBOX (TXE, RXF, IEN, RESET); LD1117V33; the Pico and the WIZ850io | 8 |
| **Interrupts** | 9 × 74LS145 (one per device: its level jumpers decoded onto that `-IRQ` line, open collector); 2 × 74HCT14 (request polarities); 74HCT138 (the level acknowledged); GALs IVA–IVE (each device: asking on that level?), IOPRI (the lowest), IOVEC (ID, revision, the vector onto the byte bus); 74HC05 (`-ACK`, the Pico's RUN) | 19 |

## 4. How it works

- **Decoding:** IOSEL matches A31:A8 against the window. IOBE turns the byte enables into *byte* (exactly one), *word* (all four) and the lane, LN1:LN0. A byte register's address is its word's address plus the lane, and the lane is its chip's A1:A0. IODEC1 and IODEC2 select one device or register, and only for an access of the right size.
- **Strobes in the cycle's second half:** the byte devices' `-RD` / `-WR`, the latches' clocks and the clock chip's AS are all gated by CLK low. By then the address has been stable for half a cycle, so a strobe can never fire on a wrong address while the decode settles. (Found by the simulation: AS in the first half fell on a passing glitch and latched a wrong register number.)
- **The byte devices take two clocks (D72):** the UARTs, the clock, the USB modules, the mailbox, the ports and the ID answer with one wait state. GAL IOSTB's W1 (a register on CLK) marks the second clock and `-WAIT` (a 74HC05 gate) holds the first. `-RD` runs from the first clock's middle to the cycle's end, so the data stays until the master takes it; `-WR` from the first clock's middle to the second's, a whole clock with half a clock of hold. GAL NSTB's latch clocks and AS act in the second clock; the mailbox's flags see one clock of the access. Without it a read could not get back to the CPU in time (docs/timing.md, D71).
- **The timer's reload latch takes D at the cycle's end** (D72), and its counters are 74F163s: their carry chain to ALLONES (eight stages) must settle within a clock of a load, which 74HCT's 40 ns a stage does not.
- **The byte bus:** the four lane transceivers put the selected byte lane on ID0–7, where every byte device sits. Word registers (the timer, the status word) sit on D0–31 directly.
- **The timer:** P32's design (D11). The counter holds the complement of TCOUNT and counts up on CLK every third clock (1 MHz); every fourth when the bus line `-FAST` is low, the CPU card's clock jumper at 4 MHz (D74), so the tick stays 1 MHz and the kernel needs no change. Its carry out of the top (`ALLONES`) means TCOUNT = 0. At that edge, while enabled, it reloads from the reload register and sets pending, exactly as `p32uemu -C`'s `io_hwedge()`.
- **Interrupts:** each device's request goes to its 74LS145, whose open-collector outputs are wired to `-IRQ0`–`-IRQ7`; the jumpers pick the output. In the acknowledge cycle (D57) the 74HCT138 decodes A4:A2, the IV GALs find

which devices are asking on that level, and IOPRI picks the lowest. IOVEC puts `window × 16 + its status bit` on D7:0, and `-ACK` answers.

- **The mailbox:** P32's design (net-mailbox.md). P32W writes the request latch. The Pico reads it through the 3.3 V buffer (`-PRD`) and writes the reply latch (`PWR`). GAL MBOX, clocked by CLK, keeps TXE and RXF, acting on each Pico strobe's leading edge once. RESET is a one-clock pulse on the Pico's RUN pin.

## 5. Verification

- **Its 16 GALs** against their specs, exhaustively; **the netlist's** electrical checks.
- **At gate level with the CPU, MMU / data and memory cards** (`generators/sim_sys.py --mem --io`): the card's own decoding, strobes, lanes, timer, status word, interrupt levels and acknowledge cycle, and its mailbox latches and flags, all wired as built. Only the insides of the TL16C550s, the DS12C887, the CH376S modules and the Pico's firmware are `p32io.h`'s models, reached through the card's pins. The timer counts as the card does, so the reference is `p32uemu -C`:
  - `hello`, `card_test` (IDs, UARTs 2 and 3, shared levels), `io_test` (UARTs, ports, the disk) and the new `iocard_test` (the timer at hardware timing, its vectored interrupt from WAIT, the status word, the clock's RAM and registers, both USB controllers, five accesses that must be bus errors): every instruction's state and every cycle as `p32uemu -C`;
  - `mbox_test` (`-N`): a HELLO request and its reply through the latches, IEN. The result is compared; the flags follow the coprocessor's own pace.
- `make test` runs `iocard_test` and `mbox_test` at gate level.

**Known differences from the emulators:** a UART's modem status reads CTS from its cable (the emulators: always asserted; no software uses it). With no coprocessor running (no `-N`), the emulators' NSTAT reads 0. The card's reads TXE (the request latch empty) and IEN as written. Only READY (0 in both) matters to the kernel.

## 6. To check before layout

- The CH376S module's header pinout and its 5 V setting (D43); the WIZ850io's pinout and the RJ45's height; the Pico 2 W's antenna corner free of copper.
- The 74LS145's pin numbers (as the 7445 / 74145 family).
- Timing against the datasheets: the TL16C550's and DS12C887's set-up and hold around the strobes' rising edge at CLK's rise.

# The I/O card: theory of operation

docs/io-card-theory.md

**Status:** DESCRIBES THE NETLIST AS BUILT AND VERIFIED (2026-10-30), `generators/card_io.py` (every chip and GAL equation comes from there). The card's summary, parts and jumpers are in the I/O card. It implements `emulator/p32io.h`'s model of the card at `p32uemu -C`'s timing, which the gate-level simulation checks it against. D11, D23, D43, D45, D50, D52, D57, D65, D71, D72, D74.

## 1. What the card does

The I/O card is a **bus slave** holding the devices below, the card ID, revision and interrupt status every card has (D52), and for each device a jumpered interrupt level and a vector (D57). Its 4 KiB window is set by jumpers: window n at `0xFFFF_8000 + n × 0x1000`, the registers in its top 256 bytes. In window 7, the default:

| Address | Register | Size | Status bit | Level (default) |
|---|---|---|---|---|
| `FFFF_FF00`–`FF07` | UART 0, the console | byte | 1 | 2 |
| `FF08`–`FF0F` | UART 1 | byte | 2 | 3 |
| `FF10`, `FF14`, `FF18` | the timer: TCOUNT, TRELOAD, TCTRL | word | 0 | 0 |
| `FF20`, `FF21` | the input port, the output port | byte | — | — |
| `FF30`–`FF37`, `FF38`–`FF3F` | UARTs 2 and 3 | byte | 3, 4 | 2, 3 |
| `FF40`, `FF41` | the clock: register number, register | byte | 5 | 5 |
| `FF50`, `FF51` | USB 0: data; command and status | byte | 6 | 6 |
| `FF52`, `FF53` | the mailbox: NDATA; NSTAT / NCTRL | byte | 8 | 7 |
| `FF54`, `FF55` | USB 1, as USB 0 | byte | 7 | 6 |
| `FFF0`, `FFF1` | card ID (1), revision (0) | byte | — | — |
| `FFF4` | interrupt status | word | — | — |

Any other address in the window, or a register at the wrong size, gets no `-ACK`: a bus error, as `p32io.h` behaves.

## 2. The jumpers

Thirty, along the bottom edge. **Fitted reads 0, open reads 1**: each shorts its line to ground, and a 10k network (RN3–RN6) pulls it up.

| Jumpers | Nets | Meaning | Default |
|---|---|---|---|
| JP1–JP3 | `WIN0`–`WIN2` | the window, 0–7 | 7: all open |
| JP4–JP30, three a device | `Ld_0`–`Ld_2`, d = the status bit | the level of the timer, UARTs 0–3, the clock, USB 0, USB 1, the network | 0, 2, 3, 2, 3, 5, 6, 6, 7 |

A device's jumpers read as its level, `Ld_2` the high bit: the timer's level 0 is all three fitted, level 2 `Ld_1` open and the other two fitted. Devices on this card may share a level. **No other card may use one of its levels:** both would answer the acknowledge cycle (bus.md, D57).

## 3. The bus interface and the decode

Two 74HCT541s buffer A2–A7 (`BA2`–`BA7`), BE0–BE3 (`BBE0`–`BBE3`), `-MRD`, `-MWR`, `-IACK`, CLK (`BCLK`) and `-RES` (`BRESN`), one load per line. Two 74HCT688s find A31:A16 all ones (`HI1N`, `HI2N`); A8–A15 go straight to GAL IOSEL.

- **GAL IOSEL**, *is it this card's cycle?*: A31:A16 all ones, A15 = 1, A14:A12 = the window jumpers, A11:A8 = F; then `CRD` with `-MRD`, `CWR` with `-MWR`, `CACC` with either. `INTA` is `-IACK` low, at any address.
- **GAL IOBE**, *what size?*: `BYTE` (exactly one byte enable), `WORD` (all four), and `LN1:LN0`, a byte's lane. A byte register's offset is A7:A2 × 4 plus its lane; the lane is a UART's A1:A0 and a USB module's A0.
- **GAL IODEC1**, the byte devices' selects, for a byte access in range: `U0N`–`U3N` (eight bytes each), `USB0N`, `USB1N` (two each) and `RTCDN` (`FF41`), active low; `NETDS` (`FF52`), `NETSS` (`FF53`); `BHIT1`, any.
- **GAL IODEC2**: `SRTCI` (`FF40`), `SPIN`, `SPOUT` and `SIDR` (`FFF0`–`FFF1`) for a byte access, `BHIT2` any of them; `STCNT`, `STREL`, `STCTL`, `SISR` for a word access at `FF10`, `FF14`, `FF18`, `FFF4`, `WHIT` any of them.

Every select needs the right size, so a wrong size selects nothing. **The byte bus** `ID0`–`ID7` carries every byte device. Four 74HCT245s join it to D's lanes; GAL IOSTB enables only the access's lane (`LE0N`–`LE3N`), pointing onto the card for a write (`LDIR` = `CWR`) and onto the bus otherwise. The word registers sit on D0–D31 directly.

## 4. A byte access takes two clocks (D72)

With its strobe from mid-cycle, a byte read did not fit 3 MHz (D71), so every byte register answers with one wait state, made by **GAL IOSTB** on CLK: `W1`, a register, marks the access's second clock; `WAIT`, its first (§13); `IORDN` (`-RD`) runs from mid clock 1 to the cycle's end, so the data stays until the master takes it; `IOWRN` (`-WR`) from mid clock 1 to mid clock 2, a whole clock with half a clock of hold.

| | Clock 1, CLK high | Clock 1, CLK low | Clock 2, CLK high | Clock 2, CLK low |
|---|---|---|---|---|
| `W1` | 0 | 0 | 1 | 1 |
| `-ACK` | low, from the decode | low | low | low |
| `-WAIT` | low, from the decode | low | high | high |
| `-RD` (a read) | high | low | low | low |
| `-WR` (a write) | high | low | low | high |
| NSTB's `REQCLK`, `POCLK`, `RTCAS` (a write) | low | low | low | high |
| NSTB's `NDWR`, `NDRD`, `CTLWR` | 0 | 0 | 1 | 1 |

At the edge ending clock 1 the master sees `-WAIT` and holds everything for another clock (bus.md); the MMU / data card holds the CPU with `STALL` (mmu-card-theory §6). The edge ending clock 2 ends the cycle: a read's data is taken, AS falls, GAL MBOX acts, `W1` clears. `-RD` and `-WR` reach every UART, the clock and both USB modules; only the selected one acts.

## 5. The UARTs and RS-232

Four TL16C550s, UART k selected by `-CS2` = `UkN` (CS0, CS1 high). D0–D7 are on the byte bus, A2:A0 = `BA2`, `LN1`, `LN0`; `-RD1` and `-WR1` are the card's strobes (`-ADS`, RD2, WR2 tied low); MR is `-RES` inverted, by GAL TIMCTL. One **1.8432 MHz oscillator** drives all four XINs, and each one's `-BAUDOUT` is its receive clock: baud = 1.8432 MHz / (16 × divisor) (programmers-guide §6.1). INTRPT, active high, is its request (`UINTk`).

A **MAX232A** each: SOUT → TXD and `-RTS` → RTS out, RXD → SIN and CTS → `-CTS` in, with four 100 nF pump capacitors. A **DB9 plug** each (J4–J7) at the front edge, wired as a PC's COM port: 2 RXD, 3 TXD, 5 GND, 7 RTS, 8 CTS, the shell to ground. `-DSR` and `-DCD` are tied low, `-RI` high; `-DTR`, `-OUT1`, `-OUT2` are not used. So the modem status's CTS is the cable's; the emulators always assert it (io-card.md §5).

## 6. The interval timer

P32's design (D11): **the counter holds ~TCOUNT and counts up**, so TCOUNT counts down, and its carry out (all ones) means TCOUNT = 0.

- **The counter:** eight 74F163s on CLK, in the datasheet's synchronous cascade: the first ENT high, each next ENT the previous RCO, every ENP `TICK`. The last RCO is `ALLONES`. 74F, because the eight-stage carry must settle within a clock of a load (D72). They load from **the load bus** `LD0–LD31` at an edge while `CNTLDN` is low.
- **The reload register:** four 74HCT374s hold ~TRELOAD, inputs and outputs both on the load bus, driving it except while TCOUNT or TRELOAD is written (`ROEN`).
- **Three sets of four 74HCT540s** join D and the load bus, each path inverting once, so D carries the true value. GAL TIMDEC enables them:

| Access | Path | TIMDEC's signals |
|---|---|---|
| TCOUNT write | D → 540 → LD; the 163s load at the cycle's end | `D2LDN`; `ROEN` (the 374s off); `WRCNT`, to TIMCTL |
| TRELOAD write | D → 540 → LD; the 374s take it as `RCLK` rises with CLK, at the cycle's end (D72) | `D2LDN`, `ROEN`, `RCLK` |
| TCOUNT, TRELOAD read | the 163s → 540 → D; the 374s → LD → 540 → D | `RDCNTN`; `RDLDN` |
| TCTRL write, read | D0, D1 straight into TIMCTL; TIMCTL → the status bus → D | `WRCTL`; `RDCTL`, `STOEN` |

**GAL TIMCTL**, on CLK and cleared by `-RES`, is `p32uemu -C`'s `io_hwedge()`:

- `EN` is TCTRL bit 0, as written. **The prescaler** `PS1:PS0` counts 0–2 while enabled (0–3 with `-FAST` low, the 4 MHz clock: D74), else holds 0; `TICK`, its last count, lets the counters count: 1 MHz at either clock.
- At an edge with `EN` and `ALLONES`, `CNTLDN` **reloads** the counters from the load bus and `PEND` sets; a TCOUNT write in that cycle wins the load. TCOUNT reads 0 for the cycle before (programmers-guide §6.2).
- `PEND`, TCTRL bit 1 and the request, clears at a TCTRL write with bit 1 set; an expiry at the same edge wins. `SB0` = `EN` and `SB1` = `PEND` go onto the status bus for a TCTRL read.
- Reset clears `EN`, `PEND` and the prescaler, not TCOUNT or TRELOAD (the 163s' `-CLR` is tied high; the 374s have no reset).

## 7. The status bus and the interrupt status

`SB0–SB8`, pulled down by 10k (an undriven bit reads 0), carries TCTRL or `FFF4` to D0–D8 through four 74HCT541s (`STOEN`; D9–D31 read 0). For `FFF4` (`ISROEN`) a 74HCT541 drives `SB0–SB7` and a 74HCT125 gate `SB8`. Each bit is a device's request, the same one that drives its level (§12), so reading `FFF4` changes nothing:

| Bit | 0 | 1–4 | 5 | 6, 7 | 8 |
|---|---|---|---|---|---|
| Device | the timer | UARTs 0–3 | the clock | USB 0, USB 1 | the network |
| From | `PEND` | INTRPT | `-IRQ`, inverted | INT#, inverted | `NETASK` (`ATTN` and `IEN`) |

## 8. The ports

- `FF20:` a 74HCT541 from J8 pins 1–8 (`PINRDN`), each pulled up by 10k (RN2), so an open input reads 1. A write is answered and ignored.
- `FF21:` a 74HCT574 to J9 pins 1–8, its outputs always on, clocked by NSTB's `POCLK` in the middle of a write's second clock; a 74HCT541 reads it back (`PORDN`). It has no reset.
- J8, J9: 10-pin IDC box headers, pins 1–8 bits 0–7, 9 +5 V, 10 GND.

## 9. The real-time clock

A DS12C887 (D23) in Intel bus mode (`MOT` = GND), `AD0`–`AD7` on the byte bus. **Writing** `FF40` raises NSTB's `RTCAS` (the chip's AS) in the second half of the second clock. It falls with CLK at the cycle's end, and the chip latches the register number as it falls, the data still valid. (Made in the first half, AS had fallen on a decode glitch and latched a wrong number: D65.) **Reading** `FF40` gives `0xFF` from GAL IOVEC, as `p32io.h`. `FF41` is `-CS` = `RTCDN` with the card's `-RD` and `-WR`. `-RESET` is `-RES`, `-RCLR` is high, SQW is not used. `-IRQ`, open drain with a 10k pull-up, is the request. The registers: programmers-guide §6.5.

## 10. The USB modules

Two CH376S modules (D43) in 2 × 8 sockets (J10, J11), their USB-A sockets at the front edge, wired to the module's header (usb-design.md §2):

| Pins | 1, 3, … 15 | 8 | 10 | 12 | 14, 16 | 2, 4; 6 |
|---|---|---|---|---|---|---|
| Module | D0–D7 | INT# | A0 | CS# | RD#, WR# | GND; +5 V |
| Card | the byte bus | `USBINT0N` / `1N`, 4.7k to +5 V | `LN0`: 0 data, 1 command / status | `USB0N` / `USB1N` | `-RD`, `-WR` | |

INT# low is the request. The pull-ups go to +5 V, which suits the modules run at 5 V (D43); usb-design.md §2 warns against a 5 V pull-up on a module left at 3.3 V. RSTI is not on the header: the card cannot reset a module, so the driver sends `RESET_ALL` first.

## 11. The network coprocessor (D45)

A Raspberry Pi Pico 2 W (A1) and a WIZ850io (A2: a W5500 and its RJ45) on the Pico's SPI0 (GP16 MISO, GP17 select, GP18 clock, GP19 MOSI; GP20 its reset, GP21 its interrupt). VSYS comes from +5 V through a 1N5817 Schottky diode; an LD1117V33 makes +3V3 for the WIZ850io, the 74LVC244 and the CD74HC4050.

**Registers** (net-mailbox.md §1): `NDATA` (`FF52`); `NSTAT` (`FF53` read, a 74HCT244): bit 0 `RXF`, 1 `TXE`, 2 `ATTN`, 3 `IEN`, 7 `READY`, 4–6 zero; `NCTRL` (`FF53` write): bit 3 `IEN`, bit 7 `RESET`. An NDATA write clocks the 74HCT574 **request latch** with NSTB's `REQCLK`, mid second clock; a 74LVC244 puts it on the Pico's `PD0`–`PD7` (GP0–GP7) while the Pico holds `-PRD` (GP8) low. The Pico clocks the 74HCT574 **reply latch** from `PD0`–`PD7` with `PWR` (GP9); an NDATA read puts it on the byte bus (`REPOEN`).

**GAL MBOX**, on CLK and cleared by `-RES`, keeps the flags. `-PRD` and `PWR` are asynchronous: each passes two registers, and its **leading edge** (the first set, the second not yet) acts, once.

- `TXE` resets high; an NDATA write (`NDWR`) clears it, `-PRD`'s edge sets it, the write winning a tie. `RXF`: `PWR`'s edge sets it, an NDATA read (`NDRD`) clears it, the set winning a tie.
- `IEN` takes NCTRL bit 3; `RST`, NCTRL bit 7, lasts one clock, sets `TXE` and clears `RXF`. `NETASK` = `ATTN` and `IEN`.

- `NRUN` = `RST` or `-RES`: a 74HC05 gate pulls RUN low, and 10k to the Pico's 3.3 V with 2.2 nF stretch `RESET` to about 15 µs.

`TXE` and `RXF` reach the Pico (GP10, GP11) through a CD74HC4050 at 3.3 V; `READY` (GP12) and `ATTN` (GP13) come back. Until the firmware runs, 10k resistors hold `-PRD` high, `PWR`, `READY`, `ATTN` low, and the W5500's reset and select high: the card reads `READY` = 0.

## 12. Interrupts

**The levels.** A **74LS145** per device (a BCD-to-decimal decoder with open-collector outputs): A–C its level jumpers, D its request, **active low** (a 74HCT14 inverts `PEND`, the UARTs' INTRPT and `NETASK`; the clock's and the modules' are low already). Outputs 0–7 go to `-IRQ0`–`-IRQ7`, 8 and 9 nowhere. Asking, the code is the level and that line is pulled; not asking, it is 8 + the level, and none is. Devices on one level wire-OR. The CPU takes the lowest-numbered pending, enabled level first (bus.md's Interrupts, cpu-card-theory §14).

**The acknowledge cycle** (D57): `-IACK` low and A4:A2 = n, the level the CPU takes.

1. `INTA` enables a **74F138** (a 74F for speed, D72): A4:A2 → `ACK0N`–`ACK7N`.
2. **GALs IVA–IVE**, two devices each (IVA the timer and UART 0, IVB UARTs 1 and 2, IVC UART 3 and the clock, IVD the USB modules, IVE the network): `Md` = device d asks and its level's `ACK` line is low.
3. **GAL IOPRI**: `VEC3:0`, the lowest d matching (its status bit); `VHIT`, any; and `IDOE`, IOVEC's output enable (a 22V10's is one product term): `INTA` with `VHIT`, or a read of `FFF0`–`FFF1` or `FF40`.
4. **GAL IOVEC** drives the byte bus: **the window × 16 +** `VEC` in an acknowledge cycle, else `0xFF` (`FF40`), 1 (`FFF0`) or 0 (`FFF1`); and **GAL IOSTB** turns lane 0's 245 toward the bus and raises `ACK`.

With none of its devices asking on that level the card does not answer; with no card answering, the MMU / data card's pull-ups make the vector `0xFF` (mmu-card-theory §7), and the kernel reads the cards' interrupt status instead (D57). The vector lands in CAUSE bits 15:8.

## 13. -ACK and -WAIT

IOSTB's `ACK` (a register selected, or `INTA` with `VHIT`) and `WAIT` (§4) drive the open-drain lines through 74HC05 gates, `-WAIT` through a spare one (D72):

| Cycle | `-ACK` | `-WAIT` | Clocks |
|---|---|---|---|
| a byte register, a byte access | yes | the first clock | 2 |
| TCOUNT, TRELOAD, TCTRL or `FFF4`, a word access | yes | — | 1 |
| acknowledge, one of its devices asking on that level | yes | — | 1 |
| acknowledge, none of its devices asking | no | — | — |
| anything else in the window, or a wrong size | no: a bus error | — | — |

A write to `FFF0`, `FFF1`, `FFF4` or the input port is answered and ignored, as `p32io.h`.

## 14. Two cycles, end to end

`lbu r10, 0xFFFFFF05(r0)`, UART 0's LSR:

1. **Clock 1:** the MMU / data card drives the address, BE1 and `-MRD` (mmu-card-theory §6). IOSEL gives `CRD`, IOBE `BYTE` in lane 1, IODEC1 `U0N`: A2:A0 = 1, 0, 1, register 5. IOSTB pulls `-ACK` and `-WAIT` and turns lane 1's 245 toward the bus; mid-clock `-RD` falls, and LSR crosses the byte bus to D15:8.
2. **Clock 1's end:** `-WAIT` is low, so the cycle repeats unchanged and the CPU holds its commit; `W1` sets. **Clock 2's end** ends the cycle: the MMU / data card has steered lane 1 onto R, zero-extended, and the CPU commits it to r10. Two clocks: 667 ns at 3 MHz.

**The timer's interrupt**, at the default jumpers:

1. Enabled, a tick takes TCOUNT to 0 (`ALLONES`). At the next edge TIMCTL reloads and sets `PEND`; the timer's 74LS145 pulls `-IRQ0`, and `FFF4` bit 0 reads 1.
2. The CPU card samples `-IRQ0`; at an instruction's end, with `IE` set and level 0 enabled, it enters INTR (cpu-card-theory §14).
3. The acknowledge cycle, A4:A2 = 0: `ACK0N`, IVA's `M0`, IOPRI's `VEC` = 0 and `VHIT`; IOVEC drives 7 × 16 + 0 = `0x70`, IOSTB answers `-ACK`. CAUSE becomes `0x8000_7000`.
4. The handler writes 3 to TCTRL: at that cycle's end `PEND` clears, `EN` stays, and the 74LS145 lets `-IRQ0` go.

## 15. Timing

| What | How long |
|---|---|
| CLK | 333 ns at 3 MHz; 250 ns with the clock jumper (4 MHz, D74) |
| a word register; an acknowledge cycle | one clock |
| a byte register (D72) | two clocks: `-RD` 1½ clocks to the cycle's end; `-WR` one clock, then ½ clock of hold |
| the timer's tick | every third clock (every fourth at 4 MHz): 1 µs |
| `RESET`; the Pico | one clock on `NRUN`, about 15 µs on RUN; it answers within a microsecond (net-mailbox.md §1) |
| the CH376S (usb-design.md §1) | pulses of at least 30 ns (write) and 40 ns (read); at least 1.5 µs after a command byte and 0.6 µs between data bytes, which P32W meets by itself |
| the whole machine | 3 MHz at the datasheets' worst case, the byte devices' second clock allowed (docs/timing.md, D72); the TL16C550's and DS12C887's set-up and hold around their strobes still to check (io-card.md §6) |

## 16. Verified

The 16 GALs are checked against their specs for every input, and their JEDEC files (`hardware/gal/io`) through the fuse map; the netlist passes its electrical checks (86 ICs, 436 nets, 255 parts). At gate level with the CPU, MMU / data and memory cards (`generators/sim_sys.py --mem --io`), only the insides of the TL16C550s, the DS12C887, the CH376S modules and the Pico's firmware are `p32io.h`'s models, behind the card's pins; the clock's `-IRQ` is not modelled. Against `p32uemu -C`:

- `iocard_test`: the timer's registers, counting, expiry, reload and pending, `FFF4` bit 0, the vectored interrupt from `WAIT` (`0x70`, level 0), the clock's RAM and registers, both USB controllers, five bus errors, a byte write to the ID ignored. Every instruction's state and every cycle match: 302 instructions, 1,178 cycles; with `-FAST` low (`p32uemu -C -F`), 1,238.
- `mbox_test` (`-N`): `READY`, a HELLO request and its reply through the latches on `TXE` and `RXF`, and `IEN`. Only the result is compared: the flags follow the coprocessor.

`make test` runs both, and has the card at gate level under the disk and video cards' tests too; `hello`, `card_test` and `io_test` ran on it only when it was built (D65).

## 17. Open questions

1. **Stale part names:** the netlist fits 74F163s and a 74F138 (D72), but io-card.md §3 says 74HCT163 and 74HCT138 (its §4 says 74F163), and so do card_io.py's comments (its docstring, `gal_iv()`, `ACKL`).
2. **"Every strobe in the second half"** (io-card.md §4, D65, card_io.py's docstring) predates D72: `-RD` is low through all of clock 2, `-WR` in its first half. NSTB's docstring likens AS falling at CLK's rise to "a UART takes a write", and io-card.md §6 checks the strobes' rise at CLK's rise, but `-WR` rises mid clock 2.
3. **The clock's interrupt:** the card gives it status bit 5 and a vector (window × 16 + 5). programmers-guide §6's `FFF4` list skips bit 5; `p32io.h`'s `card_reg()` has no bit 5 (card_io.py says it follows it), its `io_iack()` answers level 5 with `0xFF`, and `sim_sys.py` does not model the `-IRQ`. Which is intended?
4. **A TRELOAD write at the reload's edge:** the 374s are off the load bus during the write, so an expiry at its end reloads the value written; `io_hwedge()` says the old one. A scratch program writing TRELOAD while the timer ran parted the gate-level and `p32uemu -C` traces that way (`iocard_test` writes TRELOAD only when stopped). Which should change?
5. **The strobes at clock 2's start:** IOSTB's `IORDN` and `IOWRN` both hold the term `BCLK & !W1`. BCLK rises and `W1` follows its register at that edge, so `-RD` or `-WR` might rise for an instant mid-access. The timing analysis times edges and the simulator has zero delay.
6. **Small ones:** NSTAT bits 4 and 5 read 0 (grounded with bit 6), but net-mailbox.md §1 names only bit 6. bus.md's signal table (p32bus.py) says of the IRQ lines "7 highest", where its Interrupts section, D11, programmers-guide §7 and cpu-card-theory §14 take the lowest number first. io-card.md §3's interrupt block lists 20 ICs but counts 19 (the rows sum to 85 of 86), and leaves out the 74HC05's `-WAIT`.
7. **The output port at power-on:** its 74HCT574 has no reset; the emulators start it at 0. What should a device on J9 see first?

# The disk card

docs/disk-card.md

**Status:** NETLIST DONE (2026-10-26), verified at gate level with the CPU, MMU / data, memory and I/O cards (§5); layout to do. Any slot from 3 to 10, 9U × 340 mm (D49), through-hole (D21). D26, D50, D52, D57, D66. The netlist is `generators/card_disk.py`.

**80 ICs, 15 of them GALs**; 205 parts (`python3 generators/card_disk.py --bom`). 41 of the ICs and 6 of the GALs are the **DMA engine** (D75, §4.1).

## 1. What it holds

**Four ATA ports**, programmed I/O, 16 bits wide (D26), in the card's 4 KiB window (window 6, `0xFFFF_E000`, by default; D52):

| Port | Registers | Connector | Kernel |
|---|---|---|---|
| 0 | `FFFF_EF00` – `EF0F` | right-angle 40-pin header at the front edge: a **CF-to-IDE adapter** plugs on, its slot facing out (the boot card) | `/dev/disk`, minor 0 |
| 1 | `EF10` – `EF1F` | the same: the second CF card | `/dev/disk1` |
| 2 | `EF20` – `EF2F` | boxed 40-pin IDE header: channel A, master and slave | `/dev/hda`, `hdb` |
| 3 | `EF30` – `EF3F` | channel B | `/dev/hdc`, `hdd` |

All four ports are the same circuit. The CompactFlash cards sit in off-the-shelf CF-to-IDE adapters (Ken's choice, D66), because every CF socket is a surface-mount part.

**Each port's 16 bytes:**

| Offset | Read | Write |
|---|---|---|
| +0 | **data**: a halfword (16 bits, D26) or a byte | data |
| +1 | error | features |
| +2 … +5 | sector count, LBA 7:0, 15:8, 23:16 | the same |
| +6 | drive / head (bit 4: master or slave; bits 3:0 LBA 27:24) | the same |
| +7 | status (reading it acknowledges INTRQ) | command |
| +8 … +15 | the control block (CS1): +14 alternate status | +14 device control |

`FFFF_EFF0` card ID (2), `EFF1` revision (**1**: the DMA engine, D75), `EFF4` **interrupt status** (a word: bit *p* = port *p*'s INTRQ, bit 4 = the DMA engine). The data register answers halfwords and bytes, the other port registers bytes, `EFF4` and the DMA registers words; anything else gets no `-ACK`, which is a bus error.

**The DMA engine's registers** (D75; words):

| Address | Read | Write |
|---|---|---|
| `FFFF_EF40` DMAADR | the physical address of the next word | the first word's (bits 1:0 ignored) |
| `EF44` DMACNT | the words left (16 bits) | the words to move (0–65535: 256 KiB) |

| Address | Read | Write | |
|---|---|---|---|
| `EF48` DMACTL | bit 0 BUSY, 1 DIR, 3:2 the port, 4 IE, 5 MODE, 6 DONE, 7 ERR | bit 0 RUN (1 starts; 0 stops after the word in hand), 1 DIR (0: the drive to memory; 1: memory to the drive), 3:2 the port, 4 IE, 5 MODE (multiword DMA: reserved, write 0); a write clears DONE and ERR | **A port with nothing on it reads 00** in |

every register (pull-downs on its data lines); the kernel's probe sees "not ready" and leaves it alone.

## 2. Jumpers (at the card's edge, labelled)

Fitted = 0, open = 1:

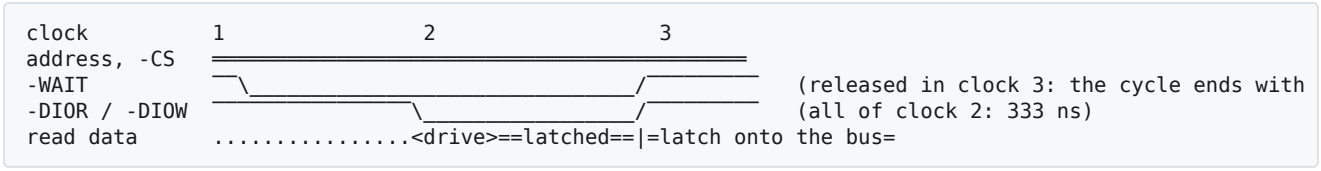
| Jumpers | Meaning | Default |
|---|---|---|
| `WIN2 WIN1 WIN0` | the card's window | 6: WIN0 fitted, the others open |
| `L2 L1 L0` | the interrupt level of all four ports | 1: L0 open, the others fitted |

## 3. Blocks

| Block | Parts | ICs |
|---|---|---|
| **Bus interface** | 2 × 74HCT541 (A2–A7, BE, strobes, CLK, `-RES`: one load per line); 2 × 74HCT688 (A31–A16); GALs DKSEL (the window), DKBE (byte, halfword, word, lane), DKDEC (a port register, the data register's halfword, ID, status word), DKSEQ (the wait states and strobes), DKSTB (`-ACK`, the lanes); 74HC05 (`-ACK`, `-WAIT`, open drain) | 10 |
| **Card bus CD0–15** | 5 × 74HCT245 (lane 0 ↔ CD0–7, lane 1 ↔ CD8–15 for the data register, lanes 1–3 ↔ CD0–7 for a byte); 2 × 74HCT574 (the read latch); 3 × 74HCT541 (zeros on lanes 1–3 for the status word) | 10 |
| **Ports** | per port: 2 × 74HCT245 (CD ↔ DD0–15), a 74HCT541 (DA2–0, `-CS0`, `-CS1`, `-DIOR`, `-DIOW`, `-RESET`); GALs DKPORT0 / DKPORT1 (each port's selects, strobes and transceiver enable); 74HCT153 (the addressed port's IORDY); pull-downs on DD0–15, IORDY's pull-up, INTRQ's and DMARQ's pull-downs | 15 |
| **Interrupts** | 74LS145 (the card's level onto its `-IRQ` line, open collector); 74HCT85 (the acknowledged level against the jumpers); GALs DKIRQ (any source asking: the four ports, the DMA engine) and DKVEC (ID, revision, the status byte, the vector onto CD0–7: the lowest asking source) | 4 |
| **DMA engine** (D75) | GALs DKDMA (its register selects), DKDCTL (DMACTL), DKENG (the sequence, `-BREQ`), DKEOUT (enables, latch clocks, the CPU's wait), DKPSEL (the port access's address: the CPU's or the engine's), DKMWD (multiword DMA: wired, reserved); 8 × 74F163 (the address, 30 bits) and 4 × 74F163 (the count); 5 × 74HCT541 (the address, byte enables, `-MRD` / `-MWR` onto the bus as master); 4 × 74HCT541, 2 × 74HCT540 and a 74HCT541 (DMAADR, DMACNT, DMACTL read back); 2 × 74HCT540 (the count loaded inverted); 8 × 74HCT574 (the word buffers, 32 bits each way); 74HCT541 (the ports' `-DMACK`) | 41 |
| **Front edge** | the two CF headers; an activity LED per port (its drive's `-DASP`), labelled DISK 0–3 | — |

## 4. How it works

- **Two wait states on every port access** (D66). ATA's slowest PIO mode (0) wants the address 70 ns before the strobe, a strobe of 165 ns, and write data held 30 ns after it. **GAL DKSEQ**, clocked by CLK, steps through three clocks:




  The read latch takes the drive's data as `-DIOR` ends, so the drive can let go, and the latch drives the bus in clock 3. A write's data is held through clock 3. A drive that pulls IORDY low stretches clock 2. An access to the ID or the status word needs no wait states.

- **The card bus:** a halfword to the data register uses lanes 0–1, all of CD0–15. A byte register sits on its own lane and is steered onto CD0–7, so a register's address is its port's base plus its offset, as in `p32io.h`. Only the addressed port's transceivers and strobes are enabled. Its DA2–0 are the offset's low three bits, and its `-CS0` / `-CS1` come from offset bit 3.
- **Interrupts** (D52, D57): each port's INTRQ goes to GAL DKIRQ. While any port asks, the 74LS145 pulls the jumpered `-IRQ` line. In the acknowledge cycle, if the level being acknowledged (A4:A2, compared by the 74HCT85) is the card's, it answers with **window × 16 + the lowest asking port**: `0x60`–`0x63` by default. Reading a port's status register acknowledges its drive's INTRQ.

### 4.1 The DMA engine (D75)

Built in now so no later board is needed; P32W Unix starts without it (its driver moves the data itself, PIO) and can take it up with no hardware change.

- **Per sector:** the driver issues the ATA command as today; when the drive has a sector ready (DRQ), it writes DMAADR, DMACNT (128 words) and DMACTL, and the engine moves the sector while the CPU runs on. The engine does the PIO itself: every drive and CF card works.
- **A word** (drive to memory): two reads of the port's data register, exactly as the CPU's (GAL DKSEQ's three clocks), into a 32-bit buffer (4 × 74HCT574; the first halfword is the low one), then **one bus cycle as master**: `-BREQ` to the MMU / data card's arbiter, and when granted, one clock driving A2–A31 from the address counter, all four byte enables, `-MWR` and the buffer onto D. Memory to the drive is the reverse (a read into the other buffer, then two data-register writes). About 11 clocks a word; the bus is the engine's for **one clock in eleven**, so the CPU, whose cycles use the bus about two in six, is rarely held.
- **Counters:** the address (8 × 74F163, A31:2) and the count (4 × 74F163, loaded with its complement through 74HCT540s; all ones is done) advance when a word is done. A bus cycle no card answers (no `-ACK`) stops the engine with **ERR**, that word not counted: DMAADR is the failing address. The count running out sets **DONE**. With IE either is the card's fifth interrupt source: status bit 4, **vector window × 16 + 4** (`0x64` by default), acknowledged by writing DMACTL.
- **Sharing the card's byte bus:** the engine's port accesses use the same DKSEQ, port transceivers and strobes as the CPU's; GAL DKPSEL gives them the engine's port and its data register (DA2–0 = 0, `-CS0`). The CPU may use the card while the engine runs: an access that needs the byte bus CD (a port register, the ID, the status word, an acknowledge) during an engine access waits on `-WAIT` (GAL DKEOUT's EWAIT) and runs after it; the engine starts its next access only when no such access is asked (CDREQ). The DMA registers are on D directly and never wait.
- **Multiword DMA wired, reserved:** each port's DMARQ goes to GAL DKMWD and its `-DMACK` comes from it through a 74HCT541; MODE and the selected port's DMARQ reach the sequencer, and DMAM the port GALs (to negate the chip selects). As programmed now `-DMACK` is never asserted and MODE does nothing: the drives' own DMA is a matter of reprogramming the socketed GALs DKENG, DKMWD and DKPORT0–1, not of a new board.

## 5. Verification

- Its 9 GALs against their specs, exhaustively; the netlist's electrical checks.
- **At gate level with the CPU, MMU / data, memory and I/O cards** (`generators/sim_sys.py --mem --io --disk`): only the drives' insides are `p32io.h`'s model, reached through each port's 40 header pins. The card's wait states are the first test of the MMU / data card's `STALL` path. `p32uemu -C` adds the same two wait states to every port access, so the reference stays exact:

- `card_test` (the card's ID and status word);
- `io_test` (the boot CF port: reads, a written sector, the disk image identical);
- the new `disk_test`: an empty port reads 00; IDE A's master and slave each read; CF 1 written and read back; the status word; a port's interrupt through its vector `0x62`.

Each runs with every instruction's state and every cycle as `p32uemu -C`. `make test` runs `disk_test`.

- **The DMA engine** (D75): `diskdma_test` at gate level, the engine a bus master through the MMU / data card's gate-level arbiter into the gate-level memory card: the registers; a sector read in two transfers while the CPU keeps reading the card's ID, status word, an empty port and DMACTL (each held while the engine has the byte bus); the sector written to CF 1 from memory and read back by DMA; the done interrupt through vector `0x64`; a transfer to an address no card answers, stopped with ERR. Self-checking (the CPU polls, so its cycle count is its own), and the same program passes on `p32emu` and `p32uemu -C`, which model the engine; the sector it writes is the same on all three.

## 6. To check before layout

- The CF-to-IDE adapter in hand: that it plugs onto a right-angle header with its slot facing out of the front edge, and its height (D48: parts up to about 20 mm).
- The 40-pin ATA pinout (standard) and the 74LS145's pin numbers.
- Timing against the datasheets: the drive's data valid before `-DIOR` rises into the 74HCT574s.

# The disk card: theory of operation

docs/disk-card-theory.md

**Status:** DESCRIBES THE NETLIST AS BUILT AND VERIFIED (2026-10-30), `generators/card_disk.py` (every chip and GAL equation comes from there). The card's summary, its parts by block and its jumpers are in the disk card; its registers as programs see them in the programmer's guide §6.4; the bus cycle, mastership and interrupts in the bus. D26, D50, D52, D57, D66, D75.

## 1. What the card does

The disk card is a **bus slave** for its registers and, while its DMA engine runs, a **bus master** (D75). It holds four **ATA ports**, programmed I/O with the data register 16 bits wide (D26, §2); the **card bus CD0–15** between the bus's byte lanes and the ports (§4); the **port sequencer**, which gives every port access two wait states (§5); the **interrupt logic**, one vectored level for the card (§7); and the **DMA engine** (§8). Its registers are the top 256 bytes of its 4 KiB window (D52): `0xFFFF_EF00`–`EFFF` in the default window 6.

| Offset | Register | Answers |
|---|---|---|
| `+00`–`+3F` | port *p* at `16 p` (§2) | the data register (`+0`): a halfword or a byte; the others: a byte |
| `+40`, `+44`, `+48` | DMAADR, DMACNT, DMACTL (§8.1) | a word |
| `+F0`, `+F1` | card ID 2, revision 1 (the DMA engine) | a byte |
| `+F4` | interrupt status (§7) | a word |

Any other address or size in the window gets no `-ACK`: a bus error. A write to the ID, the revision or the status word is acknowledged and does nothing.

## 2. The ports

| Port | Offsets | Connector |
|---|---|---|
| 0 | `+00`–`0F` | J4, a right-angle 40-pin header at the front edge: a CF-to-IDE adapter plugs on, its slot facing out (the boot card) |
| 1 | `+10`–`1F` | J5, the same: the second CF card |
| 2 | `+20`–`2F` | J6, a boxed 40-pin IDE header: channel A, master and slave |
| 3 | `+30`–`3F` | J7, the same: channel B |

The four ports are one circuit, four times:

- **two 74HCT245s** between CD0–15 and the cable's DD0–15, enabled only in that port's access, toward the drive for a write;
- **a 74HCT541** driving DA2–0, `-CS0`, `-CS1`, `-DIOR`, `-DIOW` and `-RESET` (the bus's `-RES`);
- **10k pull-downs on DD0–15**: an empty port reads 00 everywhere, so its status says "not ready". IORDY 1k up; INTRQ 10k (no drive, no interrupt) and DMARQ 5.6k down; CSEL grounded; `-IOCS16`, `-PDIAG` not connected;
- **an activity LED** at the front edge, labelled DISK *p*, lit through 470 Ω while the drive pulls `-DASP`.

Offset bit 3 picks `-CS0` (the task file, +0–7) or `-CS1` (the control block, +8–15); bits 2–0 are DA2–0 (the registers: disk-card.md §1).

## 3. The jumpers

Six jumpers at the card's edge, labelled. **Fitted reads 0, open reads 1**: each grounds its line, and a 10k network pulls it up.

| Jumpers | Meaning | Default (D52) |
|---|---|---|
| `WIN2 WIN1 WIN0` | the window, 0–7: the registers at `0xFFFF_8F00` + window × `0x1000`; the vector's high digit (§7) | 6: WIN0 fitted, the others open |
| `L2 L1 L0` | the interrupt level of all five sources | 1: L0 open, the others fitted |

## 4. Decode and the byte lanes

**Off the bus**, one load per line: two 74HCT541s buffer A2–A7, BE0–3, `-MRD`, `-MWR`, `-IACK`, CLK (it clocks the registered GALs and the counters) and `-RES` (it clears those GALs and is each port's `-RESET`); two 74HCT688s see A16–31 all ones; A8–A15 go to GAL DKSEL directly.

**GAL DKSEL**, *is this cycle for the card?* The registers: A31:A16 all ones, A15 = 1, A14:A12 = the window jumpers, A11:A8 = F; `CRD` with `-MRD`, `CWR` with `-MWR`, `CACC` with either. `INTA`: `-IACK` low, whatever the address (§7).

**GAL DKBE**, *how wide?* `BYTE` (exactly one byte enable) with its lane `LN1:LN0`; `HALF` (BE0 and BE1, no others); `WORD` (all four).

**GAL DKDEC**, *which register?* With `CACC`: `PSEL`, a port register (offset bits 7:6 = 0, and a byte at any of the 64 offsets, or a halfword at a port's +0: `WIDE`, the data register); `SIDR`, a byte at +F0 or +F1; `SISR`, a word at +F4. GAL DKDMA decodes the DMA registers (§8.1).

**GAL DKSTB**, *does the card answer, on which lanes?* `ACK` for `PSEL`, `SIDR`, `SISR`, the DMA registers (`DMAK`), or an acknowledge cycle the card answers (`INTA` and `VHIT`, §7); a 74HC05 open-drain gate puts it on `-ACK`, two more put the sequencer's `WAIT` and the engine's `EWAIT` on `-WAIT`. DKSTB enables **five 74HCT245s** between the lanes and CD, and sets their direction (`LDIR`: toward the card for a write):

| Access | Lane ↔ CD |
|---|---|
| the data register, a halfword | lane 0 ↔ CD0–7 and lane 1 ↔ CD8–15 |
| a byte register, the ID, the revision | its lane ↔ CD0–7 |
| the status word, the vector | lane 0 ↔ CD0–7 |

None is enabled while the DMA engine has CD (`EACC`, §8.4). A byte register sits on its own lane at its own address, as in `p32io.h`. Three 74HCT541s drive zeros onto D8–15 (`Z01N`: a read of the status word or DMACTL) and D16–31 (`Z02N`: those and DMACNT).

## 5. The port access: DKSEQ, DKPSEL, DKPORT0/1

ATA's slowest PIO mode (0) wants the address 70 ns before the strobe, a strobe of 165 ns, and write data held 30 ns after it (disk-card.md §4). The card gives every port access **three clocks: two wait states** (D66).

| S | Clock | What happens |
|---|---|---|
| 0 | 1: address | DA2–0 and `-CS0` or `-CS1` come out; `-WAIT` (a CPU access) |
| 1 | 2: strobe | `-DIOR` or `-DIOW` low for the whole clock; `-WAIT`; repeated while IORDY is low |

| S | Clock | What happens |
|---|---|---|
| 2 | 3: recovery | the strobe off; a write's data still held; a CPU read's data from the read latch, `-WAIT` released, so its cycle ends with this clock |

**GAL DKSEQ**, clocked by CLK and cleared by `-RES`, holds S: 0 → 1 → 2 → 0 while a port access is asked (`PACC`), 0 whenever none is. It makes `WAIT` (a CPU port access in S = 0 or 1, or any while the engine's runs), `-DIOR` / `-DIOW` and `STRB` in S = 1, `RLOEN` for a CPU read in S = 2.

**GAL DKPSEL**, *whose access?* `PACC` is the CPU's (`PSEL`) or the engine's (`EACC`). The CPU's: port BA5:4 (`PS1:PS0`), block BA3 (`PA3`), DA2 = BA2 (`PA2`), DA1:0 the lane (`PL1:PL0`), direction `CWR` (`PWR`). The engine's: DMACTL's port and direction, the rest 0 (`-CS0`, DA2–0 = 0: the data register).

**GALs DKPORT0** (ports 0, 1) and **DKPORT1** (ports 2, 3) drive the addressed port only: `-CS0` or `-CS1` for the whole access, `-DIOR` / `-DIOW` from DKSEQ, and its transceivers' enable: a write all three clocks (the data held through the recovery), a read only in S = 1.

**IORDY:** a 74HCT153 picks the addressed port's IORDY (`IORDYS`); DKSEQ looks at it as the strobe clock ends, and while it is low the strobe lasts another clock. **The read latch:** two 74HCT574s across CD0–15, clocked by DKSEQ's `-DIOR`, take the drive's data as the strobe ends, so the drive can let go, and drive it back onto CD in S = 2 (`RLOEN`). The drive's own `-DIOR` passes DKPORT and the 541 first, so it rises after that clock.

## 6. A register access, end to end

`lhu r2, 0xFFFFEF20(r0)`: port 2's data register (IDE channel A).

1. **Clock 1** (S = 0). The MMU / data card drives the address, BE0–1 and `-MRD` (mmu-card-theory §6). DKSEL gives `CRD`, DKBE `HALF`, DKDEC `PSEL` and `WIDE`. DKSTB pulls `-ACK` and turns lanes 0–1 toward the bus. DKPSEL and DKPORT1 put DA2–0 = 0 and `-CS0` on port 2's cable. DKSEQ pulls `-WAIT`; the MMU / data card raises `STALL`, and the CPU's commit waits.
2. **Clock 2** (S = 1). `-DIOR` low on port 2; its transceivers carry DD0–15 onto CD0–15. With IORDY low at the clock's end, it repeats.
3. **The edge.** S = 2: DKSEQ's `-DIOR` rises and the read latch takes CD0–15; the port's transceivers turn off.
4. **Clock 3** (S = 2). The latch drives CD0–15, lanes 0–1 carry it to D0–15 (DD0–7 on D0–7: the first byte in bits 7:0). `-WAIT` is released.
5. **The edge that ends clock 3** ends the cycle: the MMU / data card steers the halfword onto R, and the CPU commits. S returns to 0.

**A write** takes the same three clocks, the lane transceivers toward the card (`LDIR`) and the port's toward the drive in all three; the master holds D through the waits (bus.md), so the data stays a clock after `-DIOW` rises. **A byte register**: `lbu r2, 0xFFFFEF27(r0)` (port 2's status) has BE3 alone: lane 3 onto CD0–7, DA2–0 = 7 (A2 and the lane), `-CS0`. **The ID, the revision and the status word** take one clock: DKVEC drives CD0–7 (§7), unless the engine has CD (§8.4). At 3 MHz the strobe lasts 333 ns; a port access costs the CPU two clocks more than a memory access, and `p32uemu -C` adds the same two (D66).

## 7. Interrupts

Five sources: the ports' INTRQ0–3 and the DMA engine's `DMAIRQ` (§8.1). A drive's INTRQ goes away when its status register (+7) is read (programmer's guide §6.4), the engine's when DMACTL is written. One level for the card, set by L2–L0.

**GAL DKIRQ:** `ANYN`, low while any source asks; `VHIT`, an acknowledge cycle (`INTA`) for the card's level (`EQ`) with a source asking; `IDOE`, DKVEC onto CD0–7 for `VHIT` or a read of the ID, revision or status word, never while the engine has CD.

**The request: a 74LS145**, a BCD-to-decimal decoder with open-collector outputs: A–C are L0–L2, D is `ANYN`. While a source asks the code is the level, 0–7, and that output pulls its `-IRQ` line; with none asking it is 8–15, which pulls none (outputs 8 and 9 are not connected).

**The acknowledge cycle** (D57). The MMU / data card pulls `-IACK` with the level on A4:A2:

1. DKSEL raises `INTA`; a **74HCT85** compares BA4:BA2 with L2:L0: `EQ`.
2. DKIRQ raises `VHIT` if a source asks; DKSTB pulls `-ACK` and turns lane 0 toward the bus.
3. **GAL DKVEC** drives the vector onto CD0–7: **window × 16 + the lowest asking source** (ports 0–3, the engine 4): `0x60`–`0x64` in window 6.
4. The MMU / data card puts D7:0 into CAUSE bits 15:8 (mmu-card-theory §7): port 2 at level 1 gives CAUSE = `0x8000_6201`.

If no source asks by then, the card does not answer and the vector is `0xFF` (D57). **The status word** (+F4): DKVEC drives bit *p* = INTRQ*p* and bit 4 = `DMAIRQ` onto CD0–7, the 541s zeros above.

## 8. The DMA engine (D75)

Built in so that no new board is needed; P32W Unix does not use it yet. Per sector, a driver issues the ATA command and, at DRQ, writes DMAADR, DMACNT and DMACTL; the engine moves the sector while the CPU runs on, doing the drive's PIO itself through the CPU's port circuits (§4.1 of disk-card.md).

### 8.1 Its registers

| Register | Read | Write |
|---|---|---|
| `+40` DMAADR | the physical address of the next word | the first word's (bits 1:0 ignored) |
| `+44` DMACNT | the words left (16 bits) | the words to move, 0–65535 |
| `+48` DMACTL | bit 0 BUSY, 1 DIR, 3:2 the port, 4 IE, 5 MODE, 6 DONE, 7 ERR | bit 0 RUN, 1 DIR (0: the drive to memory), 3:2 the port, 4 IE, 5 MODE (reserved, 0); clears DONE and ERR |

**GAL DKDMA** decodes a word access to each (the load and read strobes, `DMAK` for `-ACK`). They sit on D, not CD, and never wait; a write takes effect at the edge that ends its cycle.

- **The address:** eight 74F163s, A31:2, counting words, loaded from D31:2; four 74HCT541s read them back, bits 1:0 zero.
- **The count:** four 74F163s hold its complement, loaded from D15:0 through two 74HCT540s and counted up once a word; their carry out of all 16 bits, `ALLONES`, means none left. Two 74HCT540s read them back inverted. (74F, so the carry chains settle within a clock.)

- **GAL DKDCTL**, clocked by CLK, is DMACTL: a write loads RUN, DIR, the port, IE and MODE from D5:0 and clears DONE and ERR. `BUSY` is RUN with words left. RUN clears itself when the count runs out (DONE sets, a clock after the last word) or a bus cycle has no `-ACK` (`BERR`: ERR sets). `DMAIRQ` is IE and (DONE or ERR). A 74HCT541 reads DMACTL back. `-RES` clears it; the counters keep their contents.

### 8.2 The word buffers

**In** (DIR 0): four 74HCT574s. The first halfword goes from CD0–15 into the low pair at `CKLO`, the second into the high pair at `CKHI`, each as the engine's `-DIOR` rises; they drive D0–31 in the engine's write cycle (`EWRN`). **Out** (DIR 1): four 74HCT574s take D0–31 as CLK rises at the end of the engine's read cycle (`CKRD`); the low pair drives CD0–15 in the first port write (`OLON`), the high pair in the second (`OHIN`). Either way the first halfword is bits 15:0 of the word in memory, as the CPU's own `lhu` / `sh` copy loop leaves it.

### 8.3 The sequence: GAL DKENG

Clocked by CLK, cleared by `-RES`. A word is three steps, `PH`:

| PH | DIR 0: the drive to memory | DIR 1: memory to the drive |
|---|---|---|
| 0 | read the data register: the low halfword | the bus read |
| 1 | read it again: the high halfword | write the data register: the low halfword |
| 2 | the bus write | write it again: the high halfword |

- **A port step** (`EACC`) starts only with DKSEQ idle (S = 0) and no CPU access asking for CD (`CDREQ`, §8.4). It is exactly a CPU access's three clocks through DKSEQ, DKPSEL, DKPORT and the port's transceivers; it ends with S = 2, and PH advances.
- **The bus step:** `REQ` pulls the card's `-BREQ`. With `-BGNT` low at an edge, `CYC` is the bus cycle: one clock, longer while `-WAIT` is low. At its end PH advances (to 1 after a read with `-ACK`, otherwise to 0).
- A word **starts** only while BUSY; one started is finished, so writing RUN = 0 stops the engine after the word in hand.
- `INC` advances both counters: in DIR 0 at the end of a bus write with `-ACK`, in DIR 1 at the end of the second port write.
- `BERR`: the bus cycle ended with no `-ACK`. The word is not counted: DMAADR holds the failing address, DMACNT the words left. In DIR 0 its two halfwords have already left the drive.

**GAL DKEOUT** makes `MSTN` (the engine's bus drivers, on in `CYC`), `ERDN` / `EWRN` (`-MRD` / `-MWR`; `EWRN` also opens the in buffer onto D), the buffers' clocks and enables (§8.2), `CDREQ` and `EWAIT` (§8.4).

### 8.4 Sharing CD with the CPU

The CPU may use the card while the engine runs. `CDREQ` is a CPU access that needs CD: a port register (`PSEL`), the ID, the revision, the status word, or an acknowledge cycle (`INTA`, of any level). While a port step runs, such an access waits: `EWAIT` pulls `-WAIT` (so does DKSEQ's `WAIT` for a port register), and DKSTB's lane enables and DKIRQ's `IDOE` stay off. When the step ends the access runs (a port register with its own three clocks); the engine starts its next step only with no `CDREQ`. The DMA registers are on D and never wait. A CPU port access that the engine's grant takes off the bus after its strobe is run again in full when the grant drops (§10).

### 8.5 As a bus master

- **The request:** `-BREQ` is the slot's radial request. The MMU / data card's arbiter grants at a CLK edge when no grant is out and `-LOCK` is high: slot 0 first, then the highest slot (mmu-card-theory §9).

- **The cycle:** in `CYC` five 74HCT541s drive A2–A31 from the address counter, BE0–3 all high (a word), and `-MRD` (DIR 1) or `-MWR` (DIR 0), the other high; `-IACK` is left alone. For a write the in buffer drives D0–31. The addressed card answers as it would the CPU, and `-ACK` is looked at as the cycle ends. The address is physical: no MMU (bus.md).
- **The release:** `-BREQ` goes high after `CYC`; the arbiter drops `-BGNT` at the next edge.

So `-BGNT` is low **three clocks a word**: the bus idle, the cycle, and one after `-BREQ` is released; a CPU bus cycle in any of them waits (`STALL`). The engine's bus cycles need 261.5 ns of the 333 ns clock at worst case, 144 ns typical (D75, timing.md).

### 8.6 A word, end to end

DIR 0, nothing else using the card or the bus (DKENG's equations; the gate-level simulation of `diskdma_test` shows the same):

| Clock | What |
|---|---|
| 1–3 | `EACC`, PH 0: DA2–0 = 0, `-CS0` on DMACTL's port; `-DIOR` in clock 2; `CKLO` takes the low halfword as it rises |
| 4 | idle; PH = 1 |
| 5–7 | the second read: the high halfword at `CKHI` |
| 8 | idle; PH = 2 |
| 9 | `REQ`: `-BREQ` low |
| 10 | `-BGNT` low; the bus idle |
| 11 | `CYC`: the address, BE0–3, `-MWR`, the in buffer on D; the memory card pulls `-ACK`; `INC` at the end |
| 12 | `-BREQ` high, `-BGNT` still low; PH = 0 |

**DIR 1** is the reverse: `REQ` in clock 1, the grant in 2, the bus read in 3 (`CKRD` at its end), the release in 4, the port writes in 5–7 (`OLON`) and 9–11 (`OHIN`; `INC` at the end), idle in 8 and 12. Either way a word takes **12 clocks**: a 128-word sector 1536 clocks, about 0.5 ms at 3 MHz. A CPU access to CD between two steps delays the next.

### 8.7 Multiword DMA: wired, reserved

Each port's DMARQ goes to GAL DKMWD, and its `-DMACK` comes from DKMWD through a 74HCT541. DKMWD gives the selected port's DMARQ (`DRQS`) to DKENG, which also has MODE; DKPSEL gives `DMAM` (MODE in an engine access) to DKPORT0/1, to negate the chip selects. As programmed, `-DMACK` is never asserted and no equation uses `DRQS`, MODE or `DMAM`: the drives' own DMA is a reprogramming of the socketed GALs DKENG, DKMWD and DKPORT0–1.

## 9. Verified

`python3 generators/card_disk.py` checks every output of the 15 GALs against its spec, exhaustively, and the netlist (every pin connected or declared unused, every input driven, at most one totem-pole driver a net, the connector pins as bus.md). `make test` runs it, then two programs at gate level with the CPU, MMU / data, memory and I/O cards (`generators/sim_sys.py --mem --io --disk`; only the drives' insides are `p32io.h`'s model). `disk_test` (the ID and revision, an empty port reading 00, IDE A's master and slave, CF 1 written by PIO and read back, the status word, port 2's interrupt through vector `0x62`) matches `p32uemu -C` in every instruction's state, the cycle count and the written image. `diskdma_test` (the DMA registers; port 2's sector read by DMA in two transfers while the CPU reads the ID, the status word, the empty port 3 and DMACTL; a sector written to CF 1 by DMA and read back; the interrupt through

`0x64`; a transfer stopped with ERR) checks itself, also passes on `p32emu` and `p32uemu -C`, and writes the same sector on all three. Not covered: the CPU using a live port while the engine runs (§10), the drive's data timing into the read latch (disk-card.md §6).

## 10. Open questions

- **A CPU port access can be cut in two.** The arbiter grants at any CLK edge with no grant out (mmu-card-theory §9), even inside a cycle held by `-WAIT`; bus.md says "at a cycle boundary". A CPU port access that waited behind the engine's last port step of a word strobes in the clock where `REQ` first shows; the grant at its end takes the cycle off the bus, and the MMU / data card runs it again: two strobes. `diskdma_test` at gate level does this to three reads of the empty port 3, unnoticed. A scratch program reading port 1's data register by PIO, with varying delays, while the engine read port 2's sector gave port 1 seven extra strobes and a wrong copy at gate level; `p32emu` and `p32uemu -C`, which move a word at once, pass it. A write would be strobed twice. How to prevent it is open.
- **Clocks a word.** D75 and disk-card.md §4.1 say about 11, the bus the engine's for one; `p32uemu` moves a word every 11 cycles (`p32io.h`). The equations and the gate-level runs give 12, `-BGNT` low for 3.
- **Every acknowledge waits for the engine:** `CDREQ` and `EWAIT` take `INTA` without the level, so another card's acknowledge also waits while a port step runs; disk-card.md §4.1 says "an acknowledge". Intended?
- **A bus error in DIR 0** comes after the word's halfwords left the drive; `p32io.h`'s `dma_word()` checks memory first and reads nothing (`diskdma_test`'s bus error is in DIR 1, where they agree).
- **Before layout:** CD0–15 has no pull resistors (it floats between accesses; in a byte write CD8–15 is undriven, copied onto DD8–15), and IORDY reaches DKSEQ's registers unsynchronised.
- **Stale text.** `card_disk.py`'s docstring: revision 0 (`CARD_REV` is 1), no status bit 4, no DMA registers. `p32io.h`'s header and the programmer's guide's §6 table: the status word without bit 4. disk-card.md: 9 GALs (§5; 15), "all four ports" on the level (§2; five sources), zeros "for the status word" (§3; DMACTL and DMACNT too), the DMACTL row running into the next paragraph (§1). The programmer's guide §6.4: port 0's CF card "in a socket" (an adapter on a header, D66).

# The video card, revision 2: an HD63484 ACRTC (D103)

docs/video-card-acrtc.md

**Status:** DESIGNED, SIMULATED AND LAID OUT; NOT ORDERED (2026-11-12). Its choices are decided (D104: §7), **the software is done** (D105: §5: both emulators model this card, and P32W Unix runs on it there), **the netlist is written** (D106: §4.7, `generators/card_video.py`: 58 ICs, 5 of them GALs; its kept part checked equal to YACC1-D's netlist), **it is simulated and timed** (D107: §4.8: with every part's data-sheet delays, at every phase of the chip's clock; and in a machine at gate level, cycle for cycle as the emulator) and **it has its schematic and its board** (D108: §4.9: six layers, routed, the pre-order check without a failure, and the timing checks run again with the board's own tracks as delays). What is left is Ken's: §8's checks by hand, and the order (§9). It replaces the 6845 description, revision 1 (D98), which never got past its description, and revision 0 (D67), which was laid out and never built: revision 0's netlist and GALs are in the repository's history only (last in commit `78473d7`), and its schematic and board (last in commit `5d447f7`) have made way for this card's in `hardware/video-card/kicad`. D103, D104, D105, D106, D107, D108, D109; D50, D52, D53, D57, D72, D98.

The card is YACC1-D's video card V3, which is not being built for the YACC1 (`~/Developer/YACC1-D/docs/cards/video-v3.md`; its section 13 sorts the design for another bus). **The memory side and the dot path are taken as they are**: they are simulated gate for gate and timed with the parts' data-sheet numbers. **The bus side is new**, for P32W's bus, and is what this note designs.

## 1. What the card is

- **Hitachi's HD63484P8** (the ACRTC, 8 MHz grade, 64-pin DIP; two are in hand) draws **VGA, 640 × 480 at 60 Hz, one bit a dot**, from a 25.175 MHz oscillator. Its clock, 2CLK, is the dot clock / 4: 6.294 MHz, 158.9 ns.
- **The chip draws, not the CPU.** The frame buffer (two 128K × 8 RAMs: 128K words, six screens) is on the chip's own memory bus. The CPU sends commands: line, rectangle, fill, copy, 38 in all, at about a microsecond a dot. **Nothing of the picture is on P32W's bus.**
- **Text** two ways, as the YACC1 note's section 8: **B**, glyphs copied in the bitmap by the chip, with no text hardware; **A**, a character screen of 80 × 30 laid over the graphics, in eight more chips.
- **Two PS/2 ports** through an ATmega328P with YACC1-D's firmware, as D98 decided.
- **The console** (D53), `/dev/tty4`, as before.

| | Revision 0 (D67) | Revision 1 (D98) | Revision 2 |
|---|---|---|---|
| The raster | counters and GALs | a 6845 | the ACRTC |
| Graphics | 640 × 480, the CPU writes bytes | 640 × 400, the CPU writes bytes, two waits | 640 × 480, the chip draws from commands |
| Text | cells in RAM on the bus | cells in dual-port RAM on the bus | drawn by the chip (B), or a character screen in the chip's memory (A) |
| On the bus | 64 KiB of RAM and a register page | the same | **a register page only** |
| ICs | 88, 18 GALs | about 53, 6 GALs | **58, 5 GALs** (§4.7); 50 with method A's sockets empty |

## 2. What is kept, and where it is

All in `~/Developer/YACC1-D` (branch `main` unless said):

| What | Where |
|---|---|
| The design: the memory cycle and every equation (10b), text method A (10c), the simulation and its margins (10d) | `docs/cards/video-v3.md` |
| The circuit: parts, pins, nets | `hardware/cards/video/kicad/v3.0/v30_netlist.py` |
| The simulation and the timing check: eight corners, set-up, hold, runt pulses, two outputs on a net | `tests/video/v30_netsim.py` |
| The clock and register figures for any mode | `tools/acrtc_timing.py` |
| The emulators' model of the chip; the font image | `software/acrtcmodel.h`, `tools/v3font.py` (branch `video-v2`) |
| A driver: the start sequence, a console by method B | `firmware/monitor/monitor.asm` (branch `video-v3`) |
| The PS/2 firmware | `embedded/video-ps2` |
| Hitachi's user's manual (1985) and application note (1986) | `../private-resources/YACC1-D/docs/datasheets/` |

**Kept, part for part and net for net: 35 ICs** (checked: §4.7). The chip (1); the 74F163A dot counter on the 25.175 MHz oscillator, the control logic, the delay line and the two registers that make the control register from the chip's attribute byte (16); the address latches, the two RAMs, the MAD buffers, the font EPROM and its copy (8); the two graphics shift registers (2); text method A's sockets (8). With them the DE-15 and its resistors. The control register's bits stay: three colour-off bits, the two syncs' polarity, the text layer's three.

**The rule that comes with it** (the YACC1 note's section 13): the margins belong to these parts wired this way. A changed part, a gate moved into a GAL or an equation rewritten is a new circuit to the timing check. So the kept part stays in 74F and 74ACT, outside P32W's GALs, and P32W's netlist is checked equal to `v30_netlist.py` for those parts (§9).

**The seam** between the kept part and the bus side is small: the chip's RES, DACK, CS, RS, R/W, D0–D15, IRQ and DTACK, and the clear of the two control registers.

**Not kept** (the YACC1's): the I/O ports PA–PF and their two decoders, the latch on RS and R/W, the 8-bit bus buffer, the card status port, the DIN 41612 connector and the 250 mm outline. The PS/2 section's own parts stay; their strobes and status are redesigned (§4.6).

## 3. The chip's host side, from the manual

From the user's manual's AC table and its figures T-2 and T-3 (the 8 MHz grade), read again for this note. T is one 2CLK cycle, 158.9 ns.

| What | The manual | So |
|---|---|---|
| RS and R/W before CS falls; after CS rises | 50 ns; 0 | |
| CS before the 2CLK rising edge that takes it | 40 ns | the chip samples CS with its own clock |
| CS high between accesses | 60 ns | |
| A read's data | 80 ns after the **2CLK falling edge** that follows that rising edge | at worst 1.5 T + 120 = **358 ns after CS falls** |
| DTACK low | 70 ns after the next 2CLK rising edge | at worst 2 T + 110 = **428 ns after CS falls** |

| What | The manual | So |
|---|---|---|
| CS rises | not before DTACK is low | |
| A read's data after CS rises | held 10 ns, off within 60 ns | |
| A write's data | 40 ns before CS rises, 10 ns after | the chip takes it as CS rises |
| RES | low for ten 2CLK cycles | DACK's level as RES rises: high = the 16-bit bus, low = the 8-bit bus |

Three things follow.

1. **The chip is an asynchronous peripheral.** CS is taken by 2CLK, which has nothing to do with P32W's `CLK`. The chip says when it is done with DTACK (open drain), "between 1 and 1.5 2CLK cycles" after it takes CS.
2. **Without DTACK, CS is held at least three 2CLK cycles: 477 ns.** That is the application note's rule for a synchronous bus (its section 3.1, the HD6809). It covers the 428 ns above with 49 ns in hand.
3. **The chip may stretch an access** in two cases the manual names: an access straight after the one before, when 2CLK is much slower than the processor's clock; and a read straight after a write. Neither happens on P32W: 2CLK is faster than `CLK`, and the CPU's instruction fetch always stands between two accesses to the card.

**A correction to the YACC1 note.** Its sections 5 and 13 give a read's data "80 ns after CS", and its simulation's stand-in chip answers so. The figure times the data from 2CLK: up to 358 ns after CS. On the YACC1 that changed no result at its 1 MHz clock (a 2 µs strobe); at the note's 4 MHz corner the margin of 300 ns it lists would be about 20. P32W's cycle (§4.2) is designed with 358 ns.

# 4. The bus side on P32W

## 4.1 Where the card is

A register page and nothing else: the top 256 bytes of device window n, `0xFFFF_8F00 + n × 4 KiB` (D52); n is 5 by default, `0xFFFF_DF00`. The card's 64 KiB at `0xFE00_0000 + n × 64 KiB` is **not used**: no `-ACK`, a bus error.

| Offset | Size | Read | Write |
|---|---|---|---|
| `+00` | halfword | **the chip's status**, in bits 7–0: bit 0 WFE (write FIFO empty), 1 WFR (it has room), 2 RFR (read FIFO has a word), 3 RFF (it is full), 4 light pen, 5 CED (command end), 6 ARD (area detect), 7 CER (command error) | **the chip's address register** (bits 7–0) |
| `+04` | halfword | the register addressed, or the read FIFO (address register 0) | the register addressed, or the write FIFO |
| `+08` | byte | **card status**: bit 0 FULL (a PS/2 byte waits), 1 SRC (0 keyboard, 1 mouse), 2 BUSY (a command not yet taken), 3 TEXT (method A's parts are fitted), 4 the chip's IRQ | **PS/2 command**, YACC1-D's |
| `+0C` | byte | **PS/2 data**; reading it clears FULL | — |
| `+F0` / `+F1` | byte | card ID 3 / revision 2 | — |
| `+F4` | word | interrupt status: bit 0 PS/2, bit 1 the chip (bits 31–2 read 0) | — |

The chip's RS is **A2** (the bus has no A1 or A0). Anything else in the page, or another size, gets no `-ACK`: a bus error; so does a write to `+0C`. The status's bits 15–8 are not defined by the manual: the emulators give ones there, so a driver that does not mask them fails (on the card they read zero if the chip leaves them alone: its data lines have pull-downs). The card's own registers answer with no wait, but one: **a PS/2 command written to `+08` takes two clocks** (one wait, D106: §4.6). A byte register answers on its own lane only: `+08`, `+0C` and `+F0` on D0–D7, `+F1` on D8–D15.

### 4.2 The cycle: four clocks, no handshake

Every access to the chip takes **four clocks, three of them waits**, made by a count in a GAL on `CLK`. Nothing comes back from the chip's clock to the bus's, as on revision 1's graphics access.

| | Clock 1 | Clock 2 | Clock 3 | Clock 4 |
|---|---|---|---|---|
| `-ACK` | low, from the decode | low | low | low |
| `-WAIT` | low | low | low | high |
| RS, R/W at the chip | move | **held** | held | held, and a clock more |
| CS | high | **low from mid-clock** | low | a read: low to the end. A write: **high from mid-clock** |
| The bus buffers | off | on from mid-clock | on | on to the end |

- **RS and R/W** are two registers in the GAL. They take A2 and the direction at the edge that ends clock 1 (when the decode is done, as for `-ACK`) and do not move again until a clock after the cycle. So they are steady half a clock before CS falls and a clock after it rises, whatever the master does at the cycle's end. This is the job of the YACC1 card's latch.
- **A write ends half a clock early**, as the I/O card's `-WR` does (D72): the chip takes the data as CS rises, and the master holds it half a clock more.
- **A read's data** is taken by the master at the edge that ends clock 4.

| | Wanted | At 3 MHz | At 4 MHz |
|---|---|---|---|
| CS low, a read (2.5 clocks) | 477 ns | 833 | 625 |
| CS low, a write (2 clocks) | 477 ns | 667 | **500** |
| RS, R/W before CS | 50 ns | 167 | 125 |
| A read's data at the chip before the cycle ends | — | 475 | 267 (for the buffer, the backplane and the MMU / data card's steering) |
| A write's data after CS rises | 10 ns | 167 | 125 |
| An access | | 1.33 µs | 1.0 µs |

The 500 ns is the tight one: 23 ns over the application note's rule and 72 over the chip's own figure (DTACK by 428 ns). Both of its edges come from the same clock edge through the same flip-flop, so its width is two clocks, give or take that part's rising delay against its falling one: 11.5 ns at the data sheet's two extremes, so 488 ns at the least (§4.8). Three clocks would not do at any clock (a write's CS would be one clock). At 5 MHz, which P32W does not promise, the count would be five: another fuse file, not another board.

**DTACK goes to a spare input of that GAL**, with a pull-up, and to a test point. The equations do not use it. If the bench shows the chip stretching an access, the wait can be made to follow it with a new fuse file.

**How it is built** (D106). GAL VWT, on `CLK`, counts the clocks and holds RS and R/W. **CS is not a GAL's decode: it is a flip-flop** (half a 74ACT74) clocked by the bus's `-CLK`, which rises in the middle of a clock, and cleared by RUN, a register of VWT that is high in clocks 2 to 4. It takes "not the middle of a write's clock 4" from VWT. So CS falls in the middle of clock 2; a write's rises in the middle of clock 4, out of the same flip-flop from the same clock edge; a read's rises when RUN clears the flip-flop as the cycle ends. Its twin, the other half, is set in the middle of clock 2 and cleared with it: the two data buffers' window. **Nothing that reaches CS can make a runt pulse**, which matters because the chip samples CS with its own clock: an address settling at the start of some other cycle moves the decode, and the decode moves nothing here until a `CLK` edge has taken it.

### 4.3 Sixteen bits

The chip has a 16-bit host bus, which the YACC1 could not use (decided: §7, 1). On P32W a halfword is lanes 0 and 1, and P32W is little-endian: **the chip's D0–D7 are the bus's D0–D7, D8–D15 the bus's D8–D15**, through two 74HCT245 (the chip's outputs sink 2.2 mA). A command word or a register is then one access, a `sth` or `lhu`, where the 8-bit bus takes two with the high byte first. In this mode the chip allows no byte transfers, its address register counts in twos, and WPTN's count is in words. DACK is pulled high for it.

**The chip's 16 data lines are also the card's own data bus** (HD0–HD15). GAL VRD puts the card status, the ID, the interrupt status and the vector on HD7–0, the PS/2 data latch its byte, and the command latch takes its byte from there; one output of GAL VSEL is the revision's bit. The two 74HCT245s are then the only load the card puts on D0–D15, and they open for the chip only inside its cycle. Two 74HCT541s put zeros on D16–D31 when the interrupt status is read, a word.

### 4.4 Reset and the pins not used

`-RES`, buffered, is the chip's RES, the clear of the two control registers and, through two open-drain gates of the card's 74HC05 (the YACC1 card's 7416 is not needed), the ATmega's reset. DONE is pulled up, DREQ open, LPSTB grounded, EXSYNC open (the chip is the master), as in the kept netlist. No DMA (§6).

### 4.5 Interrupts

Two devices, as D52 and D57 want them:

| Device | Status bit | Asks when | Level jumpers (default) | Vector |
|---|---|---|---|---|
| 0, PS/2 | `+F4` bit 0 | FULL, with the firmware's interrupts on | 4 | n × 16 + 0 |
| 1, the chip | `+F4` bit 1 | its IRQ is low: whatever its command control register enables (write FIFO empty or ready, read FIFO, command end, command error, area detect) | 4 | n × 16 + 1 |

A 74LS145 for each pulls its level; in the acknowledge cycle the card answers for the lowest of its devices asking on that level, as the I/O card does: a 74F138 gives the level acknowledged, GAL VINT matches it against each device's jumpers, GAL VRD puts the vector on D7–0. The chip makes no vector of its own. Its IRQ is open drain: a pull-up, then straight to its 74LS145.

**Why the chip's IRQ is worth its jumpers:** the chip is busy about 30 ms on a scroll and its write FIFO holds eight words. A kernel that polls would spin for those 30 ms with every process waiting. With "write FIFO empty" as an interrupt the console is a transmitter like a UART's: fill the FIFO, return, go on from the interrupt.

### 4.6 PS/2

D98's choice stands: the ATmega328P with YACC1-D's firmware, the keyboard as ASCII and VT100 sequences, one queue for both ports. The V3 card's section moves onto P32W's registers: the ATmega loads a byte into a latch and sets FULL; a read of `+0C` puts it on D0–D7 and clears FULL; a write of `+08` latches a command (at the cycle's end, D72) and sets BUSY. The ATmega never sits on the bus. Its pins and its firmware do not change.

**The two strobes are registers of GAL VWT, never a decode** (D106). A strobe made from the decode in the cycle itself could come from an address that was only settling, and would lose a byte or latch a false command.

- **FULL's clear** is a pulse a clock long, in the clock after a read of `+0C`. The firmware already waits 5 µs after it sees FULL clear before it loads the next byte, and checks that the load took; the clear here lasts a third of a microsecond.

- **A command written to `+08` takes two clocks**, one of them a wait. The command latch takes the byte, and BUSY is set, as the second clock begins: the master has held the byte for a clock by then and holds it a clock more. (Taking it after a one-clock cycle would have left the latch's hold time to the gates' shortest delays.) `p32uemu -C` adds that wait.

**The firmware's time (D109).** The ATmega runs a loop, and the card's two flip-flops show how long it takes. Both emulators and the gate-level machine's stand-in model it (`emulator/p32io.h`, "the firmware's time"):

| what software sees | how long | why |
|---|---|---|
| after a reset, nothing | **about 2.1 s** | the ATmega's start-up and its bootloader's second (Optiboot), then the sketch's `start_all()`: a reset and self-test of the keyboard and of the mouse, half a second each |
| a command written before the sketch runs | **dropped** | BUSY's flip-flop has no reset, so `setup()` reads the command latch and throws the byte away |
| FULL clear after a byte is read | **about 50 µs** | the loop comes round, waits 5 µs, sets SRC and the byte and pulses LDDATA |
| BUSY after a command is written | **about 30 µs**, or until the command in hand is done | the loop comes round and reads the latch; it does one thing at a time |
| a command to a device | **3 ms** each: the mouse off one, the keyboard's LEDs two; a reset about half a second | the PS/2 frame both ways at the device's own clock, the device's self-test |

Estimates, all of them: the sketch's own delays and the Arduino core's (a pin read or written 4 to 5 µs), rounded up, for a part that has run on no hardware yet. **Software must depend on FULL and BUSY, never on the figures.** What follows from the firmware, and holds whatever the figures:

- **Read `+0C` only when FULL is set**, one byte at a time. The next byte is not there yet when the first has been read: an interrupt handler gets one byte, and FULL asks again for the next.
- **Write `+08` only when BUSY is clear.** A command written over one the firmware has not taken replaces it: the first is lost.
- **After a reset, the first command may be dropped**: send one that does not matter first (02, interrupts off, as a reset leaves them). When BUSY has cleared after it the sketch is past that point.
- **Never wait for the ATmega with the machine stopped**: BUSY can stay set for two seconds after a reset, and for ever with the socket empty. The kernel writes its three commands from its clock tick, one each time BUSY is clear.
- **03 (acknowledge) silences more than it says.** No interrupt comes until a device sends something new; bytes the firmware already holds (the rest of an arrow key's `ESC [ A`) then arrive in the latch with FULL and no interrupt. The kernel does not use it: reading the byte releases the line anyway.
- **06 (flush) empties the firmware's queue, not the card's latch**: a byte already there stays, with FULL, until it is read.

## 4.7 The netlist (D106)

`generators/card_video.py`: **58 ICs, 5 of them GALs**; 351 nets, 173 parts. Text method A's eight sockets are among the 58: 50 ICs on the first build.

| Block | Parts | ICs |
|---|---|---|
| Buffers | 2 × 74HCT541: A2–A7, BE0–3, `-MRD`, `-MWR`, `-IACK`, `CLK`, `-CLK`, `-RES`, one load a line (A8–A15 go to one GAL only, A16–A31 to the comparators) | 2 |
| Decode | 2 × 74HCT688 (A31–A16 all ones, as the I/O card); GAL VSEL (the page, read or write; the revision's bit); GAL VDEC (which register and its size, `-ACK`, who is on HD, the lanes) | 4 |

| Block | Parts | ICs |
|---|---|---|
| The cycle | GAL VWT (the count, RUN, `-WAIT`, RS, R/W, the PS/2 strobes); a 74ACT74 on `-CLK` (CS; the buffers' window) | 2 |
| Data | 2 × 74HCT245 (D0–D15 to HD0–HD15); 2 × 74HCT541, zeros on D16–D31 for `+F4`; GAL VRD (status, ID, interrupt status, the vector) | 5 |
| Interrupts | 2 × 74LS145; a 74F138 (the level acknowledged); GAL VINT (who asks, the vector's device) | 4 |
| `-ACK`, `-WAIT`, the ATmega's reset | 74HC05, open drain | 1 |
| PS/2 | ATmega328P, 2 × 74HCT574 (data, command), a 74HCT74 (FULL, BUSY) | 4 |
| **New** | | **22** |
| Kept (§2) | the 35, and the oscillator | 36 |
| **The card** | | **58** |

**The kept part is checked equal to YACC1-D's netlist.** A copy of `v30_netlist.py` is in `hardware/video-card/yacc1-v3/` (so the check runs wherever P32W is checked out). The generator transcribes the kept part line for line, by that file's own pin numbers and under its net names (its A0–A16 are VA0–VA16 here: A2–A31 are the bus's). `kept_check()`, part of `make test`, then proves:

- every part of YACC1-D's netlist is kept, or named as not kept (its bus side, its PS/2 section, its power parts);
- **50 parts, 765 pins, 178 nets**: a kept part is the same part, and each of its pins is on the same net. Two kept pins share a net here if and only if they do there; a pin left open there is open here;
- **11 pins differ, each listed with its reason**: DACK (pulled high, not tied to RES: the 16-bit bus), D8–D15 (connected), DTACK (a pull-up, a test point, GAL VWT's spare input), the oscillator's pin 1 (tied high, as on P32W's other cards);
- **nothing of the bus side is on a kept net** except at the seam: RS, R/W, CS, D0–D15, RES, IRQ, DACK, DTACK. So the bus side adds no load to anything the timing check counted.

Six planted faults (a pin on a neighbour's net, a pin opened, a load added to the memories' output enable, ...) were each caught.

**The bus side is run at gate level, as wired** (`generators/sim_video.py`, part of `make test`): the card's netlist alone in the project's simulator, its backplane pins driven half a clock at a time, with stand-ins at the chip's and the ATmega's pins. It checks §4.2's table (in which half of which clock CS, RS, R/W, the buffers and `-WAIT` move, for a read and a write), every register of §4.1 with its size, 12 bus errors, the PS/2 handshake, both interrupts on their levels and the vector, the jumpers. Twenty-two planted wiring faults were each caught. It is at zero delay: the chip's timing against its own clock is §4.8's.

**The five GALs** are fitted and each equation is checked against its specification over every combination of its inputs; their fuse files are in `hardware/gal/video`. The fuse maps are not yet compared with GALasm's (§8).

**Jumpers**, at an edge: `WIN2`–`WIN0`; the PS/2 section's level and the chip's (three each); TEXT (fitted with method A's parts); the RAMs' 128K / 512K jumper of the kept netlist. **The front edge:** the DE-15, the two mini-DIN-6 (KEYBOARD, MOUSE; +5 V through a polyfuse), the POWER LED. The ATmega's FTDI header and the DTACK test point at an edge. 9U × 340 mm, six layers, through-hole (D21, D48, D77): the YACC1 card's trouble with a 218 mm outline does not arise.

**The EPROMs:** the font (a 27C256) is `video/font-acrtc.bin`, which `python3 generators/gen_font.py --acrtc` writes (ROM images are built, not kept in git); the emulators hold the same bytes (`emulator/p32vfont.h`). Method A's character generator has no image yet: it comes with method A's driver.

### 4.8 The simulation and the timing (D107)

Four checks, all in `make test`.

**1. The memory side and the dots: YACC1-D's own check, on this netlist.** `generators/sim_video_timing.py` loads YACC1-D's `v30_netsim.py` (a copy is beside the netlist's, in `hardware/video-card/yacc1-v3/`, both byte for byte) and runs its two scripts, with its models of the parts and of the chip's memory side and its checks, unchanged, on the kept part of P32W's netlist: 20 runs at its eight corners, all pass. Three faults planted in the kept part's wiring each failed all 20.

**2. The bus side with delays, against the chip's host timing.** The same engine, with the bus side's parts added at their data-sheet delays, and at the chip's pins a model of §3: CS taken by the first rising 2CLK edge it has been low 40 ns before, a read's data 80 ns after the next falling edge, DTACK 70 ns after the rising edge after that. A write and a read of the chip are run **at every 4 ns of the chip clock's phase** against the bus clock, at 3 MHz and at 4, at six sets of corners (the chip and the parts slow or fast, and rising edges against falling ones): 480 runs; then the card's own registers and the PS/2 section, 36 runs. All pass. The least to spare, all at 4 MHz:

| What the chip or the machine asks | Asked | To spare |
|---|---|---|
| CS does not rise before DTACK | | **62 ns** (CS low 488 ns, DTACK 427 ns after its fall) |
| CS low three 2CLK cycles (the application note's rule) | 477 ns | 12 ns (a write, the flip-flop's rising edge fast and its falling one slow) |
| RS and R/W steady before CS falls | 50 ns | 70 ns |
| A write's data steady before CS rises | 40 ns | 392 ns |
| CS high between accesses | 60 ns | over a microsecond |
| A read's word on the backplane's D lines before the cycle ends | 140 ns (below) | **56 ns** (196 ns) |
| The command latch's data before its clock | 25 ns | 73 ns |

Between the chip's two accesses go another master's cycles, one whose address is the chip's for 40 ns on its way to another window's; likewise past `+0C` and `+08` with a byte waiting. Nothing answers, CS does not move, FULL stays, no command is latched. The engine also checks every flip-flop's and register's set-up and hold on the way, any pulse under 5 ns on a clock, a clear or CS, and two outputs on a net for more than 3 ns: none. Planted faults that bring a decode back onto a strobe (FULL's clear, the command latch's clock) fail it.

**3. The static timing pass** (docs/timing.md) has the video card in the machine's runs for its bus side: nothing fails at 3 MHz worst case, and the machine's fastest clock is what it was. `-WAIT` reaches the CPU card 104 ns before it is needed, the cycle's GAL has its inputs 202 ns early, the CS flip-flop its data 147 ns early. The pass now also says how early the machine needs a slave's read data on the D lines: **140 ns** before a cycle's last edge at worst case, 71 typical. The chip's word is checked against that figure in 2.

**4. The card in a machine at gate level** (`generators/sim_sys.py --video`): the CPU, MMU / data, memory, I/O and video cards as wired, with `p32io.h`'s models at the pins of the chip and of the ATmega328P. A short program, `emulator/test/vidcard_test.asm`, runs there and on `p32uemu -C` with the same trace, instruction for instruction, and the same cycles, at 3 MHz and at 4: the chip's cycle, its registers and FIFO, the PS/2 latches, **the interrupts taken, each with its vector from the acknowledge cycle**, bus errors. The whole of `video_test` was run the same way once, by hand (ten minutes a run): identical, 70,585 cycles at 3 MHz and 92,055 at 4 (before D109 gave the PS/2 firmware its time; not run again since).

Since D109 the ATmega's stand-in there follows the firmware's model clock by clock (§4.6): the model says in which clock the firmware loads a byte or reads the command latch, the stand-in moves LDDATA, -RDCMD and MD0-7 for it, and **after every clock the card's own FULL and BUSY flip-flops must equal the model's, or the run stops**. `vidcard_test` has the gap after a read, a command written over another at the card's latch, and a second PS/2 interrupt that comes when the firmware loads the next byte by itself: 1,555 cycles at 3 MHz and 1,731 at 4, identical; and identical again with one to four instructions put in before the reads and the interrupts (by hand, once).

**What the machine-level run found.** The emulator saw the chip's interrupt an instruction too soon. The CPU card samples the `-IRQ` lines in the middle of a bus clock; the chip takes a register write at that same moment of its cycle's last clock, as CS rises. So a request that a write causes (the kernel turning "write FIFO empty" on) is first seen in the next microcycle, and taken after the next instruction. `p32uemu -C` now has it so; nothing else changed for it.

**What these do not cover:**

- **The chip's real delays where the manual gives none** (its minimums; how long after a write its IRQ moves). The bench decides.
- **The PS/2 firmware's real figures.** Its time is modelled since D109 (§4.6), from the sketch and the Arduino core's own delays: estimates. The bench has the last word, and software may depend on none of them.
- **The dot side in the machine-level run:** the chip's memory side stands still there (no dot clock); check 1 is what runs it.

## 4.9 The schematic, the board and the review (D108)

**The schematic** is drawn from the netlist (`generators/p32kicad_sch.py`): eleven sheets, `hardware/video-card/kicad/video-card.kicad_sch` and its PDF. KiCad's rule check finds nothing, and the netlist KiCad reads back from the drawing is the generator's: 351 nets, the same pins on each.

**The board** (`video-card.kicad_pcb`; placed by `p32kicad_pcb.py` from the rows in `card_video.py`, routed by Freerouting): 340 × 366.7 mm, six layers (signals on the two outer and two inner ones, a ground plane and a supply plane), every part through-hole. 730 connections, all routed, in under five minutes; 59.5 m of track. Down the card:

| where | what |
|---|---|
| the top two rows | the bus side: comparators, buffers, the five GALs' first four, the CS flip-flop |
| the middle, towards the front | **the kept part**, six rows close together (below) |
| under it | the five resistors of the video output, beside the DE-15 |
| lower | the PS/2 section: the ATmega328P, its crystal, the three latches |
| the bottom row | the interrupt logic; along the bottom edge the ten jumpers (window, two levels, TEXT), each named |
| the front edge | the power LED and the DTACK test point at the top; the RAM-size jumper (128K / 512K at its ends); the DE-15 ("VGA"); the two mini-DINs ("KEYBOARD", "MOUSE"); the FTDI header for the ATmega (GND and DTR at its ends) |

Every part has its reference and its type or value on the silkscreen, outside its outline. **Five chips are turned end for end**, pin 1 at the right where every other chip's is at the left: the ACRTC, the two address latches (74ACT573) and two of text method A's latches (74ACT574). Each says `PIN 1 >` on its line of the silkscreen. Why they are turned is the next part.

**The tracks as delays.** The kept part's margins are small: YACC1-D's own check (§4.8, 1) leaves 0.2 ns at the least, and a track costs about 0.07 ns a centimetre. So the layout was reviewed with the tracks in the timing:

- `generators/board_tracks.py` measures, on the routed board, the track from every pin that can drive a net to every other pin of it, and how much of it runs on an inner layer (`video-card-tracks.json`, with the board file's SHA-256: `make test` fails if the file is not the committed board's; `build_board.sh` writes it after routing).
- `sim_video_timing.py` then runs its checks again with each pin seeing what another pin drives that much later: 5.9 ps a mm on an outer layer, 7.0 between the planes. A driver whose net is over the 50 pF the data sheets' delays are given at (the track at 0.12 pF a mm, 5 pF a pin) is slower by 0.03 ns a pF. **These figures are estimates**; the point of the check is that no margin hangs on them. Two outputs on a net at once are judged as before, by when each is on. With every track of length nothing the runs give §4.8's figures exactly.

**What it found.** The first routing, of a placement made row by row from the netlist's groups, **failed one run in twenty**: with text method A (the eight sockets the first build leaves empty), chip slow, the left character's latch got its data 0.7 ns too late. The first build's own runs passed; over all the runs the dot flip-flop was down from 7.2 ns to 0.7. The router was not at fault (its tracks were within a tenth of the shortest trees that join the pads): the parts were in the wrong places, and the chip faced the wrong way, its memory side away from the memories.

Taking each family of nets' tracks away in turn (`sim_video_timing.py --cost`) shows what the thin margins hang on. It is not the address path (the chip's MAD lines, the latches, the RAMs' address lines), whose tracks cost none of them anything: it has time in hand.

| margin | what its tracks cost |
|---|---|
| **MAD's hand-over** (the buffers off before the chip is on; 0.2 ns) | the way from the oscillator through the buffers' flip-flop and gate to the two 74ACT245 (`CK`, `BUFQN`, `BUFEN#`), against the way to the chip (`CK` to the counter, `2CLK`): the first must not be the longer |
| **text method A's latches' set-up** (2.9 ns) | `2CLK`; the RAMs' chip enable (`CHR` to the gate that inverts it, `CHRN`, `RAMCE#`); the RAMs' data lines to the latches; less the dot clock's way to the latches' own clock |
| **one memory off before the next is on** (2.0 ns) | the output enable `MOE#` to the three memories, which must reach them alike; `AS#` |
| **the dot flip-flop's set-up** (7.2 ns) | the dot's way through its gates |

So the placement is made for those nets (the comment in `card_video.py` lists it): the chip turned, its memory side down; the counter under its `2CLK` pin; the oscillator between the counter and the buffers' flip-flop; that flip-flop's gate straight below it, between the RAMs and the buffers; a RAM, and under it its text latch; the gate of `MOE#` between the font and the RAMs. A placement can be tried without routing it: `board_tracks.py --estimate` takes each net as the shortest tree of its pads, and came within 0.3 ns of the routed board on the three thinnest margins (within 1 ns on the others).

**The board as routed** (all 20 memory-side runs pass, and the bus side's at every second phase of the chip's clock, 276 runs):

| the least to spare, ns | no tracks | this board's tracks |
|---|---|---|
| two outputs on MAD at a hand-over | 0.2 | 0.5 |
| one memory off before the next is on | 2.0 | 1.9 |
| text method A's latches: data before their clock | 2.9 | 1.3 |
| the dot flip-flops' D (74F74) | 7.2 | 4.9 |
| the counter's -LOAD | 7.7 | 6.3 |
| RAM write data | 14.4 | 12.0 |
| the chip's read data | 19.2 | 18.2 |
| bus side: CS low over three 2CLK cycles | 12 | 12 |

| the least to spare, ns | no tracks | this board's tracks |
|---|---|---|
| bus side: a read's word on D before the cycle ends (140 needed) | 196 | 190 |

The first figure went up, not down: the chip's clock now takes a little longer to arrive than the order to the buffers does. It is still only half a nanosecond, and with the dot clock's way to three parts deciding it, **a board routed again must be measured again**; `make test` does that.

**The pre-order check** (`pcb-check --layers 6`, D100's master check): no failure. Two warnings: the files are not yet committed (they are after this commit), and two footprints are not in its table, so not checked by it: the 27C256's (an ordinary 28-pin, 0.6-inch DIP) and **the HD63484P8's**, drawn as a 64-pin DIP with 0.9 inches (22.86 mm) between its rows, as Hitachi's DP-64 outline is: to be held against a chip before ordering (§8).

**What the review does not cover:** reflections and crosstalk (a track is a delay here, nothing else); the chip's outputs' own dependence on their load (its sheet's test load is not 50 pF); the PS/2 section's lines (slow); and everything §4.8 lists.

## 5. The software (done 2026-11-12, D105)

- **The chip's model**, `emulator/p32acrtc.h`: YACC1-D's `acrtcmodel.h` on the 16-bit bus. Registers as words, the address register in steps of 2, the two FIFOs of eight words, the status register, IRQ from the command control register's enables, the commands a driver needs (it names the ones it lacks and sets the command error bit), the chip's busy time, a word lost when the FIFO is full. `emulator/test/acrtc_test.c` checks it by itself, driven as the console drives it: 55 checks.
- **The card in both emulators** (`p32io.h`): the chip at `+00` and `+04`; the PS/2 section as the ATmega328P's firmware shows it (its key translation, its queue of 63 bytes, its nine commands); the two interrupts and their vectors; no memory on the bus. `p32uemu -C` adds the three waits. The chip's time is the machine's, and a `WAIT` ends when the chip finishes its work and interrupts. `-V FILE` writes the picture (`FILE.pbm`) and the console's text, read back from the dots: each cell matched against the font, since no character codes are kept anywhere.
- **The kernel's** `video.c`: the start sequence, then the console by method B. A character is one AGCPY (five words) from the font's picture under the screen; a run of characters needs one move. Seven more pictures (bold for bright, underline, inverse and their mixes) are made by the chip at start, about 0.2 s. Clearing is CLR; a scroll is one CPY and one CLR; the cursor is lines 14 and 15 of its cell in XOR. The command words wait in a ring and go to the chip eight at a time from its "write FIFO empty" interrupt; kernel messages wait for the chip, as they must work in a panic. The keyboard's bytes are ASCII: the kernel's scan-code table is gone. The mouse is turned off.
- **The chip takes** about 290 µs a character by the model (918 2CLK cycles, doubled while the display shares the memory: an estimate) and about 30 ms a scroll.
- **The font**: P32W's own (`generators/gen_font.py --acrtc`), written as the EPROM's picture in YACC1-D's layout, bit 0 the left-hand dot.
- **The debugger's Screen** draws the chip's picture, and its `screen` command prints the console's text.
- **Tests** (`make test`): the chip's model; `video_test` (12 cases, both emulators identical, and with the card's timing at 3 and 4 MHz); P32W Unix on the video console (`session23`, as recorded before the change); the debugger. Revision 0's two gate-level runs (`vidcard_test`, `ps2_test`) went with its model, and its netlist with step 3 (D106); `sim_sys.py --video` runs this card since D107 (§4.8).
- **Still to do in software**, when wanted: graphics for programs (`/dev/video` taking the chip's commands, a C library, BASIC's statements: the backlog); circles, arcs and PAINT in the model; text method A (character screens, the cursors) in the model and a driver.

## 6. What this card does not do

- **The CPU cannot read or write the picture directly.** Every dot goes through the chip (its RD, WT and MOD commands move single words).
- **No DMA.** The chip's DREQ, DACK and DONE could feed a bus master like the disk card's (D75); not designed here or in the YACC1 note.
- **One bit a dot.** Eight colours for the whole picture (the attribute byte), not for a dot. More bits a dot, 800 × 600 and 1024 × 768 are the YACC1 note's section 9: a wider memory, a new circuit to the timing check.
- **Method A has no attribute for a character**: a cell is a byte.
- **The emulators' model stops short of the chip**: no circles, ellipses, arcs, PAINT, PTN; no character screens (method A), cursors or interrupts. The interrupts are needed for §4.5 and are small; the rest as wanted.
- **Never on a bench.** The kept part is proven by simulation only. The YACC1 note's sections 10d and 12 list what it leaves for the bench: the chip's minimum delays behind the two smallest margins, the pattern RAM's walk, which cycle the cursor's skew counts from.

## 6a. The picture on the bus as well (asked 2026-11-12; decided: left off, D106)

Ken asked what it would take. It can be done, as a second port that steals the chip's idle memory cycles, but it reopens the kept memory side. The recommendation given: leave it off. **Ken: "leave it off"** (D106): the card has no such port, and the netlist was written without it. What it would have taken stays here for the record.

- **Why it is not a decode.** The chip owns the frame buffer's address and data lines in every 318 ns cycle and has no hold pin there. The only free cycles are its dummy ones, a drawing slot with nothing to draw: every other cycle while it is idle, none while it draws.
- **What it needs**, about 12 more ICs, 2 of them GALs: three drivers for the bus's address onto the RAMs, with the two address latches switched off for that cycle; a 16-bit read latch and write drivers; two more lane buffers; control on the dot clock (the request across from the bus clock, a cycle with no address strobe by dot 3, the RAM run in dots 3–7, "done" back across to release `-WAIT`); the 64 KiB window's decode.
- **What it costs.** The latches' enables, the RAMs' second chip enable, the output enable and the write strobe each gain a term, and the address and data lines gain loads: the timing check runs again with a new kind of cycle. The two-clock handshake that D98 dropped comes back; an access is four to six clocks as the phases fall. While the chip draws there is no free cycle, so software touches the picture only at command end, or the card gives up with a bus error.
- **What it buys** (estimates): about twice the speed for words written in bulk and for characters, since P32W's CPU is the limit either way; and the picture mapped into a process, which nothing else gives.
- **Without it:** the chip's own word read, write and modify commands; text method A for fast text; later the chip's DMA transfers with a bus-master block like the disk card's.
- **Had it been wanted**, the time was before the netlist, as a socketed block like method A's, with the timing check run fitted and unfitted.

## 7. Decided (D104, 2026-11-12; Ken: "agree with all your recommendations")

| | Decided | The other way, not taken |
|---|---|---|
| 1. The chip's host bus | **16 bits** (§4.3): a command word or a register in one access, plain `sth` and `lhu`, one more 74HCT245. The model's FIFO and register access change as it moves to P32W | 8 bits, the model as it is, high byte then low byte |

| | Decided | The other way, not taken |
|---|---|---|
| 2. The cycle | **A fixed four clocks** (§4.2). DTACK goes to a spare GAL input and a test point, and the equations do not use it. Nothing crosses between the two clocks, and the emulator stays cycle-exact | `-WAIT` held until DTACK has passed two flip-flops: four or five clocks as the phases fall |
| 3. The chip's interrupt | **Wired**, as device 1 with its own level jumpers and vector (§4.5), so the kernel does not spin through a scroll | polled, as the YACC1 ROM does |
| 4. The console's text | **Method B first; the board has method A's eight sockets and tracks.** A's parts go in when a driver uses them (TEXT says they are there) | method A from the start; or B alone with no sockets |
| 5. Rows | **25**, as D50, in the middle of the 480 lines (lines 40–439) | 30 |
| 6. The two ROMs | **27C256, as timed** (the font; the character generator an AT27C256R-70) | the SST39SF010A, with the timing check run again |

Carried over without asking again: PS/2 on the ATmega328P with YACC1-D's firmware (D98); VGA only, on a DE-15; window 5, level 4, card ID 3.

Decided after (D106, 2026-11-12): **the picture is not on the bus** (§6a; Ken: "leave it off").

## 8. To check

- **Parts in DIP, in stock:** AS6C1008-55PCN, AT27C256R-70, 74F163A, 74F299, 74F74, 74F21, the 74ACT types of the kept netlist's bill of materials. A substitute is a new circuit to the timing check.
- **The chip's supply current**: the manual's table says "T.B.D."; measure one before the power budget.
- **The chip on the bench before the board is ordered**, if a breadboard can do it: that it answers on a 16-bit bus as the manual says, and its DTACK against §3's 428 ns.
- **The DE-15's footprint** against the part in hand (the YACC1 note's caution; revision 0's DE-15 was put right in D90).
- **The HD63484P8 against its footprint** (§4.9): 64 pins, 0.1 inch apart, the two rows 0.9 inches (22.86 mm) apart. `pcb-check` has no entry for it. And the mini-DINs' against the parts in hand.
- **Five chips are turned end for end** (§4.9: U15, U28, U29, U46, U47), marked `PIN 1 >`: to remember when the sockets go in.
- **A board routed again is measured again** (§4.9): the tracks' file is of one board file; `build_board.sh` rewrites it and `make test` runs it. If a run then fails, `sim_video_timing.py --cost` says which nets to look at.
- **The five GALs' fuse files against GALasm.** The other cards' were each assembled by GALasm too and found identical (D67); GALasm is not on this Mac. They are checked through the fuse map against their specifications, as all are.
- **The supply currents marked "estimate"** in docs/power-budget.md for this card's new parts (the RAMs, the EPROMs, the 74F299, 74F74 and 74F21, the ATmega), against their data sheets.
- **The data buffers going off as a read ends** (RUN, the flip-flop's clear, GAL VDEC, the 74HCT245): about as late as the I/O card's buffers, up to some 115 ns into the next cycle at worst case. The timing checks look at set-up, not at this; the other device cards are the same.
- **The PS/2 firmware's time on the real card** (§4.6): how long FULL is clear after a read, BUSY after a command, the two seconds from a reset. The tests wait for the flags, so different figures do not break them.

## 9. The order of work

1. **This note: Ken's choices** (§7). Done, D104.

2. **The software in the emulators:** the chip's model, the three waits in `p32uemu -C`, the debugger's Screen, the kernel's driver, the tests. Done, D105 (§5): P32W Unix runs on the new card in both emulators.
3. **The netlist** (`generators/card_video.py`): the 35 kept ICs transcribed from `v30_netlist.py` and **checked equal to it, part for part, pin for pin, net for net**; the bus side new, and run at gate level as wired. Done, D106 (§4.7). Revision 0's netlist, its 18 GALs and its place in the timing pass went with it.
4. **The simulation and the timing:** `v30_netsim.py`'s memory-side scripts run on P32W's netlist at their eight corners, in `make test`. The bus side against a model of the chip's host side from figures T-2 and T-3, at every phase of 2CLK against `CLK`, checking each line of §3's table. Then the card at gate level with the CPU, MMU / data and memory cards against `p32uemu -C` (`sim_sys.py --video`), and the bus side in the static timing pass (docs/timing.md). Done, D107 (§4.8).
5. **The schematic, the layout** (six layers, the layout rules), `pcb-check`, a design review. Done, D108 (§4.9): the review ran the timing checks with the board's tracks as delays and the kept part was placed again for what they showed. Revision 0's schematic and board in `hardware/video-card/kicad` are replaced.
6. **Ken orders it**, after §8's checks by hand and the pre-order check on the committed files (`~/Developer/pcb-check/pcb-check hardware/video-card/kicad/video-card.kicad_pcb hardware/video-card/kicad/video-card-gerbers.zip --layers 6 --tests`, then `--archive --fab F --order NUMBER`).

# The video card

docs/video-card.md

**Superseded (D98, 2026-11-10; D103, 2026-11-12):** this card, revision 0, is being replaced by a card on Hitachi's HD63484 ACRTC, from YACC1-D's V3 design: revision 2. (Revision 1, a 6845 card, was a description only.) Revision 0 was never ordered or built. **The emulators and the kernel no longer model it** (D105, 2026-11-12): they model revision 2, and this card's two gate-level runs against them (§6: `vidcard_test`, `ps2_test`) are gone. **Its netlist is gone too** (D106, 2026-11-12): `generators/card_video.py` is now revision 2's. Revision 0's generator, its 18 GALs' fuse files, its gate-level helper (`sim_video.py`) and its place in the timing pass are in the repository's history, last in commit `78473d7`. **Its schematic and board are gone as well** (D108, 2026-11-12): `hardware/video-card/kicad` now holds revision 2's; revision 0's are in the history, last in commit `5d447f7`. This page describes the card as it was.

**Status:** NETLIST DONE (2026-10-26), verified at gate level with the CPU, MMU / data, memory and I/O cards, its picture included (§6); layout to do. Any slot from 3 to 10, 9U × 340 mm (D49), through-hole (D21). D50, D52, D53, D57, D67. The netlist is `generators/card_video.py`, the font `generators/gen_font.py`.

**88 ICs, 18 of them GALs**; 207 parts (`python3 generators/card_video.py --bom`).

## 1. What it holds

VGA **640 × 480 at 60 Hz** (a 25.175 MHz dot clock): **80 × 25 text** in 8 × 16 cells from a font ROM, on the screen's lines 40–439, and a **640 × 480 one-bit overlay**, each layer switched on and off; **two PS/2 ports** (keyboard and mouse). The console (D53).

**Its RAM**, 64 KiB at `0xFE00_0000 + n × 64 KiB` (n = the window jumpers; default 5, `0xFE05_0000`), bytes, halfwords and words, at any time:

| Offset | What |
|---|---|
| `0x0000`–`0x0F9F` | **text**: 25 rows × 80 cells; a cell a halfword: the character, then its attribute (bit 0 inverse, 1 underline, 2 bright, 3 blink) |
| `0x1000`–`0xA5FF` | **overlay**: 480 lines × 80 bytes, bit 7 the leftmost dot (D67: first planned at `0x8000`, where it does not fit) |
| the rest | RAM, not shown |

**Its registers**, bytes, in the window's top page (`0xFFFF_DF00` by default):

| Offset | Read | Write |
|---|---|---|
| `+00` | control | bit 0 TEXT, 1 OVERLAY, 2 XOR (else OR), 3 CURSOR |
| `+01` | status: bit 0 VBLANK, 1 DE (a drawn dot's column and line) | taken, ignored |
| `+02` / `+03` | cursor row / column | the same |
| `+08` | keyboard data (reading takes it; 0 when nothing waits) | a command byte to the keyboard |
| `+09` | keyboard status: bit 0 FULL, 1 BUSY (a command being sent) | taken, ignored |
| `+0C` / `+0D` | the mouse's, as `+08` / `+09` | |
| `+10` | interrupt enables: bit 0 keyboard, 1 mouse | the same |
| `+F0` / `+F1` | card ID 3 / revision 0 | taken, ignored |
| `+F4` | interrupt status (a word: bit 0 keyboard FULL, 1 mouse FULL) | taken, ignored |

Anything else in the window, or another size, gets no `-ACK`: a bus error. Reset clears control, the cursor and the enables: a black screen.

**The picture.** A text dot is the font's, unless the cell blinks and the blink is off (32 frames, ~1.9 Hz); the underline lights scan line 15; inverse inverts the cell; the cursor (CURSOR on, its row and column) inverts scan lines 14–15 while the blink is on. Bright text is 0.7 V, the rest 0.5 V, on all three guns (white). The overlay ORs over the text, or with XOR inverts it.

## 2. Jumpers (at the card's edge, labelled)

Fitted = 0, open = 1:

| Jumpers | Meaning | Default |
|---|---|---|
| `WIN2 WIN1 WIN0` | the window: registers at `0xFFFF_8F00 + n × 4 KiB`, RAM at `0xFE00_0000 + n × 64 KiB` | 5: WIN1 fitted |
| `L2 L1 L0` | the interrupt level (both ports) | 4: L0, L1 fitted |

## 3. Blocks

| Block | Parts | ICs |
|---|---|---|
| **Bus interface** | 3 × 74HCT541 (A2–A15, BE, strobes, CLK, `-CLK`, `-RES`: one load a line); 74HCT688 (A31–A25); GALs VHI (A24–A16: the RAM's window, the device windows), VSEL (the access: RAM or register page), VBE (lanes, `-ACK`, the transceivers), VDEC (register reads), VWR (register writes); 4 × 74HCT245 (D ↔ the card's XD0–31); 74HC05 (`-ACK`, `-WAIT`, the PS/2 lines: open drain); pull-downs on XD0–15 | 15 |
| **Registers** | GAL VCTL (control, enables, the interrupt); GALs VRD0 / VRD1 (the lane 0 and lane 1 registers read); 74HCT32; 74HCT244 (the vector's window); 2 × 74HCT273 + 2 × 74HCT541 (the cursor, read back) | 8 |
| **Timing** | 25.175 MHz oscillator; 74ACT04 (DCLK, DCLKN); 6 × 74F163 (DC: the dot; HC: the character, 0–99; VC: the line, 0–524); 4 × 74F161 (the overlay's address); GALs HORZ, VERT (the decodes, sync), VTIM (the slots on DCLK's rise, the handshake), VSLOT (on its fall: whose address the RAM sees), VBUS (on `-CLK`: the handshake's bus side); 74HCT393 (frames: the blink) | 17 |
| **RAM** | 2 × CY7C199 (32K × 8, 15 ns: the even bytes, the odd bytes); 6 × 74ACT541 (the text's, the overlay's and the bus's addresses); GAL ROW5 + 2 × 74F283 (row × 80 + column); 4 × 74HCT541 (write buffers), 4 × 74HCT574 (read latches) | 19 |
| **Display** | 74HCT574 × 2 (CL: the character; AL: the attribute, the cursor's match, the column's place); SST39SF010A (the font); 3 × 74ACT574 (FL: its dots; OL0, OL1: the overlay bytes); 2 × 74F151 (a character's dot, bit 7 first); 2 × 74HCT688 (the cursor); GALs CELL (a character's attributes), VOUT (the dots, the syncs); 74ACT541 (the VGA drivers) | 13 |
| **PS/2** | per port: GAL PS2K / PS2M, 74HCT163 (bits), 74HCT595 (receive), 74HCT597 (send); 74HCT14, 74HCT175 (the lines in), 74HC280 + 74HCT74 (the command's parity), 74HCT00 | 13 |
| **Interrupts** | 74LS145 (the level, open collector), 74HCT85 (the level acknowledged) | 2 |
| **Front edge** | the DE-15 (VGA) socket; two mini-DIN-6 (KEYBOARD, MOUSE), +5 V through a 0.5 A polyfuse | — |

## 4. Theory of operation

### 4.1 The raster

A **character time** is 8 dots (318 ns). DC counts the dots, HC the characters of a line (0–99: 80 drawn, then 2 front porch, 12 sync, 6 back porch: 800 dots, 31.47 kHz), VC the lines (0–524: 60 Hz). VC 0 is the screen's line 40, the first text line, so that **the text row is VC8:4 and the scan line VC3:0**; the overlay's line 0 is VC 485. After `-RES` (synchronized: DRESN) the counters start at 0. HC and VC advance at dot 5 of a character time (ENH), the overlay's address at dot 1 (OENA), so neither changes while a slot uses it.

### 4.2 The RAM's four slots

The RAM is **16 bits wide**: the even bytes in one CY7C199, the odd in the other, so a text cell (character and attribute) is one halfword. Each character time has four 2-dot slots. GAL VSLOT, on the dot clock's falling edges, puts a slot's address on the RAM (enabling one pair of 74ACT541s); GAL VTIM, on the rising edges, takes its data 1.5 dots later and lets the address go half a dot after that:

```
dot          0    1    2    3    4    5    6    7    0
address        [text      )[bus, low  )[bus, high )[overlay   )
data taken           TLK ^      RLKA ^     RLKB ^      BCK ^
-WE (a write)                 [   ]      [   ]        (per byte lane)
```

- **Text** (dots 0.5–2.5): the cell at **row × 80 + column** ((VC8:4 × 5 + HC6:4) × 16 + HC3:0: GAL ROW5 and two 74F283s). CL takes the character, AL the attribute, the cursor's match (two 74HCT688s: the row against VC8:4, the column against HC) and whether this column is text / drawn. The character and the scan line address the font ROM; at the next character time's start (BCK) FL takes the dots.
- **The bus** (dots 2.5–6.5): the word's low halfword, then its high one.
- **The overlay** (dots 6.5–8.5): the halfword holding the column's byte (the overlay's address counter, from `0x1000` at the frame's start, one byte a drawn column); OL0 and OL1 take both bytes at BCK, and the address's bit 0 (in GAL CELL) enables the right one.

So **a character's font and overlay bytes are both ready as its dots begin**. Two 74F151s pick each dot (DC selects, bit 7 first). GAL CELL has the character's attributes as K (the glyph shows) and M (light or invert regardless): the text dot is **(TD & K) ^ M**, the overlay's **OD & E**. GAL VOUT ORs or XORs them and registers every dot with the syncs (HSR from GAL HORZ, VSR from GAL VERT), so the dots and the syncs leave together. A 74ACT541 drives the resistors: a dot is 0.5 V into 75 Ω (680 Ω), a bright one 0.7 V (with 1.5 kΩ more), the syncs TTL through 47 Ω.

### 4.3 The bus: the handshake

The card **answers its RAM at once** ( `-ACK`) and holds `-WAIT` until the dot clock's side has done the access, whatever the bus clock:

1. REQ = the access and not DONEB (GAL VBUS). It is `-WAIT` itself, and crosses into the dot clock through two flip-flops (R1, R2 in GAL VTIM).
2. At dot 2 of a character time, with R2 and not DONE, GO starts the access: the bus's two slots in this character time. A write: the RAM's `-OE` off and the write buffers (XD → RD) on through both slots, `-WE` for a dot in each, per byte lane (BE0–3). A read: RLKA and RLKB clock the read latches (RD → XD), which drive the card's data bus for the whole cycle.
3. DONE (at dot 6) crosses back through one flip-flop clocked by `-CLK` (DONEB, mid-cycle): REQ, so `-WAIT`, ends; the cycle ends at the next CLK. With REQ gone, DONE clears.

At 3 MHz an access waits **one or two clocks** (from 8 to 16 dots after REQ). Back-to-back accesses work: DONEB is clear again by the middle of the next cycle's first clock, before `-WAIT` is looked at. **The registers** are on the bus's own clock and answer without waiting.

**Register writes are taken at the cycle's end** (D72): the cursor registers' and the PS/2 send registers' clocks fall in the bus clock's second half and rise with CLK as the cycle ends, so the address decode, the transceivers and the data have the whole cycle (at mid-cycle they did not, at the datasheets' worst case). The PS/2 GALs still see the command strobe (KCMDC / MCMDC) as a level in the second half.

### 4.4 PS/2

Each port is a **GAL state machine on the bus clock** (PS2K, PS2M), its clock and data lines read through a 74HCT14 and a 74HCT175 and pulled low through the 74HC05 (4.7 kΩ pull-ups).

- **Receiving.** A fall of the device's clock is CK2 & !CK1. A 74HCT163 counts them: the data bits (falls 2–9) shift into a **74HCT595**; the 11th sets **FULL**, which loads the 595's output register and **holds the clock low**, so the device waits with its next byte. Reading the data register (KRD) clears FULL. The 595 drives lane 0 only while FULL: an empty port reads 0 (the pull-downs).
- **Sending.** A write to the data register loads the byte into a **74HCT597** (and its odd parity, from a 74HC280, into a 74HCT74) and sets **BUSY**. The GAL holds the clock low for five line ticks (TICK: HSYNC through GAL VBUS: 127–159 µs, ≥ 100 µs), then pulls data low (the start bit) and lets the clock go. The device clocks the frame: at each fall the GAL presents the next bit (the 597 shifts at falls 2–9, the parity behind the eight data bits), lets data go for the stop bit, and the 11th fall (the device's acknowledge) ends BUSY. The answer (FA) arrives as an ordinary byte.
- **Interrupts** (D52, D57): FULL and its enable, either port, pull the card's level through the 74LS145. In the acknowledge cycle (the level matching, 74HCT85) the vector is **window × 16 + 0** (keyboard) or **+ 1** (mouse, if the keyboard is not asking): `0x50` / `0x51` by default.

## 5. The font

`generators/gen_font.py` draws printable ASCII on an 8 × 8 grid (capitals on rows 0–6, descenders on row 7), each row shown on two scan lines: the look of an 8 × 8 font on a VGA screen. `python3 generators/gen_font.py` writes `video/font8x16.bin`, the whole SST39SF010A (128 KiB: the font at the start, FF after it); address = character × 16 + scan line.

## 6. Verification

- Its 18 GALs against their specs, exhaustively; the netlist's electrical checks.
- **At gate level with the CPU, MMU / data, memory and I/O cards** (`generators/sim_sys.py --mem --io --video`), the card whole: its own 25.175 MHz clock against the CPU card's 24 MHz (generators/sim_video.py); only the PS/2 devices are models, driving the sockets' lines.
  - `vidcard_test`: the RAM in every width and lane (each access through the handshake), the registers, the status register three times as the raster moves, five refused accesses, then a screen. Every instruction's state and every cycle as `p32uemu -C`, which now models the card's timing exactly: the two clocks' schedule, the handshake's flip-flops and the raster (`p32io.h`, `vid_hw_*`). All 614 RAM accesses wait as many clocks in both.
  - **The picture** (`--frame PNG`): after HALT the dot clock runs on for a frame; the VGA outputs, cut by the syncs, must be dot for dot the picture the card's RAM and registers describe (`sim_video.expected`), with HSYNC 96 dots every 800 and VSYNC 2 lines every 525, and nothing lit outside 640 × 480.
  - `ps2_test`: the keyboard's bytes as they arrive, its vectored interrupt (`0x80005004`), the mouse's reset (FF: FA AA 00), a command with its data byte (ED 02: FA FA), the parity checked by the device. Self-checking on both emulators and the card; not cycle for cycle (on the card a byte takes a frame on the wire).

## 7. To check before layout

- The mini-DIN-6 footprint (Same Sky MD-60SM: as on YACC1's video card) in `hardware/lib/P32W.pretty`.
- The 25.175 MHz oscillator in a DIP-14 can (25.000 MHz works with most monitors, if that is what is in stock).

- Timing against the datasheets: the slots assume the 74ACT541's enable times and the CY7C199-15's access (≤ 45 ns of the 60); the dot-rate GALs are the -7 grade (ATF22V10C-7PX).
- **Programming** (TL866II+ or T48): the GALs' fuse files are `hardware/gal/video/*.jed` (`minipro -p ATF22V10C -w FILE`), the font `video/font8x16.bin` (`minipro -p SST39SF010A -w ...`): hardware/gal/README.md.

# The video card: theory of operation

docs/video-card-theory.md

**Superseded (D98, 2026-11-10; D103, 2026-11-12):** revision 0, being replaced by revision 2, an HD63484 ACRTC card; never ordered or built. The emulators and the kernel model revision 2 since D105 (2026-11-12), so §13's gate-level runs against them no longer exist; and `generators/card_video.py` is revision 2's netlist since D106 (2026-11-12): the generator this page describes is in the repository's history, last in commit `78473d7`.

**Status:** DESCRIBES THE NETLIST AS BUILT AND VERIFIED (2026-10-30), `generators/card_video.py` (every chip and GAL equation comes from there; the font from `generators/gen_font.py`). The card's summary, its parts by block and its verification are in the video card; the registers as programs see them in the programmer's guide §6.6; the bus in bus.md. D50, D52, D53, D57, D67, D71, D72.

## 1. What the card does

A **bus slave**, in any slot from 3 to 10. From its own 25.175 MHz dot clock it draws VGA, 640 × 480 at 60 Hz, in two layers, each switched on and off: **text**, 80 × 25 cells of 8 × 16 dots from a font ROM on the screen's lines 40–439, each cell with an attribute (inverse, underline, bright, blink), and a **cursor**; and **the overlay**, 640 × 480 one-bit dots, ORed or XORed over the text. Both are in **64 KiB of RAM** at `0xFE00_0000 + n × 64 KiB`, which the bus reads and writes at any time: the text at `+0000`–`+0F9F` (a cell a halfword: the character, then its attribute), the overlay at `+1000`–`+A5FF` (80 bytes a line, bit 7 the leftmost dot). The registers are the top 256 bytes of device window n, `0xFFFF_8F00 + n × 4 KiB` (D52); n is 5 by default. **Two PS/2 ports**, keyboard and mouse, share one interrupt level, with a vector each. Fitted, the card is the console, `/dev/tty4` (D53). The display runs on the dot clock (`DCLK`), the rest on the bus clock (`CLK`); what crosses between them is in §4, §9 and §10.

## 2. The jumpers

Six jumpers at the card's edge (JP1–JP6). **Fitted reads 0, open reads 1**: each shorts its line to ground, and a 10k network (RN3) pulls it up.

| Jumpers | Meaning | Default |
|---|---|---|
| `WIN2 WIN1 WIN0` | the window n: the RAM's 64 KiB, the register page, bits 6:4 of the vector | 5: WIN1 fitted |
| `L2 L1 L0` | the interrupt level, both ports | 4: L1 and L0 fitted |

## 3. The bus interface: five GALs

Three 74HCT541s buffer A2–A15, BE0–3, `-MRD`, `-MWR`, `-IACK`, `CLK`, `-CLK` and `-RES` (`BA2–15`, `BBE0–3`, `MRDN` … `BRESN`); a 74HCT688 finds A25–A31 all ones (`TOPN`); GAL VHI reads A16–A24: one load per line. Four 74HCT245s join D0–31 to the card's data bus, `XD0–31`; RN1 and RN2 pull `XD0–15` down, so a register's unused bits read 0.

- **VHI**: `MHIT`, A31:16 = `FE00` + n (the RAM); `RHI`, A31:16 = `FFFF`.
- **VSEL**: `MACC`, `MHIT` with `-MRD` or `-MWR` (a RAM access, any size); `RRD` / `RWR`, the register page (`RHI`, A15 = 1, A14:12 = n, A11:8 = F) with `-MRD` / `-MWR`; `RLOEN`, a RAM read: the read latches onto XD.
- **VBE**: `BYTE`, `WORD`, `LN1:0` (a byte's lane); `ACK` = `MACC` or `RHIT` or `VHIT` (§11), onto the open-drain `-ACK` through a 74HC05; `LE0N`–`LE3N`, a lane's 245 on when the card answers and the master asks for that lane; `LDIR`, toward the card for a write.

- **VDEC**, from the byte's offset (`BA7:2`, `LN1:0`): `RHIT`, a byte at `+00`–`+03`, `+08`, `+09`, `+0C`, `+0D`, `+10`, `+F0`, `+F1` or a word at `+F4` (anything else in the window, or another size, gets no `-ACK`: a bus error); `KRD` / `MRDP`, a byte read of `+08` / `+0C`; `ROWRDN` / `COLRDN`, the cursor's read-back; `R0EN0`, a read for GAL VRD0.
- **VWR**: `WCTL`, `WIE` (`+00`, `+10`), levels that GAL VCTL takes at the `CLK` edge ending the cycle; `KCMDC`, `MCMDC` (`+08`, `+0C`), levels in the second half (`CLK` low) for the PS/2 GALs; `WROWC`, `WCOLC`, `KCMDK`, `MCMDK`, clocks low in the second half that rise as the cycle ends, so the decode, the transceivers and the data have the whole cycle (D72). A register write takes effect as its cycle ends, and the registers answer with **no wait state**; only the RAM waits (§9).

## 4. The registers

| Offset | Read | Write | Parts |
|---|---|---|---|
| `+00` | control | bit 0 TEXT, 1 OVERLAY, 2 XOR, 3 CURSOR | GAL VCTL (`CTL0–3`); read through GAL VRD0 |
| `+01` | status: bit 0 VBLANK, 1 DE | ignored | GAL VRD1 |
| `+02` / `+03` | cursor row / column | the same | a 74HCT273 each, cleared by `-RES`; a 74HCT541 reads it back |
| `+08` / `+0C` | keyboard / mouse data | a command byte | the port's 74HCT595 / 74HCT597 (§10) |
| `+09` / `+0D` | its status: bit 0 FULL, 1 BUSY | ignored | GAL VRD1 |
| `+10` | interrupt enables: bit 0 keyboard, 1 mouse | the same | GAL VCTL (`KIE`, `MIE`); read through VRD0 |
| `+F0` / `+F1` | card ID 3 / revision 0 | ignored | VRD0 / the pull-downs |
| `+F4` | interrupt status, a word: bit 0 keyboard FULL, 1 mouse FULL | ignored | VRD0 |

A byte register is on its offset's lane (`+02` on lane 2, `XD16–23`).

- **VCTL**, on `CLK`, cleared by `-RES`: `CTL0–3` take XD0–3 with `WCTL`, `KIE` and `MIE` XD0–1 with `WIE`; it also makes the interrupt (§11).
- **VRD0** drives XD0–3 while `R0EN` (`R0EN0` or `VHIT`, a 74HCT32): by BA7, BA4 and BA2 the control register, the enables, the ID 3, or `KF` and `MF` (the ports' FULL); in the acknowledge cycle the device (§11).
- **VRD1** drives XD8–9 for a byte read at `+01`, `+05`, `+09` or `+0D` (`+05` gets no `-ACK`): VBLANK (not `VIS`) and DE (`OOUT`: the column being fetched is drawn, on a drawn line), straight from the dot clock's side as they are when the read ends; or the port's FULL and BUSY.

Reset clears control, the enables and the cursor: a black screen.

## 5. The dot clock and the counters

A 25.175 MHz oscillator drives a 74ACT04 (`DCLKN`, then `DCLK`). A dot is 39.7 ns, a **character time** 8 dots (318 ns). "At dot n" is the rising edge that starts dot n (dot 8 is the next character time's dot 0); "n.5" is the falling edge in its middle.

| Counter | Parts | Counts | Cleared |
|---|---|---|---|
| **DC**, the dot, 0–7 | 74F163 | every dot | in reset (`DRESN`) |
| **HC**, the character, 0–99 | 2 × 74F163 | at dot 5 (`ENH`) | after 99; in reset (`HCLRN`) |

| Counter | Parts | Counts | Cleared |
|---|---|---|---|
| **VC**, the line, 0–524 | 3 × 74F163 | as HC wraps (`VENA`) | after 524; in reset (`VCLRN`) |
| **OA**, the overlay's byte address | 4 × 74F161 | at dot 1 (`OENA`) | loads `0x1000` (`OALDN`) |

**GAL HORZ**, on `DCLK`, makes `DRESN` (`-RES` on the dot clock); `ENH` (DC = 4); `HCLRN` and `VENA` (DC = 4, HC = 99); `OENA` (DC = 0, HC 1–80, a drawn line); `TOUT` / `OOUT` (HC < 80 on a text / drawn line); and `HSR`, the horizontal sync, registered at dot 8 (§6). **GAL VERT**, combinational: `VCLRN`; `TL` (VC < 400); `VIS` (VC < 440 or ≥ 485); `VSR` (VC 450–451); `UL` (VC3:0 = 15, the underline's line); `CL` (VC3:1 = 7, the cursor's lines 14–15); `OALDN`, low on undrawn lines and in reset. So HC and VC change at dot 5 and OA at dot 1, never while a slot uses them. **The blink:** a 74HCT393 counts the falls of `VSR`, one a frame; its bit 4 is `BLINK`, 16 frames low and 16 high (about 1.9 Hz), never cleared.

## 6. A frame and a line

**A line** is 100 character times, 800 dots (31.47 kHz). Column h is fetched in the character time that begins with HC = h, and its dots go out in the next (§8). Columns 0–79 are drawn (640 dots), 80–81 are the front porch, 82–93 HSYNC (96 dots), 94–99 the back porch: as column h's fetch ends HC is already h + 1, and HORZ registers `HSR` = (HC is 83–94), so the sync goes out with columns 82–93.

**A frame** is 525 lines (60 Hz). VC 0 is the screen's line 40, so that on a text line **the row is VC8:4 and the scan line VC3:0**:

| VC | Screen lines | Drawn |
|---|---|---|
| 0–399 | 40–439 | text rows 0–24, and the overlay |
| 400–439 | 440–479 | the overlay |
| 440–484 | 480–524 | nothing (VBLANK); VSYNC on VC 450–451, lines 490–491 |
| 485–524 | 0–39 | the overlay, whose line 0 is VC 485 |

On undrawn lines OA loads `0x1000`; from VC 485 it counts a byte a drawn column, reaching `0xA5FF` in the 480th line. After `-RES` the raster starts from DC, HC and VC 0, the first text line, with OA at `0x1000`.

## 7. The RAM, its four slots, the text address

The RAM is **16 bits wide**: two CY7C199 (32 K × 8, 15 ns), the even bytes (`RD0–7`, the characters) and the odd (`RD8–15`, the attributes); `RA0–14` addresses a halfword, and `-CE` is grounded. Three pairs of 74ACT541s offer an address: the text's (HC3:0, TA4–10, zeros; `TXENN`), the overlay's (OA15:1; `OVENN`) and the bus's (`HALF`, BA2–15; `CPENN`). A character time has **four slots of two dots**: **GAL VSLOT**, on falling edges, puts a slot's address on; **GAL VTIM**, on rising edges, takes the data a dot and a half later.

```
dot           0     1     2     3     4     5     6     7     0
address          [text      )[bus, low  )[bus, high )[overlay   )
data taken            TLK ^      RLKA ^      RLKB ^       BCK ^
-WE (a write)                    [     ]     [     ]
```

The bus slots serve lanes 0–1 (`HALF` = 0), then lanes 2–3 (`HALF` = 1). VSLOT makes `TXENN`, `CPENN`, `HALF` and `OVENN` from DC alone. For a write in progress (`GO` and `-MWR`) it also turns the RAM's `-OE` off (`ROEN`, 2.5–6.5) and the write buffers on (`WBAN`, lanes 0–1, 2.5–4.5; `WBBN`, lanes 2–3, 4.5–6.5); its active-low outputs reset high, off at power-up. VTIM's `WEEN` (the even chip) is low in dot 3 with BE0 and in dot 5 with BE2, `WEON` (the odd) with BE1 and BE3: half a

dot inside the address and the data. Four 74HCT541 write buffers carry lanes 0–1 or 2–3 onto `RD0–15`; four 74HCT574 read latches take `RD0–15` back to lanes 0–1 (`RLKA`) and 2–3 (`RLKB`), driving XD while `RLOEN`. So the bus's byte at offset x is in halfword x / 2, in the even or odd chip by bit 0: where the text and the overlay read it.

**The text address: row × 5.** The cell is halfword row × 80 + column, the row in VC8:4 and the column in HC6:0. **GAL ROW5** makes row × 5 (`R50–R56`, rows 0–24), and two **74F283**s add HC6:4 to it (the low one's carry, `TAC`, into the high one):

TA10:4 = row × 5 + HC6:4; RA = TA10:4 × 16 + HC3:0 = row × 80 + HC

Row 24, column 79 is halfword 1999: bytes `0x0F9E`–`0x0F9F`.

## 8. A character cell, and the VGA output

Column h of a text line, through the character time that begins with HC = h, and the next:

| Dot | What happens |
|---|---|
| 0.5 | VSLOT puts the cell's address on the RAM |
| 2 (`TLK`) | **CL** takes the character; **AL** the attribute's bits 0–3, the cursor's match (`CEQN`), `TOUT` and `OOUT` (74HCT574s). The font ROM sees the character and the scan line |
| 6.5 | OA, at column h's byte since dot 1, on the RAM |
| 8 (`BCK`) | **FL** takes the font's byte, **OL0** / **OL1** the overlay halfword (74ACT574s), **GAL CELL** the attributes, HORZ the sync |
| the next 8 | two **74F151**s pick bit 7 − DC of FL (`TD`) and of the enabled overlay latch (`OD`); **GAL VOUT** registers each dot at the next edge |

So a column's dots leave a character time after its fetch, one dot behind the 151s, with the syncs registered beside them. Two 74HCT688s find **the cursor's cell**: the row register against VC8:4 and, enabled by that, the column register against HC; outside the text area `TOUT` hides it. **The font** is an SST39SF010A flash (128 K × 8, 55 ns): A3–A0 the scan line, A11–A4 the character, A12–A16 grounded. Its image, `video/font8x16.bin`, is the whole chip: printable ASCII (20–7E) on an 8 × 8 grid, each row on two scan lines, the other codes blank, then FF.

**GAL CELL**, on `BCK`, makes `K`, the glyph shows: TEXT on, a text column (`TOUTL`), not blink with `BLINK` high, not underline on scan line 15; `M`, lit or inverted whatever the glyph: TEXT on, a text column, and (underline on scan line 15) XOR inverse XOR the cursor (CURSOR on, `CEQL` low, `CL`, `BLINK` low); `BRTT`: TEXT on, a text column, bright; `E`: OVERLAY on, a drawn column (`OOUTL`); `OSEL` / `OSELN`: OA0, OL0 or OL1 onto the 151. The text dot is **(TD & K) ^ M**: dark with TEXT off or outside the text; else the font's, inverted for inverse; the background (dark, lit if inverse) in a blinking cell while `BLINK` is high; lit (dark if inverted) on an underline's scan line 15; inverted on the cursor's lines 14–15 while `BLINK` is low. **GAL VOUT** ORs it with the overlay's dot, `OD & E`, or XORs while XOR (`CTL2`) is set: `VID`. `VBR` is `VID` in a bright cell; `HSO` and `VSO` are the syncs, active low.

**The VGA output.** A 74ACT541 drives `VID` and `VBR` three times each, and the syncs. Each gun's pin (DE-15 pins 1, 2, 3) has 680 Ω from a `VID` output and 1.5 kΩ from a `VBR` output: into the monitor's 75 Ω a dot is 0.5 V and a bright one 0.7 V, on all three guns (white). HSYNC and VSYNC reach pins 13 and 14 through 47 Ω; pins 5–8, 10 and the shell are ground. The socket, J4 (VGA), is at the front edge.

## 9. A bus access to the video RAM

The card **answers at once** (`-ACK`) and **holds** `-WAIT` until the dot clock's side has done the access, so it is right at any bus clock (D67). `REQ` (GAL VBUS) is `MACC` and not `DONEB`, and through the 74HC05 it is `-WAIT`. GAL VTIM, on `DCLK`, takes `REQ` through two flip-flops (`R1`, `R2`); sets `GO` at dot 2 with `R2` and not `DONE`, until dot 8; and sets `DONE` at dot 6

while GO, holding it while R2. VBUS takes DONE on -CLK, mid-clock: DONEB.

**A word write:**

1. The MMU / data card drives A, BE0–3, -MWR and D from the clock's start: MACC, -ACK, the transceivers toward the card; REQ rises (DONEB is low) and -WAIT falls. R1, R2 follow at two dot edges.
2. At the next dot 2, GO. From 2.5 the RAM's outputs are off, lanes 0–1's buffers on and HALF 0: -WE for dot 3. From 4.5 HALF is 1 and lanes 2–3's buffers are on: -WE for dot 5. From 6.5 the RAM is the overlay's.
3. At dot 6, DONE; at the next mid-clock, DONEB: REQ falls and -WAIT is let go. The MMU / data card finds -WAIT high as that clock ends, and the cycle ends. With MACC gone, R1, R2 and then DONE clear; DONEB at the next mid-clock.

**A read** is the same with the RAM's outputs on: RLKA and RLKB take both halfwords, and the read latches drive XD all through the cycle (RLOEN); the transceivers pass the lanes asked for. A byte or halfword write strobes only its own lanes.

**How long.** By VTIM's equations DONE comes 6 to 14 dots after REQ rises: two or three dots until R2 can start GO, up to seven more for dot 2, four to dot 6. At 3 MHz the first mid-clock (167 ns) is always too soon and the third (833 ns) always late enough: **one or two wait states**. **Back to back**, DONEB is still high as the next access begins, so it starts without -WAIT; DONE has cleared by its mid-clock (three dot edges after REQ fell), and REQ pulls -WAIT low before that clock ends (video-card.md §4.3).

## 10. PS/2

Each port's lines come in through a 74HCT14 (Schmitt) and a 74HCT175 on CLK (its inverted outputs, KCK1, KDAT1, MCK1, MDAT1, in the lines' own sense), and go low through the 74HC05 (CKDRV, DTDRV) against 4.7 kΩ pull-ups; the sockets (J5 KEYBOARD, J6 MOUSE) take +5 V through a 0.5 A polyfuse. Per port: **GAL PS2K** or **PS2M** on CLK, cleared by -RES; a 74HCT163 (BC, which -RES loads with 0); a 74HCT595 to receive, a 74HCT597 to send. Shared: a 74HC280 (parity), a 74HCT74 (a parity bit per port), a 74HCT00 (a data register onto lane 0 only while FULL). GAL VBUS takes HSR twice on -CLK: TICK (HS1 & !HS2) is one bus clock in each line.

The GAL sees a fall of the device's clock as CK2 & !CK1 (CK2 is CK1 a clock later). CNT counts falls, or line ticks in the inhibit. CLRN clears the count on a command write, the 11th fall, the start of sending, and while FULL and not BUSY. SH pulses at falls 2–9 (BC 1–8), shifting the 595 and the 597. CKDRV is FULL, or BUSY before TXA (the inhibit).

**Receiving.** Idle, the card drives neither line. The keyboard clocks out a start bit, eight data bits (bit 0 first), odd parity and a stop bit. Falls 2–9 shift KDAT1 into the 595; the start, parity and stop bits are counted, not checked. The 11th fall sets FULL: the 595's output register takes the byte (RCLK), the clock is held low (the keyboard waits with its next byte), the count is held at 0, and with KIE the card asks for its interrupt. A read of +08 (KRD, with FULL) turns the 595 onto XD0–7, bit 0 on XD0; as the cycle ends FULL clears and the clock is let go. Without FULL the read gives the pull-downs' 0.

**Sending** (ED, say):

1. A byte write to +08: as the cycle ends KCMDK rises, the 597's input register takes XD0–7 (bit 0 into H), and the 74HCT74 the 74HC280's ΣEVEN (odd parity). The GAL sets BUSY and clears the count.
2. The inhibit holds the clock low while the counter counts line ticks. At the fifth (BC = 4) TXA rises, the data goes low (the start bit), the count clears and the clock is let go: after 4 to 5 lines, 127–159 µs (≥ 100 µs, D67).

3. The keyboard clocks the frame. With `TXA` high the 597 stops loading (`-SRLOAD`) and shifts at falls 2–9, the parity entering behind. After each fall `DTDRV` drives the next bit: `BC` 0 the start bit, 1–9 a 0 from QH' (the data, then the parity), 10 nothing (the stop bit).
4. The 11th fall, the keyboard's acknowledge, ends `TXA` and BUSY. Its answer (`FA`) arrives as a received byte.

A command written while a byte waits unread is held: after the inhibit the clock stays low (`CKDRV` = FULL) until the byte is read.

## 11. Interrupts

**GAL VCTL** asks with `KF & KIE` or `MF & MIE`. Its inverse, `ANYN`, is the **74LS145**'s D input, with L2–L0 on C, B, A: with a request the code is the level, and that output pulls its `-IRQ` line low (open collector); without one the code is 8 or more and O0–O7 are off. `VDEV` is 0 if the keyboard asks, else 1. A **74HCT85** compares BA4:2 with L2:0 (`EQ`), and `VHIT` is `-IACK` with `EQ` and a request.

**The acknowledge cycle** (D57): the MMU / data card drives `-IACK`, A4:A2 = the level and BE0 alone. `VHIT` makes VBE raise `ACK` and turn lane 0's transceiver toward the bus, and makes `R0EN` (through the 74HCT32): VRD0 drives XD0 = `VDEV` and XD1–3 = 0, and a 74HCT244, enabled by `VHITN`, drives XD4–6 = the window and XD7 = 0. The vector is **window × 16 + 0** (the keyboard) or **+ 1** (the mouse, when the keyboard is not asking): `0x50` / `0x51` by default. The CPU puts it in `CAUSE` bits 15:8: the keyboard at level 4 is `0x8000_5004`. For another level, or with no request, the card does not answer (vector `0xFF`, D57). Reading the data register clears the request. `+F4` shows FULL whether enabled or not.

## 12. Timing

| What | Time |
|---|---|
| a slot's address to its data taken | 1.5 dots, about 60 ns, for the 74ACT541's enable, the CY7C199-15 and the latch (≤ 45 ns used, video-card.md §7) |
| the font, `TLK` to `BCK` | 6 dots, about 238 ns, for the 55 ns flash |
| the dot clock's paths (timing.md, D71) | 37.5 ns of the 39.7 at worst case; 23.2 ns typical |
| `-WAIT`'s release (timing.md §4) | about 69 ns through the backplane's 680 Ω, of the 167 ns it has at 3 MHz |

The dot clock's GALs (HORZ, VERT, VTIM, VSLOT, VOUT) are ATF22V10C-7PX, the others -10. The bus side is timed with the whole machine, which meets 3 MHz at worst case (D72); paths between the two clocks are not timed, nor is metastability (timing.md §3, §5).

## 13. Verified

`python3 generators/card_video.py` (in `make test`) checks every GAL's equations exhaustively against its spec, and the netlist's electrical rules; the fuse files are checked through the 22V10's fuse map (D67). At gate level with the CPU, MMU / data, memory and I/O cards (`sim_sys.py --mem --io --video`), the card's 25.175 MHz against the CPU card's 24 MHz (`sim_video.DotClock`), **vidcard_test** (the RAM in every width and lane, sixteen words back to back, the registers, the status three times, five refused accesses, a screen) must match `p32uemu -C`, which models the handshake and the raster, in every instruction's state and the cycle count. With `P32W_FRAME=1` only (about 12 minutes more), the frame drawn after HALT must be dot for dot what the RAM and registers describe (`sim_video.expected`):

HSYNC 96 dots every 800, VSYNC 2 lines every 525. **ps2_test**, on `p32uemu -C` and at gate level with PS/2 device models on the sockets' lines (`sim_video.PS2Device`), checks the keyboard's bytes, its vector (`0x8000_5004`), the mouse's reset (`FF: FA AA 00`) and a command (`ED 02: FA FA`) whose parity the device checks; not cycle for cycle.

## 14. Open questions

1. `+F4`**'s upper half floats.** A word read of `+F4` turns on all four lanes, but nothing drives `XD16–31` and, unlike `XD0–15`, they have no pull-downs: D16–31 are undefined (floating in `p32sim`, the card alone). vidcard_test and ps2_test read the word and expect 0.
2. **The wait, in dots.** video-card.md §4.3: "from 8 to 16 dots after REQ". By VTIM's equations `DONE` comes 6 to 14 dots after `REQ`; 8 to 16 is the end of the access's character time.
3. **Block counts.** video-card.md §3 has 15, 8 and 17 ICs for the bus interface, the registers and the timing (87, not its 88); its part lists and the netlist give 14 (U1–U14), 9 (U15–U23) and 18 (U24–U41).
4. **D52** calls the card's RAM "text, font and overlay"; as built (D67) the font is in a ROM.
5. **Test runs.** video-card.md §6 has ps2_test "on both emulators and the card"; `make test` runs it on `p32uemu -C` and at gate level only, and the picture check only with `P32W_FRAME` set, which §6 does not say.
6. **BUSY** (`+09` / `+0D` bit 1) is on the card but not in the emulators: `p32io.h`'s `vid_read` returns FULL alone.
7. **Received frames.** The PS/2 GALs read no data line, so a frame's start, parity and stop bits are not checked; no document says whether that is intended.
8. **BE0 in the acknowledge cycle.** bus.md says BE0–BE3 do not matter there, but VBE turns lane 0 on only with BE0: the vector reaches D7:0 because the MMU / data card drives BE0 alone (mmu-card-theory §6).

# The debug card (bus tester)

docs/debug-card.md

**Status:** NETLIST DONE AND VERIFIED at gate level (2026-10-26; §8): `generators/card_debug.py`, **144 ICs, 6 of them GALs**, plus the Pico 2; 306 parts. §7's choices are Ken's. **D100 and D101 (2026-11-11), getting it ready to order:** a two-digit status display (two TIL311s); the mode switch turned to the edge; the capture SRAMs' write race, the power-on state and `-BREQ0` in LISTEN put right; DRIVE reaching the grants, `-FAST` and the interrupt lines; and this page brought into line with the netlist (how the card works in detail: its theory of operation). **D102 (2026-11-11):** it cannot drive against a CPU or MMU / data card that is in its slot (`-CPUIN`, `-MMUIN`; §1, §4). **THE BOARD IS ROUTED (2026-11-11, again for D102):** 6 layers, 620 nets, nothing left open, 137.3 m of 0.2 mm track and 425 vias; the pre-order check (`pcb-check`, with the system's tests) passes with no failures. Before an order, by hand: which end of S1 is LISTEN (a meter on the part), that the TIL311s' digits stand upright with the front edge up, the fab's gerber viewer. D50 (the host link, DMA with the CPU halted, a fast clock), D64 (listen or drive, never both; every signal; the dedicated slot 0).

## 1. What it is for

Bring-up and test, in two modes selected by a switch at its front edge:

- **LISTEN**: a logic analyzer on **every backplane signal, the CPU's local link (J3) included**. Every one of its drivers is held off, by wiring and not by software, so it can never disturb a running machine.
- **DRIVE**: it is the machine's master. It runs bus cycles and sets every line a card other than the CPU card reads:
  - **with the CPU and MMU / data cards present**, it borrows the bus by DMA (slot 0's own request line, the arbiter's highest priority): it loads programs, reads and writes memory and devices, with the CPU halted or running;
  - **with them pulled**, it also makes `CLK`, `-CLK` and `-RES`, so a memory, I/O, disk or video card can be tested alone. It plays the other masters' `-BREQ` lines to test the arbiter, and plays the CPU card's side of J3 (R, `RSRC`, `RDST`, `MEM`, `SIZE`, `CCLK`, `-WSTB` …) to test the MMU / data card alone, reading back `FAULT`, `FC`, `STALL` and R.

  **It cannot drive against a card that is in** (D102; Ken: "if debug is on drive mode is there protection that it wont drive bus if CPU is?"). Beside a running CPU a cycle needs the arbiter's grant, as it always did. The "alone" setting is the host's word that the cards are out, and the card no longer takes it on trust: the CPU card and the MMU / data card each tie a line to ground (`-CPUIN`, `-MMUIN`), and while a line is low the card's drivers for that card's lines stay off, by wiring (§4).

A Pico 2 is the **host link**: USB to the Mac, a small command set (read / write memory, run a bus cycle, arm a capture, read it back).

## 2. Slot 0 (D64)

An eleventh backplane position beside slot 1, wired to:

- **J1**: the whole bus cycle (70 lines);
- **J2**: the system lines (`CLK`, `-CLK`, `-RES`, `PWROK`, `-LOCK`, `-IRQ0–7`, `-IACK`), the eight radial `-BREQ3–10` / `-BGNT3–10` pairs as slot 2 sees them, and **its own pair**, `-BREQ0` / `-BGNT0`, to the arbiter;

- **J3**: the local link (62 lines), a third drop on lines that today run only between slots 1 and 2; and the two lines that say which of those cards are in, `-CPUIN` and `-MMUIN` (D102).

**169 signals in all** (167 sampled: `XCLK` and `-XCLKEN` are the card's own). This needed two changes elsewhere, on cards still netlist-only (none built or ordered):

- **the backplane** (its generator): the eleventh position, J3 and the fan-in carried to it;
- **the MMU / data card's arbiter**: a ninth requester, slot 0, at the highest priority. GAL ARB grows from 9 outputs to 10, its limit.

## 3. LISTEN: capture

- **What is sampled:** all 167 lines, into **22 × 74ACT574** input latches. The sample clock is one of:
  - `CLK`**'s rise**, one sample per bus cycle (and per microcycle: `CCLK` is the same edge), the state at the cycle's end. This is the normal mode, and its samples read like the emulators' traces.
  - **the card's own oscillator halved** (12 MHz: four samples per microcycle; D73: at 24 MHz the capture path did not meet its clock at the datasheets' worst case), a timing view for glitches and set-up problems.
- **Storage:** 22 SRAM chips side by side, one 8-bit slice each, with a counter for the address: a circular buffer that keeps running until the trigger, then stops after a chosen number of samples (pre- and post-trigger). A sample is written in the half clock after it is taken (D101), and a bit tells the host the buffer has gone round.
- **The trigger:** a pattern in **trigger RAM**. The address's two halves each address an SRAM whose stored bit says "this value matches". The host fills it, so any mask or range of addresses costs no extra logic: 8 independent trigger patterns, on any bus cycle (a read, a write or an acknowledge alike). It can also simply start and stop by command.
- **Read-out:** after the capture, the Pico reads the SRAMs slice by slice over the card's local bus and sends them to the Mac. A Mac tool turns them into a trace (`p32emu -s`'s format for the bus view, or a VCD file for GTKWave).

## 4. DRIVE: playing the master

- **Output registers: 19 × 74HCT574**, whose 3-state outputs reach every line it may drive. The register for a line is enabled only in DRIVE mode, and only for the lines that mode's sub-setting allows:
  - *bus master*: A, BE, strobes, D, `-BREQ0`;
  - *system* (alone): `CLK`, `-CLK`, `-RES`, `-BREQ3–10`, and since D101 `-BGNT3–10` and `-FAST`;
  - *CPU side of J3* (alone). The mode switch's LISTEN position ties every enable off.
- **Never against a card that is in** (D102). Two lines on J3 tell the card who is there: the CPU card ties `-CPUIN` (J3 B28) to its ground, the MMU / data card `-MMUIN` (J3 B30); the debug card pulls both up (10k), so a line is high only with that card out. GAL DBDRV, which makes every enable, reads them:

| Cards in | What "alone" may drive |
|---|---|
| neither | everything: `CLK`, `-CLK`, `-RES`, the requests, the grants, `-FAST`, D, the bus cycle, LR, J3; a GO cycle needs no grant |
| the MMU / data card | `CLK`, `-CLK`, `-RES`, `-BREQ3–10`, `-FAST`, LR and J3 (the CPU's side, to test that card). **Not** D, the bus cycle's lines or the grants: they are that card's (its address drivers and strobes are on the bus whenever no grant is out, and its arbiter makes the grants). A GO cycle drives nothing |
| the CPU card (with or without the other) | nothing. ALONE counts for nothing: no `CLK`, no `-RES`, no group |

A DMA cycle beside a running CPU (ALONE clear) is not touched by this: the arbiter's grant is its protection. The open-drain lines (`-ACK`, `-WAIT`, `-LOCK`, `-IRQ0–7`, `-BREQ0`) need none. Both lines are also sampled (slice 21), so the host can read which cards are in and say why a command did nothing.

- **The interrupt lines** (D101): a register of eight bits pulls any of `-IRQ0–7` low by open drain, in DRIVE, beside a running CPU or alone. In LISTEN the register is held cleared.
- **One bus cycle at a time:** the Pico loads a cycle (address, byte enables, read / write, data) into the registers and says "go". A GAL sequencer clocked by `CLK` requests the bus (`-BREQ0`), waits for the grant, runs **exactly one bus cycle** in step with `CLK`, honours `-WAIT`, latches D and `-ACK` at the cycle's end, and lets the bus go. The Pico reads the result. At USB speed that is a few hundred kilobytes a second: a kernel loads in well under a second.
- **Alone on the bus** (the CPU and MMU / data cards pulled): the card's oscillator, divided, makes `CLK` and `-CLK`; `-RES` is a command. The cycle sequencer then needs no grant. With a CPU card in, none of this happens (D102, above).
- **Clocking the machine (D76), in either mode:** CTRL1 bit 7 (XCLKEN) pulls the bus line `-XCLKEN` low, and the CPU card then takes the card's `XCLK` as its oscillator: GAL DBCLK's 12 MHz while CLKRUN (a 1.5 MHz machine), or **one edge per STEP command** -- one of the CPU's eight phases -- with nothing moving between. With LISTEN capturing at CLK, the host can step a program a phase or a bus cycle at a time and read every cycle back. `XCLK` and `-XCLKEN` are this card's own two lines, driven in LISTEN too (nothing else drives them); every other driver is off in LISTEN as before.
- **Testing the MMU / data card alone:** the Pico sets the J3 lines for one microcycle, pulses `CCLK` and `-WSTB` as the CPU card would, and reads back R, `FAULT`, `FC` and `STALL`. A microcycle at a time, so any TLB, fault or steering case can be checked.

## 5. The host link and the rest

- **A Raspberry Pi Pico 2** at the front edge, its USB port facing out (D48): 3.3 V, buffered to the card's 5 V by a 74LVC245; an 8-bit local bus with seven address lines and strobes to the card's registers.
- **A status display** (D100): two TIL311 hexadecimal digits at the front edge, just below the LEDs, showing a byte the Pico's firmware writes (local bus address `4D`). The debug card is the front card of the machine (slot 0, its components facing the front), so they are always in view. The digits stand upright with the card's front edge up, the high digit on the left.
- **Not built, and dropped (Ken, D101):** the card's own register window on the bus (an ID register and a mailbox for P32W software to talk to the Mac), and writing the boot ROM in place with a PROG jumper. The card is a tester only; it answers no bus address. The boot ROM's flash chips are programmed in a chip programmer (the memory card ties their write line high).

## 6. Size

**144 ICs** (22 capture latches, 22 capture SRAMs, 19 drive registers, the trigger RAMs, counters, six GALs, buffers, two hex displays) and the Pico, on a 9U card: about 3.2 A. It is the biggest card after the CPU card, and the one that makes every other card testable.

## 7. Decided (Ken, 2026-10-26)

1. **Capture depth: 32 K samples** in 22 × CY7C199 (15 ns), with a timing mode beside the normal one: one sample per bus cycle by default (about 11 ms of machine time), or four per microcycle for timing work (12 MHz: D73 halved the 24 MHz first decided).
2. **Both changes on the not-yet-built cards:** the backplane's eleventh position (slot 0: J1, J2 with the request / grant fan-in, J3), and the MMU / data card's arbiter taking slot 0 as a ninth requester at the highest priority.

## 8. As built

**The slices.** The 167 lines are grouped into **22 slices of eight**, each with a 74ACT574 sample latch, a CY7C199 capture SRAM and a 74HCT541 onto the local bus. The 19 slices of driven lines also get a 74HCT574 drive register:

| Slices | Lines | Driven as group |
|---|---|---|
| 0–3 | D0–D31 | GD (alone, both cards out), or a DMA write |
| 4–8 | A2–A31, BE0–3, `-MRD`, `-MWR`, `-IACK` | GBUS (alone, both cards out), or a DMA cycle |
| 9–10 | `-RES`, `-BREQ3–10`, CLK, `-CLK`, `-FAST` | GSYS (alone, no CPU card in; CLK from GAL DBCLK) |
| 11 | `-BGNT3–10` | GSYS too, but only with the MMU / data card out as well (D102) |
| 12–15 | LR0–LR31 | GLR (alone) |
| 16–18 | the CPU's J3 lines: RSRC, RDST, MEM, SIZE, SIGNED, SUPER, TLBW, CRSEL, `-WSTB`, LOCKREQ, PCINC, MARPC, MARINC, CCLK | GJ3 (alone) |
| 19–21 | `PWROK`, `-LOCK`, `-IRQ0–7`, its own `-BREQ` / `-BGNT`, `-ACK`, `-WAIT`, FAULT, FC0–3, STALL, `-CPUIN`, `-MMUIN` | no drive register; `-ACK`, `-WAIT`, `-LOCK`, `-BREQ` and `-IRQ0–7` driven by open drain |

**The GALs:** DBADR (the local bus's decoding), DBDRV (every driver's enable: none without DRIVE, and none against a card that is in, D102), DBSEQ (one DMA cycle on the bus clock), DBCAP (the capture: run, post-trigger, done; which of the latches or the SRAMs holds the slices' bus), DBTRIG (the trigger RAMs' patterns and the bus cycle), DBCLK (CLK and `-CLK` when alone, from the 24 MHz oscillator; and T12, that oscillator halved: the timing mode's 12 MHz, D73).

**The control registers** are 74HCT273s, cleared while the Pico holds `CRESN` low; a pull-down holds it low until the Pico's firmware runs. `CRESN` also clears the GO, ARM and STEP flip-flops, the DMA and capture state machines and the interrupt register (D101). So at power-on nothing is enabled and nothing can start, whichever way the mode switch stands.

**The local bus** (the Pico's GP0–18): data through a 74LVC245, seven address lines, `-XWR`, `-XRD`, the 245's direction, `CRESN`. Its register map is in `card_debug.py`'s opening comment.

**Verified at gate level** (`generators/sim_debug.py`, in `make test`), the Pico's firmware being a script of local-bus operations:

- **listen**: in slot 0 beside the CPU, MMU / data and memory cards, the card captures `hello` (one sample a CLK). After HALT its SRAMs are read back through the local bus, and every bus cycle captured is the cycle the bus carried (768 samples, 355 cycles); `-CPUIN` and `-MMUIN` are low in every sample (both cards are in);
- **dma**: DRIVE, with the CPU running a loop that polls a word. The card asks on `-BREQ0`, the arbiter grants slot 0 between the CPU's cycles, and the card writes the word; the program sees it and halts with it. Then the card reads the word back by DMA, and the snapshot of that cycle in the capture SRAMs holds its address, strobe, `-ACK` and data;

- **alone**: only the debug and memory cards. The card makes CLK itself (STEP), writes a word into RAM, and reads it back through its latches. Also: a byte written to the status display shows on both digits; the reset clears GO, ARM and STEP; the interrupt register pulls lines low in DRIVE and none in LISTEN; the system group drives a grant and `-FAST`. And the cards that are in (D102): with every group and ALONE set, `-MMUIN` pulled low turns off the D, bus cycle and grant enables and leaves the rest, and a GO then drives nothing; `-CPUIN` low turns off every enable and `CLK`; released, they come back;
- **xclk** (D76): the debug and CPU cards. With XCLKEN off the CPU card keeps its own oscillator; on, each host STEP advances the CPU's phase counter by exactly one (ten steps) and nothing moves between; with CLKRUN, one phase every two of the card's oscillator periods; and the CPU card's own CLOCK switch at STEP gives one phase per CLOCK STEP press.

**Not yet exercised:** the 12 MHz timing mode, the trigger RAMs' patterns, and playing the CPU's side of J3 against the MMU / data card alone.

# The debug card: theory of operation

docs/debug-card-theory.md

**Status:** DESCRIBES THE NETLIST AS BUILT (2026-10-30), `generators/card_debug.py` (every chip, net and GAL equation comes from there). The card's purpose and decisions are in the debug card; the bus cycle and slot 0 in the bus. What the gate-level simulation (`generators/sim_debug.py`) verifies is §14; what is unclear, §15. D48, D50, D64, D73, D76. **Brought up to D100 and D101 (2026-11-11):** a status display, two TIL311s written at `4D`; the mode switch turned so its lever points out past the front edge; the capture SRAMs written in the sample clock's high half (the write race gone); `CRESN` clearing GO, ARM, STEP and the two state machines; `-BREQ0` released at once in LISTEN; DRIVE reaching `-BGNT3–10`, `-FAST` and `-IRQ0–7`; the buffer's wrap readable.

## 1. What the card does

The debug card is the machine's bus tester. It sits in **slot 0**, which every backplane signal reaches, in one of two modes chosen by a switch at its front edge, never both at once (D64):

- **LISTEN**: a logic analyzer. It samples 167 lines (the bus, the system lines, J3, the CPU card's local link, and the two lines that say which of those cards are in) into 32 K samples, with a trigger. None of its bus drivers can turn on.
- **DRIVE**: a bus master. With the CPU and MMU / data cards in, it borrows the bus by DMA, one cycle at a time, and can pull any of `-IRQ0–7` low. **Alone** (the CPU card pulled) it makes `CLK` and `-CLK` and may drive the bus cycle, `-RES`, `-BREQ3–10`, `-BGNT3–10`, `-FAST` and the CPU card's side of J3. **Never against a card that is in** (D102): with a CPU card in, "alone" does nothing; with an MMU / data card in, that card's lines are left to it (§9).

In either mode it can **clock the CPU card** over its own two bus lines, `XCLK` and `-XCLKEN` (D76). A **Raspberry Pi Pico 2** is the host link: USB to the Mac, an 8-bit local bus to the registers. **Its firmware, USB command set and Mac-side tools are not written yet**: this chapter gives the register interface, and §12 walks a host through it.

## 2. Slot 0

Slot 0 is an eleventh backplane position beside slot 1 (D64). It carries 169 signals (`p32bus.busnet()`): J1's 70 (the bus cycle); J2's 35 (`CLK`, `-CLK`, `-RES`, `PWROK`, `-LOCK`, `-IRQ0–7`, `-IACK`, `-FAST`, `XCLK`, `-XCLKEN`, every requester's `-BREQ3–10` / `-BGNT3–10` as the arbiter in slot 2 sees them, and the card's own pair); J3's 64 (LR0–31, the CPU card's 24 control lines, `FAULT`, `FC0–3`, `STALL`, and since D102 `-CPUIN` and `-MMUIN`). The card samples 167: all but `XCLK` and `-XCLKEN`, which are its own outputs.

`-CPUIN` **and** `-MMUIN` (J3 B28, B30; D102) say which cards are in. They are two of the J3 pins that were spare, which the backplane joins between slots 0, 1 and 2 (D99). The CPU card ties its `-CPUIN` pin straight to its ground plane, the MMU / data card its `-MMUIN` pin: no part, nothing to fit or to forget. On the debug card each has a 10k pull-up (R6, R7), so a line is high only when that card is out of its slot. §9 has what they do.

**The card's own pair**, its `-BREQ` / `-BGNT` pins (J2 A16, A17), is in slot 0 the backplane's `-BREQ0` / `-BGNT0`: the first priority of GAL ARB, the arbiter on the MMU / data card (mmu-card-theory §9). This chapter uses the backplane's names.

## 3. The host link

The Pico (A1) sits at the front edge, its USB port facing out (D48). It runs from the backplane's 5 V through a 1N5817 Schottky into its VSYS; its 3.3 V output supplies the 74LVC245 and two pull-ups.

| Pico | Net | What |
|---|---|---|
| GP0–7 | PD0–7 | data, through a **74LVC245** to the card's 5 V `XD0–7` |
| GP8–14 | `XA0–6` | the register address |
| GP15, GP16 | `-XWR`, `-XRD` | the write and read strobes |
| GP17 | `XDIR` | high: the Pico drives `XD`; low: `XD` drives the Pico |
| GP18 | `CRESN` | low: the control registers held clear |

The 245 is always enabled: `XDIR` alone decides who drives `XD`. The address and strobes go straight to the decoders. **Until the firmware runs**, 10k pull-ups hold `-XWR` and `-XRD` high and a 10k pull-down holds `CRESN` low, so the control registers, the GO, ARM and STEP flip-flops and the two state machines stay clear whichever way the mode switch stands (§5). **A write:** address and data out, `XDIR` high, `-XWR` low, then high; registers take `XD` at the strobe's rising edge, and the GO, ARM and STEP flip-flops are preset while it is low. **A read:** `XDIR` low, `-XRD` low; a 74HCT541 drives `XD`. sim_debug's `Pico` holds each strobe low for two periods of the card's 24 MHz oscillator.

## 4. The local bus decode

**GAL DBADR** decodes `XA6–XA3` and the strobes into seven decoder enables: `SWLON` / `SWHIN` (a write with `XA6`, `XA5` low; `XA4` picks the 74HCT154 for slices 0–15 or 16–21), `SRLON` / `SRHIN` (the same for a read), `CW0N` / `CW1N` (a write with `XA6` high; `XA3` picks the 74HCT138 for `40–47` or `48–4E`) and `CRN` (a read with `XA6` high: the 74HCT138 for `40–42`). The 154s decode `XA3–XA0`, the 138s `XA2–XA0`. The map (every local-bus address and value here is hex):

| XA | A write | A read |
|---|---|---|
| `00–12` | slice 0–18's drive register (§7) | slice 0–18 (§7) |
| `13–15` | nothing (no drive register) | slice 19–21 |
| `40` | **CTRL0** | **status** |
| `41` | **CTRL1** | the capture address CA, bits 7–0 |
| `42` | the post-trigger count, bits 7–0 | CA bits 14–8; bit 7 is CA15: the buffer has wrapped (D101) |
| `43` | the post-trigger count, bits 15–8 | — |
| `44` | **GO**: one DMA cycle | — |
| `45` | **ARM**: a capture | — |
| `46` | **STEP**: one `CLK` period alone, or one `XCLK` pulse | — |
| `47` | **CSTEP**: CA + 1 | — |
| `48`, `49` | the trigger setup address TS, bits 7–0, 14–8 | as `40`, `41` |
| `4A`, `4B` | trigger RAM A, B at TS | `4A` as `42` |
| `4C` | **CAPCLR**: CA = 0 | — |
| `4D` | **the status display** (D100): two hex digits, bits 7–4 and 3–0 | — |
| `4E` | **the interrupt lines** (D101): bit n = 1 pulls `-IRQn` low (§9) | — |

GO, ARM, STEP, CSTEP and CAPCLR act by their strobe; their data is not used.

**The status display** (D100; Ken: for status information, always visible, since the debug card is the front card of the machine). Two **TIL311** hexadecimal displays, U143 (the high digit, `XD7–4`) and U144 (the low one, `XD3–0`), stand at the front edge just below the LEDs, on 0.45 in centres. A TIL311 holds its own 4-bit latch, decoder and LED drivers: its latch follows its data inputs while its strobe is low and holds them when it is high. The strobe of both is `DISPN`, the `48–4F` decoder's output for `4D`, so a write to `4D` shows its byte and the digits keep it; no register chip is needed. Their blanking inputs are tied low and their decimal points are not connected. They are never cleared: from power-on until the Pico's firmware first writes `4D` they show what their latches woke up with. What the byte means is the firmware's to define. Each draws about 105 mA (180 mA at most), 0.2 A of the card's 3.2 A. The data must be steady 50 ns before the write ends and 40 ns after it (TI SODS001D): the firmware's concern, as for the other registers.

Above `3F` the writes ignore `XA5–XA4` and the reads `XA5–XA3`, so the registers repeat; nothing answers at `16–3F`.

## 5. The control and status registers

Six 74HCT273s hold CTRL0, CTRL1, the post-trigger count (POSTL, POSTH) and TS (TSAL, TSAH). `CRESN` low clears them. They cannot be read back, so the host keeps copies.

| Bit | CTRL0 (`40`) | CTRL1 (`41`) |
|---|---|---|
| 0 | GD: alone, D0–31's drive registers on | ODLOCK: `-LOCK` low |
| 1 | GBUS: alone, the bus cycle's | CLKRUN: DBCLK runs (`CLK` alone; `XCLK` with XCLKEN) |
| 2 | GSYS: alone, `-RES`, `-BREQ3–10`, `-FAST` and the grants | SCLKSEL: the sample clock is 12 MHz, not `CLK` |
| 3 | GLR: alone, LR0–31 | SETUP: the trigger RAMs being written; no trigger |
| 4 | GJ3: alone, the CPU card's J3 control lines | TNOW: trigger now |
| 5 | ALONE: the card drives `CLK`, `-CLK`; the groups may drive; GO needs no grant | LIVE: the slices read the latches, not the SRAMs |
| 6 | ODACK: `-ACK` low | DMAWR: a DMA cycle drives D (a write) |
| 7 | ODWAIT: `-WAIT` low | XCLKEN: `-XCLKEN` low, the CPU card on `XCLK` (D76) |

ALONE and the five group bits are what the host asks for. GAL DBDRV gives a group its drivers only if the cards whose lines they are are out (§9, D102). ODACK, ODWAIT and ODLOCK act only in DRIVE. **Three flip-flops** are set by the host and cleared by the card: **GOQ** (74HCT74; GAL DBSEQ clears it at SNAP), **ARMQ** (74HCT74; GAL DBCAP, once the capture runs) and **STEPQ** (74ACT74, D73; GAL DBCLK, when the step is done).

**The reset** (D101). `CRESN` low clears, besides the six registers: GOQ, ARMQ and STEPQ (each GAL's clear line takes `CRESN`), GAL DBSEQ's REQ, CYC and SNAP and GAL DBCAP's RUN, POST and DONE (the 22V10s' asynchronous reset), and the interrupt register (§9). So with no firmware running, or while it holds the card in reset, no cycle can start and no line is pulled, whichever way the switch stands. Not cleared, and harmless until the host loads them: the drive registers (their groups are off), the trigger RAMs and the status display's latches.

| Status (`40`) bit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Signal | DRIVE | `-BGNT0` | GOQ | ACT | DONE | ARMQ | TRIG | SNAP |

DRIVE: the switch at DRIVE. `-BGNT0` low: slot 0 granted. GOQ: a DMA cycle not yet done. ACT: a capture running. DONE: it has finished. ARMQ: armed, not yet running. TRIG: the trigger condition now. SNAP: the clock after a DMA cycle.

## 6. LISTEN and DRIVE

The **mode switch** S1 (E-Switch's EG1224, a right-angle SPDT slide switch at the front edge: Ken, D101) is turned so that its lever points out past the card's edge (D100): about 4 mm of it stands beyond the edge. It is labelled MODE, with LISTEN at one end of the lever's travel and DRIVE at the other. It ties the net `DRIVE` to 5 V (DRIVE) or to ground. `DRIVE` gates every driver: **GAL DBDRV** makes every drive register's output enable and `CLK`'s, none without `DRIVE`, and none against a card that is in its slot (§9); a **74HCT08** passes ODACK, ODWAIT and ODLOCK on only with `DRIVE`, and holds the interrupt register cleared without it; **GAL DBSEQ** asks for the bus only with it, and lets `-BREQ0` go the moment it is gone (§10, D101); and **GAL DBCAP** captures only *without* it (§8). So LISTEN captures and drives nothing; DRIVE drives, and keeps only its DMA cycles' snapshots. `XCLK` and `-XCLKEN`, the card's own lines, are driven in both modes (D76).

A 74HC05 lights the mode LEDs through 470 Ω each: **DRIVE** (red) from `DRIVE`, **LISTEN** (green) from DBDRV's `LISTEN` (`DRIVE` inverted). The green POWER LED is every card's (D76).

## 7. The slices

The 167 lines are cut into **22 slices of eight** (spare inputs grounded). Each slice has a **74ACT574 sample latch** clocked by the sample clock `SCLK`, a **CY7C199 capture SRAM** (32 K × 8, 15 ns) on the same 8-bit bus, addressed by CA14–0, and a **74HCT541** from that bus onto `XD` for the host's reads. Slices 0–18 also have a **74HCT574 drive register**, written from `XD` at the slice's address and driving its lines while its group is enabled.

| Slice | Its lines, bit 0 first | Group |
|---|---|---|
| 0–3 | D0–7, D8–15, D16–23, D24–31 | D |
| 4–6 | A2–9, A10–17, A18–25 | BUS |
| 7 | A26–31, BE0, BE1 | BUS |
| 8 | BE2, BE3, `-MRD`, `-MWR`, `-IACK` | BUS |
| 9 | `-RES`, `-BREQ3–9` | SYS |
| 10 | `-BREQ10`, `CLK`, `-CLK`, `-FAST` | SYS (not `CLK`, `-CLK`) |
| 11 | `-BGNT3–10` | GNT: the SYS group's bit, its own enable (§9) |
| 12–15 | LR0–7, LR8–15, LR16–23, LR24–31 | LR |
| 16 | RSRC0–3, RDST0–2, MEM0 | J3 |
| 17 | MEM1, MEM2, SIZE0, SIZE1, SIGNED, SUPER, TLBW, CRSEL0 | J3 |
| 18 | CRSEL1, CRSEL2, `-WSTB`, LOCKREQ, PCINC, MARPC, MARINC, CCLK | J3 |
| 19 | `PWROK`, `-LOCK`, `-IRQ0–5` | — |
| 20 | `-IRQ6`, `-IRQ7`, `-BREQ0`, `-BGNT0`, `-ACK`, `-WAIT`, `FAULT`, `FC0` | — |
| 21 | `FC1–3`, `STALL`, `-CPUIN`, `-MMUIN` | — |

A line's bit is its place in the row: in slice 8 `-IACK` is bit 4, in slice 20 `-ACK` is bit 4. (D101 moved the grants into the SYS group and added `-FAST`, so the slices from 11 on are one higher than they were. D102 gave the grants slice 11 to themselves, `-FAST` moving to slice 10, and put `-CPUIN` and `-MMUIN` in slice 21: bits 4 and 5, a 0 for a card that is in.) Slices 19–21 are listened to only; `-ACK`, `-WAIT`, `-LOCK`, `-BREQ0` and `-IRQ0–7` have open-drain drivers of their own (§9), and `CLK`, `-CLK` come from GAL DBCLK. **A slice's bus** is driven by its latch while LIVE is set, a capture runs or its last write is still to come (WQ, §8), or during SNAP; otherwise by its SRAM, showing the sample at CA (DBCAP's `LATOEN`, `SROEN`).

## 8. LISTEN: the capture

**The sample clock.** A 74ACT157 makes `SCLK` from `CLK` (SCLKSEL = 0) or **T12**, GAL DBCLK's register toggling at every edge of the card's 24 MHz oscillator: 12 MHz (D73). At `CLK`'s rise a sample is the state at a bus cycle's end (`CCLK` rises at the same edge); 32 K are about 11 ms at 3 MHz (debug-card.md §7). At 12 MHz there are four per microcycle, taken asynchronously like a logic analyzer's; 32 K are about 2.7 ms (D73).

**The write** (D101). A sample is latched at `SCLK`'s rise and written in the **high half** that follows: a 74ACT00 makes the SRAMs' `-WE` = NAND (`WEX`, `SCLK`). So the write ends at `SCLK`'s fall, with the data steady since just after the rise and for half a period more, until the latches next change. (Before D101 the write was in the low half and ended with the next rise, two gate delays after the edge that changed the latches: a race the SRAMs' 9 ns of data set-up lost, §15.) Which high halves write is `WEX`, from GAL DBCAP and **WQ**. WQ is DBCAP's `SWEN` taken by a 74ACT74 at `SCLK`'s fall (its clock is `SCLK` through a 74ACT04), so it changes only while `SCLK` is low and `-WE` cannot glitch:

- listening, `WEX` = WQ: a sample is written when the capture was active half a period earlier. The first sample written is the one latched at the edge after RUN begins, the last the one latched at the edge that ends the capture;
- in DRIVE, `WEX` = SNAP and not WQ: the one high half after the edge that ended a DMA cycle, on the sample that edge latched (§10).

**The capture address.** Four 74F161s count CA; bits 14–0 address the SRAMs, so the buffer is a circle of 32 768 samples, and bit 15, read as bit 7 of `42`, says it has gone round (D101). The counter's clock is `-WE`'s own rising edge, two gates later (a 74ACT00 and a 74ACT04): **CA moves on by one after every sample written**, never before the write has ended; or it is a CSTEP's end (a 74HCT32 lets CSTEP through only when no capture runs). CAPCLR clears CA at once (the 74F161's clear is asynchronous); nothing else sets it.

**GAL DBCAP**, clocked by `SCLK`, runs the capture:

1. ARM sets ARMQ. While it is set and nothing runs, `PLDN` has the post-trigger counter load.
2. At the next sample edge, in LISTEN: **RUN** (ACT). `ARMCLRN` clears ARMQ; DONE clears. At `SCLK`'s fall WQ sets.
3. At every later sample edge the latches take the lines, and in the high half the SRAMs write them at CA, which then steps by one.
4. At the sample edge that finds `TRIG` high: **POST**. The post-trigger counter counts at each edge while POST (its enable is DBCAP's POST pin).
5. At a sample edge in POST that finds its carry `PDONE` high: **DONE**, until the next ARM. That edge's sample is still written (WQ), and CA then points one past the last address written.

So with a post-trigger count of n, the sample the trigger's edge latched is at CA − n − 1, and n samples follow it. RUN and POST need LISTEN: moving the switch to DRIVE abandons a capture without DONE. `CRESN` low clears RUN, POST and DONE (§5). In DRIVE, the one write is SNAP's (§10).

**The post-trigger count.** Four more 74F161s on `SCLK`, loaded from POSTH:POSTL; `PDONE`, their carry, is high at `FFFF`. Written as 65 536 − n, the count brings DONE n sample edges after the edge that saw the trigger: `FFFF` is n = 1, `FF00` is 256. A count beyond the buffer overwrites the trigger's own sample.

**The trigger.** Two more CY7C199s are the **trigger RAMs**: A is addressed by A16–A2, B by A31–A17, and bit i of each belongs to **pattern i**, 0–7. Pattern i matches when A's bit i at the cycle's A16–A2 and B's bit i at its A31–A17 are both 1 (two 74HCT08s make `M0–M7`). A pattern is thus any set of low halves with any set of high halves: one word, a mask (a don't-care bit set at both values), a range within one half. **GAL DBTRIG** makes `TRIG`: never during SETUP; otherwise TNOW, or a bus cycle (`-MRD`, `-MWR` or `-IACK` low) whose address any pattern matches. Reads and writes are alike to it.

**Setting the trigger up** (SETUP = 1): eight 74HCT157s switch both RAMs' address inputs from the bus to TS14–0; the RAMs' outputs turn off (`-OE` is SETUP); two 74HCT541s, enabled by DBTRIG's `SETUPN`, put `XD` on their data pins; a write to `4A` or `4B` strobes RAM A's or B's `-WE`. TS is a register, not a counter. Nothing clears the RAMs: at power-on they hold anything (the simulated ones read 0).

**Reading back.** After DONE, `41` and `42` give the final CA. For each sample the host reads the slices it wants (LIVE off), then CSTEPs; once the buffer has wrapped, starting at the final CA gives the samples oldest first. Read-back and trigger setup happen only while nothing is captured (timing.md §3); during a capture a slice read shows the latest sample.

## 9. DRIVE: the drivers

**GAL DBDRV** makes the output enables, each needing `DRIVE`. Two of its inputs are the backplane's `-CPUIN` and `-MMUIN` (§2), and it writes "alone" in two strengths (D102):

- **alone** = `DRIVE` and ALONE and `-CPUIN` high: the host asks, and no CPU card is in;
- **the bus is the card's** = alone and `-MMUIN` high: no MMU / data card either.

| Group | Slices | Its enable | On when |
|---|---|---|---|
| D | 0–3 | `DOEN` | the bus is the card's, and GD; or a DMA cycle with DMAWR |
| BUS | 4–8 | `BUSOEN` | the bus is the card's, and GBUS; or a DMA cycle |
| SYS | 9–10 | `SYSOEN` | alone and GSYS |
| GNT | 11 | `GNTOEN` | the bus is the card's, and GSYS |
| LR, J3 | 12–15, 16–18 | `LROEN`, `J3OEN` | alone and GLR, GJ3 |
| `CLK`, `-CLK` | (GAL DBCLK) | `CLKOE` | alone |

"A DMA cycle" is GAL DBSEQ's `BUSEN` (§10), and it counts in two cases only: ALONE clear, when the cycle began on the arbiter's grant; or the bus the card's. The drive registers are not cleared at power-on: the host loads them before enabling their group.

**Why each card takes what it takes.** The CPU card drives `CLK`, `-CLK` and `-RES` and its side of the local link at all times, so with it in there is nothing "alone" could drive without a second driver on a line: ALONE counts for nothing, the clock included. The MMU / data card is the CPU's way onto the bus: its address drivers and strobes are on the bus whenever no grant is out (mmu-card-theory §9), it drives D in a write, and its arbiter makes every grant. So with it in the card leaves D, the bus cycle's lines and the grants alone, and a GO that no arbiter granted drives nothing (GAL DBSEQ still runs its cycle, and GO clears). What is left is what testing that card needs: the clock, `-RES`, the other masters' `-BREQ3–10` to its arbiter, and the CPU card's side of LR and J3 (debug-card.md §1).

Before D102 nothing but the host's ALONE bit stood between the card and a CPU card that was in: a wrong command, or a firmware fault, put two drivers on `CLK`. The lines are tied to ground on the cards themselves, so the check needs no firmware and cannot be left out of a build. A broken line reads as "card out": it is a guard against a mistake at the keyboard, not against a faulty backplane.

**The SYS group also has `-FAST`, and the grants ride on its bit** (D101): with the MMU / data card out too, nothing else drives the grants, and the card can hand a device's DMA engine the bus (a disk card's, D75) and watch what it does; `-FAST` low sets the I/O card's timer for the 4 MHz clock (D74). `-FAST` is a jumper to ground on the CPU card, which is out when the card is alone.

**The open-drain lines** come from 74HC05s: `-ACK`, `-WAIT` and `-LOCK` from ODACK, ODWAIT and ODLOCK (through the 74HCT08: only in DRIVE), and `-BREQ0` from GAL DBSEQ.

**The interrupt lines** (D101). A seventh 74HCT273, written at `4E`, has a bit for each of `-IRQ0–7`: a 1 pulls that line low through a 74HC05 (two of them), as a device would; the backplane's 2.2k pull-ups hold the rest high. It needs DRIVE but not ALONE, so it works beside a running CPU: the way to test the CPU card's interrupt logic, or a handler, without the device. The register's clear is `CRESN` AND `DRIVE` (the 74HCT08's fourth gate), so it is empty after a reset and **whenever the switch is at LISTEN**: in LISTEN no line is pulled, by wiring, and back in DRIVE the lines stay released until `4E` is written again.

## 10. The DMA cycle: GAL DBSEQ

**GAL DBSEQ**, clocked by `CLK`, runs one bus cycle per GO, from GOQ, `-BGNT0`, `-WAIT`, ALONE and DRIVE:

- **REQ** (DRIVE, not alone): set at a `CLK` edge when GO is pending and nothing is under way; held until the grant;
- **CYC**, the cycle: set at the edge that finds REQ and the grant (alone: at the first edge after GO, with no request); held while `-WAIT` is low;
- **SNAP**, the clock after: set at the edge that ends CYC.

Its outputs: `BUSEN` = CYC; `BREQOD` = DRIVE and (REQ or CYC) (`-BREQ0` low: it goes the moment the switch leaves DRIVE, not at the next `CLK`, D101); `GOCLRN` = not SNAP, and `CRESN` (GO cleared). `CRESN` low clears REQ, CYC and SNAP. It does not look at `-ACK`.

**A DMA cycle, edge by edge** (the CPU, MMU / data and memory cards in, no wait; sim_debug's dma case runs it):

| `CLK` edge | What happens |
|---|---|
| (any time) | the host writes GO: GOQ (status bit 2) |
| 1 | REQ: `-BREQ0` low |
| 2 | the arbiter, the bus free, grants: `-BGNT0` low (bit 1); the MMU / data card's drivers are off while the grant is out (mmu-card-theory §9) |
| 3 | CYC: the card drives A2–31, BE0–3 and the strobe from its registers, and D for a write |
| 4 | the cycle ends (a slave's `-WAIT` would hold it): a write is taken; the latches sample the lines; SNAP: GO clears, `-BREQ0` released, drivers off |
| 5 | SNAP ends; the arbiter drops `-BGNT0` |

GO clears three clocks after edge 1: three to four clocks from the host's GO, about 1–1.3 µs at 3 MHz. Another master or a held `-LOCK` delays the grant; the CPU is held (`STALL`) only if it wants the bus meanwhile (bus.md).

**SNAP** keeps the result (D101): in the high half that opens it, the SRAMs write what the latches took at the edge that ended the cycle, at CA, which then moves on by one, so a run of DMA cycles leaves its snapshots at consecutive addresses. During SNAP the slices show the latches; after it, the SRAMs (at the new CA: CAPCLR before the cycle and again after it puts the snapshot under the slices). The word read is in slices 0–3, `-ACK` in slice 20 bit 4 (0: a card answered). The snapshot is the cycle's end only with the sample clock at `CLK`. sim_debug's dma case reads one back.

## 11. The clock: GAL DBCLK

`CLK` **alone.** GAL DBCLK is clocked by the card's 24 MHz oscillator. Its phase counter, P2–P0, counts the oscillator's edges while CLKRUN or STEPQ is set. With `CLKOE` (DBDRV: DRIVE and ALONE) it drives `CLK` high in phases 0–3 and `-CLK` in 4–7: the oscillator divided by eight, 3 MHz, as the CPU card makes it (cpu-card.md §1).

**STEP** presets STEPQ; the counter runs until phase 3, where `STEPCLRN` clears STEPQ. From phase 3 a STEP is one whole period: `CLK` falls at phase 4, rises at phase 0, and stands high at phase 3. The first STEP after power-on only takes the counter from 0 to 3, with no rising edge (sim_debug's alone case steps once first). At phase 3 `STEPCLRN` goes low as soon as STEPQ is set, so the host's strobe, which holds the preset, must span an oscillator edge (the simulation's spans two).

`XCLK` (D76). `-XCLKEN` is XCLKEN inverted; low, it makes the CPU card's GAL CKSEL take `XCLK` as its oscillator, ahead of that card's CLOCK switch and oscillators (cpu-card.md §1). `XCLK` is DBCLK's register XCK: with XCLKEN and CLKRUN it toggles at every oscillator edge (12 MHz, a 1.5 MHz machine); with XCLKEN and a STEP it is high for one oscillator period, one edge for the CPU card, one of its eight phases (XDN marks the pulse given while the host's strobe still holds STEPQ); without XCLKEN it stays low. `CRESN` low clears XCLKEN, so with no firmware running the CPU card keeps its own oscillator. **Change XCLKEN with RESET held** (D76): CKSEL is a plain multiplexer, and a change mid-phase can cut a phase short.

**T12** toggles at every oscillator edge, whatever else is set: the 12 MHz sample clock.

## 12. Using it

There is no firmware yet: each step below is one local-bus write (`XA` ← value) or read. Sequences marked *tested* are what sim_debug.py's Pico script does in `make test`; the rest follow from the netlist.

**Start-up.** `-XWR`, `-XRD` high, `XDIR` low, GP0–7 inputs; then raise `CRESN` (GP18). Every control register starts at 0.

**(a) A capture with a trigger.** The switch at LISTEN. Trigger on any bus cycle at the word `0x0018_0000` (pattern 0) and keep 256 samples from it:

1. `41` ← `08` (SETUP).
2. For t = `0000` to `7FFF`: `48` ← t's bits 7–0, `49` ← its bits 14–8 (when they change), `4A` ← RAM A's byte, `4B` ← RAM B's. Here A's byte is `01` at t = `0000` (the word's A16–A2), B's `01` at t = `000C` (its A31–A17), and both `00` elsewhere.
3. `41` ← `00` (SETUP off; the sample clock `CLK`, or `04` for 12 MHz).
4. `42` ← `00`, `43` ← `FF` (65 536 − 256). `4C` (CA = 0). `45` (ARM).
5. Read `40` until bit 4 (DONE). Without a trigger: `41` ← `10` (TNOW), wait for DONE, `41` ← `00`.
6. Read `41`, `42` (the final CA); `4C`; then for each address read the slices wanted among `00–15` and write `47` (CSTEP).

By §8 the trigger's sample lies 257 addresses before the final CA (CA − n − 1), with the 256 after it. *Tested* without the trigger RAMs: sim_debug's listen case does steps 3–6 with the count `FFFF`, ending by TNOW after HALT and reading samples 0 to CA − 1.

**(b) A write and a read as bus master.** The switch at DRIVE, CTRL0 = `00`, the CPU and MMU / data cards in. The word `5A5A1234` to `0x0018_0000` by DMA (*tested*: sim_debug's dma case):

1. `41` ← `40` (DMAWR; the sample clock `CLK`, LIVE off).

2. `00–03` ← `34 12 5A 5A` (D); `04–07` ← `00 00 06 C0` (A2–A31, BE0, BE1); `08` ← `17` (BE2, BE3; `-MRD` 1, `-MWR` 0, `-IACK` 1).
3. `44` (GO); read `40` until bit 2 (GOQ) is 0.

Reading it back by DMA (*tested*: the dma case, after the program has halted): `41` ← `00`; `4C` (CA = 0); `04–07` as before, `08` ← `1B` (`-MRD` 0, `-MWR` 1); `44`; read `40` until bit 2 is 0; `4C` again (the snapshot is at 0, CA now 1); read `00–03` (the word) and `14` (bit 4: `-ACK`).

Alone, with only a memory card besides, `C0FFEE42` to `0x0010_0040` and back (*tested*: sim_debug's alone case):

1. `40` ← `20` (ALONE: the card drives `CLK`); `46` (STEP) once.
2. `00–03` ← `42 EE FF C0`; `04–08` ← `10 00 04 C0 17`; `09` ← `FF`, `0A` ← `09`, `0B` ← `FF` (`-RES`, `-BREQ3–10`, `-FAST` and `-BGNT3–10` high). `40` ← `27` (GD, GBUS, GSYS, ALONE) puts the write on the bus; `46` gives it one `CLK` period (the memory card writes while `CLK` is low, mem-card-theory §5); `40` ← `20`.
3. `41` ← `20` (LIVE); `08` ← `1B`; `40` ← `26` (GBUS, GSYS, ALONE); `46`. The latches took D at `CLK`'s rise and the clock stands still: read `00–03`.

**(d) An interrupt by hand** (*tested* as far as the lines: the alone case). The switch at DRIVE: `4E` ← `04` pulls `-IRQ2` low, `4E` ← `00` lets it go. Moving the switch to LISTEN lets every line go at once.

**(e) Which cards are in** (D102; *tested*: the alone and listen cases). `41` ← `20` (LIVE) and a `CLK` edge, then read `15` (slice 21): bit 4 is `-CPUIN`, bit 5 `-MMUIN`, a 0 for a card that is in. Every captured sample has them too. A host should read them before it sets ALONE, and tell its user when a card that is in makes a command do nothing: the card itself refuses silently.

**(c) Stepping the machine's clock.** The CPU card in, the switch either way (*tested*: sim_debug's xclk case, in LISTEN):

1. Holding the CPU card's RESET, `41` ← `80` (XCLKEN): its clock stops. Release RESET.
2. `46` (STEP): one `XCLK` pulse, one phase of eight, nothing moving between. Eight STEPs are one microcycle, one bus clock.
3. `41` ← `82` (XCLKEN, CLKRUN) runs the machine at 1.5 MHz; `80` stops it.
4. Back to the CPU card's oscillator: hold RESET, `41` ← `00`.

To watch it (not tested): arm a capture as in (a), keeping XCLKEN in every CTRL1 write: one sample per eight STEPs, and slice reads show the latest. Or `41` ← `A4` (XCLKEN, LIVE, SCLKSEL): the latches follow the lines at 12 MHz, so between steps the slices show them as they stand.

## 13. Timing

The timing pass (D71, D73; timing.md §1) times the card alone in its 12 MHz mode: every path meets the oscillator's 41.7 ns, at 41.0 ns worst case (26.5 typical). It takes the sample latches as asynchronous, leaves the read-back and the trigger setup untimed (only while stopped) and checks no hold times (timing.md §3, §5); the card is also timed in the whole machine at 3 MHz. The local bus is not timed: nothing yet specifies the host's strobe widths or read access time. `-BREQ0`, released at SNAP, rises through its 10k pull-up in about 255 ns (timing.md §4), before the arbiter's next look.

**The write, by hand** (D101; the pass checks set-up, not the order of edges). `-WE` falls a 74ACT00's delay after `SCLK` rises and rises one after it falls. The data: the 74ACT574s' outputs are steady 10.5 ns after the rise at most, 31 ns before `-WE` rises in the 12 MHz mode (the SRAMs want 9), and stay until the next rise, half a period after it. The address: CA

changes two gates and a 74F161 after `-WE` has risen, by construction, and is steady 9 ns or more before `-WE` next falls at 12 MHz. WQ and `WEX` change only in `SCLK`'s low half, 13 ns or more before it ends at 12 MHz, so `-WE` has no short pulses.

## 14. Verified

`card_debug.py` checks that every pin is connected or declared unused, and each GAL's equations against its spec for every input (`make test`, run_tests.sh 6c6); `p32jed.py` checks the six JEDEC files in `hardware/gal/debug/` through their fuse maps (6c8). At gate level, `sim_debug.py` in `make test`, the Pico a script of local-bus operations:

- **listen**: `hello` beside the CPU, MMU / data and memory cards, one sample a `CLK`; read back after HALT, the samples hold every bus cycle from the capture's start to HALT as the bus carried it (768 samples, 355 cycles), and `-CPUIN` and `-MMUIN` low in every one (D102);
- **dma**: one DMA write while the CPU polls the word (`dma_test.asm`); the program halts with `5A5A1234`; then a DMA read of the word, whose snapshot in the capture SRAMs holds the cycle as it ended: its address, `-MRD` and `-ACK` low, the data, with CA one further (D101);
- **alone**: the debug and memory cards, ALONE and STEP: a word written and read back through the latches; and (D100, D101) the status display showing a byte written to `4D`; `CRESN` clearing GO, ARM and STEP; `4E` pulling interrupt lines low in DRIVE and none in LISTEN; the SYS group driving `-BGNT5` and `-FAST` low and the other grants high; and (D102) the cards that are in: a GO alone turns the bus cycle's drivers on; with every group and ALONE set and `-MMUIN` held low, the enables of D, the bus cycle and the grants are off, the grants float, `-FAST` and `CLK` are still driven, and the same GO turns nothing on; with `-CPUIN` low every enable is off and `CLK` floats; let go, all of them return;
- **xclk** (D76): the debug and CPU cards: XCLKEN off, the CPU card on its own oscillator; on, ten STEPs advance its phase by exactly ten; CLKRUN, a phase per two of the card's oscillator periods; the CPU card's CLOCK STEP button, a phase a press.

**Not exercised:** the trigger RAMs' patterns (the simulated RAMs read 0, so none matches), a post-trigger count other than 1, the 12 MHz mode, CLKRUN alone, what a GO alone puts on the bus (only that its drivers come on), `-WAIT` in a DMA cycle, ODACK / ODWAIT / ODLOCK, the buffer's wrap, and the J3 and LR groups.

## 15. Open questions

Settled by D102 (2026-11-11): the grants beside the arbiter, and with them every line the card alone could have driven against a card that was in (§9). Settled by D100 and D101 (2026-11-11; they were items 1–4 and 6–9 here): the SRAMs' write race (§8), the power-on state (§5), `-BREQ0` in LISTEN (§10), the lines DRIVE did not reach (§9), the buffer's wrap (§8), the stale comments, debug-card.md's drift from the netlist, and the mode switch's part (a slide switch, E-Switch EG1224).

1. **RESET for XCLKEN** (D76): with the CPU card in, the card cannot drive `-RES` (GSYS needs ALONE, and since D102 ALONE does nothing with a CPU card in), so someone holds the CPU card's RESET button. Intended?
2. **Which end of S1 is LISTEN.** The silkscreen puts LISTEN at pin 1's end of the lever's travel (pin 1 is the grounded one) and DRIVE at pin 3's, taking the usual slide switch, whose common joins the end the lever stands at. E-Switch's drawing could not be fetched: to check on the part with a meter before the board is ordered. (The LEDs show the true mode either way.)

3. **The firmware.** The Pico's firmware, its USB commands and the Mac's tool are not written; nothing yet specifies the host's strobe widths. The trigger RAMs hold anything at power-on: the firmware writes all of both before the first armed capture.
4. **The mode LEDs' colours** are only in their function text (DRIVE red, LISTEN green); the parts list says "LED 3mm".

# The backplane

docs/backplane.md

**Status:** NETLIST DONE (2026-10-26, D68, accepted by Ken 2026-10-27), checked against the bus definition and against every card's netlist in every slot it may use; laid out and routed by its generator (D79, §4: 0 DRC violations, 0 unconnected). **Changed 2026-11-11 (D99, Ken): ten slots, 1.5 inches apart, every part and the power entry at the back, 14 mm taller than the cards at the top and at the bottom with the edge mounting holes there, the slots labelled.** Not ordered. The netlist is `generators/card_backplane.py`, the board `generators/p32backplane.py`; the bus itself, pin by pin, is bus.md (`generators/p32bus.py`).

**30 sockets, 122 parts** (`python3 generators/card_backplane.py --bom`).

## 1. What it holds

**Ten positions, 1.5 inches (38.1 mm) apart** (D99; eleven at 1 inch before, D49, D64): the debug slot 0, then slots 1–9. Each has three female DIN 41612 C 3 × 32 sockets, vertical, through-hole: **S*n*J1, S*n*J2, S*n*J3**, mating the card's J1–J3.

**Front and back (D99).** Slot 0's end is the **front**; the other end, past slot 9, is the **back**, and holds every part that is not a socket or its bypass, and both power entries. The one exception is the +5V LED with its resistor, at the front edge where it can be seen.

**Taller than the cards (D99).** A card's connector edge is 366.7 mm; the board is **394.7 mm**, 14 mm past the cards' ends at the top and at the bottom, as a standard 9U backplane is. The frame's rails bolt to those two strips, outside the cards, and the corner and edge mounting holes are in them. **Every card goes in with its components facing the front** (toward slot 0): the sockets are turned so (§4).

| Slot | Card | Its J3 |
|---|---|---|
| 0 | the debug card (D64) | the local link, a third drop |
| 1 | CPU | the local link |
| 2 | MMU / data, with the arbiter | the local link |
| 3–9 | any: memory, I/O, disk, video | power only |

Seven general slots: four memory cards, I/O, disk and video fill them. The bus definition still has an eleventh position, slot 10, and the arbiter a request pair for it; this backplane has no such slot (the MMU / data card pulls an empty slot's `-BREQ` up, so it is never granted).

**The nets are the bus definition's.** Every pin of every socket is on `p32bus.busnet(connector, pin, slot)`, so the backplane cannot disagree with docs/bus.md or with the cards:

- **J1 and J2 are bused to every slot**: the cycle (D0–31, A2–31, BE0–3, `-MRD`, `-MWR`, `-ACK`, `-WAIT`) and control (CLK, `-CLK`, `-RES`, PWROK, `-LOCK`, `-IRQ0–7`, `-IACK`). **Their 19 spare pins too** (J2 A22–A30, C21–C30; A19–A21 became `-FAST`, XCLK and `-XCLKEN`, D74, D76): a later signal needs no new backplane (as P8X's rev D).
- **The radial request lines.** Each requester's own `-BREQ` / `-BGNT` pins (J2 A16 / A17) run to their own pins in slot 2 (the arbiter: J2 C3–C20) and in slot 0, so the debug card sees every request as the arbiter does: `-BREQ3`–`-BREQ9`, `-BGNT3`–`-BGNT9`, and slot 0's own pair, `-BREQ0` / `-BGNT0`, to slot 2 (slot 10's pair joins slots 0 and 2 only). In slots 1 and 3–9, J2 C3–C20 are not connected.

- **J3, the local link** (LR0–31, RSRC, RDST, MEM, SIZE, SIGNED, SUPER, TLBW, CRSEL, FAULT, FC, `-WSTB`, STALL, LOCKREQ, PCINC, MARPC, MARINC, CCLK), joins slots 0, 1 and 2 only. In slots 3–9 a card's J3 carries power and ground; its signal pins are not connected. **J3's eight spare pins (B28, B30, C25–C30) join slots 0, 1 and 2 as well** (D99): spare private lines between the debug, CPU and MMU / data cards, should one be needed after the backplane is made. **Two were needed before it was** (D102): B28 is now `-CPUIN` and B30 `-MMUIN`, which the CPU card and the MMU / data card each tie to ground so that the debug card can tell they are in. No copper changed, only the two nets' names. C25–C30, six pins, are still spare.
- **Power on every socket**: pins 1–2 of every row VCC, 31–32 GND, row B's odd pins GND, and J2 row B's even pins VCC as well (D57): 32 VCC and 60 GND pins per card.

## 2. The parts no card can hold

| Ref | Part | Why |
|---|---|---|
| P1, P2 / P3, P4 | M4 bolt and ring lug, in a plated hole stitched to its plane | the power entry: +5V and GND, two pairs, both at the back (D78, D99) |
| U1 | TL7705ACP (DIP-8) | **PWROK**: the +5 V supervisor (D78) |
| D1, R1 | LED, 1k | **+5V**: the supply is on |
| R2 | 1k | `-ACK` pull-up |
| R3 | 680 | `-WAIT` pull-up |
| R4 | 1k | `-LOCK` pull-up |
| R5 | 10k | PWROK pull-up |
| R6 | 10k | `-FAST` pull-up: the CPU card's clock jumper, open = 3 MHz (D74) |
| RN1 | 2.2k × 8, bussed (SIP-9) | `-IRQ0` – `-IRQ7` pull-ups |
| RN2–RN13 | 10k × 8, bussed (SIP-9) | hold D0–31, A2–31, BE0–3, `-MRD`, `-MWR`, `-IACK`; each network on eight consecutive pins of one connector column, so it stands beside the socket rows on the board (D79; some resistors unused) |
| R7, C1 / R8, C2 | 100, 150 pF | AC termination of CLK / `-CLK` past slot 9 |
| C3–C42 | 100 nF beside each socket; 10 µF a slot | bypass |
| C43, C44 | 1000 µF 16 V | bulk, one at each power entry |
| C45, C46, C47 | 100 nF, 10 µF, 100 nF | U1's REF, its delay (CT), its bypass |
| H1–H18 | M3 hole, plated, on GND | the subrack (D99): the four corners (H1–H4); in the 14 mm strip above the cards, in line with the gaps between slots 1–2, 3–4, 5–6 and 7–8 and over the parts at the back (H5–H9); the same in the strip below the cards (H10–H14); four mid-board, in the gaps between the sockets (H15–H18) |

### 2.1 Power: a dedicated 5 V supply (D78)

The supply is a **dedicated +5 V switching supply**, 30 A at least and **40 A recommended** (a known brand, e.g. Mean Well LRS-200-5): the machine's worst case with four memory cards is 30.3 A, and typical about 17 A (power-budget.md). It is switched on the mains side (a switch or a switched, fused inlet on the case); the backplane has no power switch or jumper.

- **The entry: two pairs of M4 bolts**, +5V and GND, both at the back (D99: P1 / P2 between the J1 and J2 rows, P3 / P4 between the J2 and J3 rows, past slot 9). Each is a plated 4.3 mm hole ringed with vias into its plane, an M4 bolt and nut through it and a ring lug on **10 AWG** wire under the head. Both pairs come from the same supply's terminals,

so each carries about half the current and no one point runs hot; one pair alone works for a light machine. A ring lug cannot pull out as a clamped bare wire can.

- **PWROK** comes from **U1, a TL7705A supply supervisor** (TI): its open-collector `RESET` output holds PWROK low until +5 V is above 4.55 V (SENSE on VCC), then for t = 1.3 × 10⁴ × CT (10 µF: about 130 ms), and pulls it low again whenever +5 V dips below the threshold. The CPU card holds reset until PWROK is high and sees it fall first when power fails, as it would have with an ATX supply's PWR_OK. `RESIN` is tied high; REF has the datasheet's 100 nF.
- **+12 V is not on the backplane.** The disk card's IDE hard drives take +12 V (and their +5 V) from the supply's own outputs if it has a 12 V rail, or from a second, 12 V supply; CompactFlash needs 5 V only.

### 2.2 The pull-ups: a rule, checked against the cards

**Every bus line on J1 and J2 that no card drives with a totem-pole output gets a resistor to VCC here, and no other line does.** The generator reads every card's netlist to find each line's drivers, and fails if a line breaks the rule either way:

- **The open-drain lines** (74HC05, 74LS145): they rise only through the resistor, so it is sized for speed. A card lets `-WAIT` go at mid-cycle (on `-CLK`); it must be high at the next CLK edge, half a clock later, less the card's GAL and gate delays and the master's setup: about 60 ns at 5 MHz, 130 ns at 3 MHz. The line is about 200 pF (eleven sockets, their cards' inputs, 30 cm of track); reaching HCT's 2.0 V takes 0.5 RC. `-WAIT` **680 Ω**: 70 ns, with 7.4 mA into a 74HC05 at about 0.45 V (HCT reads up to 0.8 V as low). `-ACK` and `-LOCK` have a whole clock: **1k**. `-IRQ0–7` are levels: **2.2k**, as P32's.
- `-FAST` (D74): a static level, the CPU card's clock jumper, which grounds it when fitted (the 4 MHz clock). **10k** holds it high when the jumper is open, or when no CPU card is fitted (the debug card alone).
- **The three-state lines**: D0–31 (driven only during a read or a write), A2–31, BE0–3, `-MRD`, `-MWR`, `-IACK` (driven by the master, so loose while mastership changes hands, or with the MMU / data card out). **10k** only holds them, so no CMOS input floats (the MMU / data card reads D through 74HCT253s on every cycle). The MMU / data card's own 10k on D0–7 (no vector reads `0xFF`) stay; with these, 5k.
- **PWROK**: 10k, for the supervisor's open-collector output (D78).
- **Not here:** the radial `-BREQ` lines (the MMU / data card pulls them up, so an empty slot asks for nothing, D62), and J3's lines (between the CPU and the MMU / data card, which drive them).

### 2.3 Termination

**CLK and** `-CLK` clock every card. The CPU card drives them from slot 1, near one end; at the other end, **past slot 9**, each line has **100 Ω in series with 150 pF to ground** (AC termination, as P32's terminator card, D56 there): it damps the reflection from the open end while an edge passes, and draws no current in the steady state, so it is no load on a 74HCT or GAL driver. Values to settle on the bench (scope CLK at slots 0, 1 and 10, with every card in). The other lines change while the clock is steady, and are taken at its edges.

### 2.4 Bypass

A 100 nF (0.1" leads, D47) beside each of the 30 sockets, a 10 µF a slot, and two 1000 µF at the power entry. Each card has its own bypass and bulk capacitors as well.

## 3. Verification

`python3 generators/card_backplane.py` (in `make test`):

- the netlist's electrical checks (p32net: every pin connected or declared unused, no net with one end);
- every pin of every socket against `busnet()`; every bus net reaches at least two sockets;
- **every card in every slot it may use**: the CPU card in slot 1, the MMU / data card in 2, the debug card in 0, the memory, I/O, disk and video cards in 3–9: each connector pin the card uses must be wired there (a device card using a J3 signal, for instance, would fail);
- the pull-up rule above, against every card's drivers; termination only on lines a card drives totem-pole.

The gate-level system simulation (`generators/sim_sys.py`) joins the cards by these same net names, with the open-drain lines' pull-ups modelled.

## 4. The board (D79)

`generators/p32backplane.py` places **and routes** it; no autorouter. `sh generators/build_board.sh backplane` adds the gerbers, renders, the placement drawing and the DRC.

- **477 × 394.7 mm, 6 layers** (D77, D90, D99): F.Cu, In1 GND, In2, In3, In4 VCC, B.Cu. Slot 0 at the left (the front), 38.1 mm apart; 34 mm of board in front of slot 0, the room its card's components have; 14 mm above and below the cards' ends (a card's top edge is 14 mm below the board's).
- **The sockets** are turned 180°, so that pin 1 is at the bottom as on the cards: each socket's a1 height comes from the card generator's own placement of J1–J3, so the two cannot disagree. Turned so, column c is on the front side of column a. On a card's right-angle connector row a is the row nearest its board and row c the farthest above it, so **a card's components face the front** (D99). That reading is from KiCad's footprints of the two connectors: **VERIFY with a real connector and socket before ordering.**
- **Routing:** a bused net is the same pin in every slot, at one height. Each pin drops a 1.27 mm stub into the channel between its row and the next, and one straight track runs along the channel: column a on F.Cu, b on In2, c on In3. The request lines (`-BREQn` / `-BGNTn`): the C pins in slots 0 and 2 are one channel, carried on along the same row to slot n; there a via and a riser on B.Cu, between two pin columns, climb to the slot's A16 or A17. Every track 0.2 mm, at least 0.395 mm from the next pad.
- **Everything else is at the back** (D99), in a field past slot 9: the pull-ups, every network pin and resistor end on its own net's row, so each net's channel simply runs on to it; the termination (R7 / C1, R8 / C2) and the **TL7705A**, on PWROK's row, among them.
- **Power:** the two pairs of M4 bolts (+5V above GND) at the back, in the gaps between the socket rows where no channel runs, a 1000 µF beside each pair; the **+5V LED (D1) and its resistor (R1) at the front edge**, 6 mm in, at the J1 end; the bypass capacitors in the gaps beside each slot; the planes reach every VCC and GND pin.
- **Mounting holes** (D99): 18, M3, plated and on GND. Fourteen are in the two 14 mm strips past the cards' ends, 7 mm from the board's edge: the corners, one in line with every second gap between slots (1–2, 3–4, 5–6 and 7–8: 76.2 mm apart) and one over the field. Four are mid-board, in the same four gaps, between the socket rows.
- **Silkscreen:** along the front edge, reading from the front: FRONT, **the board's name, revision and `cottageworker.com`, 7 mm tall**, and "CARDS GO IN WITH THEIR COMPONENTS FACING THE FRONT (SLOT 0)"; BACK along the back edge; **each slot's number and what it is for, 3 mm and 2.5 mm tall**, between its J1 and J2 sockets: SLOT 0 DEBUG, SLOT 1 CPU, SLOT 2 MMU/DATA, SLOT 3–9 GENERAL PURPOSE (D99); **"a1" beside pin a1 of every socket**, outside the socket's body (below); +5V and GND at each bolt, +5V at the LED, every reference beside its part.
- **Fitting the sockets.** The board's wiring fixes how a *card* must stand: its components toward the front and its pin 1 at the bottom, so that its pin a1 meets the hole wired as a1 (on the memory card J1's a1 is 89.37 mm below the card's

top edge, and so is the board's a1 hole; J2 222.72, J3 356.07). The socket itself is 96 straight-through contacts, the same electrically either way round, and its markings are only markings (Ken); but its keyed shell decides which way a card can be pushed in. So each socket is soldered the way round that takes a card standing as above: by the connectors' own markings that is with the socket's a1 at the board's printed "a1". The other way round, a card would only go in turned about, with its a1 in the hole wired as c32: every signal on the wrong pin and +5 V and ground swapped (pins 1–2 are +5 V, 31–32 ground). To try with a real card connector before soldering. A socket's two fixing holes are 0.3 mm off its pins' centre line, so turned about they sit 0.6 mm from the board's.

- **Renders:** top, bottom and 3D (`backplane-render-*.png`), as for every board since 2026-11-11.
- **The supply:** a dedicated +5 V supply, 40 A recommended (§2.1, D78; docs/power-budget.md).

# The backplane: theory of operation

docs/backplane-theory.md

**Status:** DESCRIBES THE NETLIST AND THE BOARD AS GENERATED AND CHECKED (2026-10-29): the netlist `generators/card_backplane.py` and the board `generators/p32backplane.py`, on the bus definition `generators/p32bus.py`. The board's summary and parts list are in the backplane; the bus, pin by pin, in the bus. D49, D50, D52, D57, D64, D68, D74, D76, D77, D78, D79. **Brought up to D99 (2026-11-11, Ken): ten slots, 1.5 inches apart, front and back, every part and the power at the back, 18 mounting holes, the slots labelled.**

## 1. What the backplane does

The backplane is **passive**: sockets, copper and the parts no card can hold (the pull-ups, the clock's termination, the power entry, a +5V LED, the capacitors, the mounting holes, and its only chip, U1, the supply supervisor that makes `PWROK`). It carries **the bus** (D50: physical memory cycles, interrupts, arbitration, the clock, reset and power) to ten positions, and **the local link** (J3) between the CPU card and the MMU / data card. Every socket pin's net **is** `p32bus.busnet(connector, pin, slot)`, from the file the cards take their pins from, so the backplane cannot disagree with the bus or the cards: **30 sockets, 122 parts, 194 bus nets** (`card_backplane.py --bom`).

## 2. The slots

Ten positions, **1.5 inches (38.1 mm) apart** (D99; eleven at 1 inch before, D49, D64), slot 0 at the left: the **front**. The other end, past slot 9, is the **back**. Each position has three female DIN 41612 C 3 × 32 sockets, S*n*J1–S*n*J3, mating a card's J1–J3. **Every card goes in with its components facing the front**: on the board a socket's column c is on the front side of its column a, and on a card's right-angle connector row c is the row farthest above the card (§10; to verify with real parts).

| Slot | Card | Its J3 | Silkscreen |
|---|---|---|---|
| 0 | the debug card (D64): listens to every signal, or drives, never both | the local link, a third drop | SLOT 0, DEBUG |
| 1 | the CPU card: drives `CLK`, `-CLK`, `-RES`; its CLOCK jumper, fitted, grounds `-FAST` | the local link | SLOT 1, CPU |
| 2 | the MMU / data card: the CPU's bus master, and the arbiter | the local link | SLOT 2, MMU/DATA |
| 3–9 | any: memory, I/O, disk, video | power only | SLOT *n*, GENERAL PURPOSE |

**J3, the local link** (62 lines: R as `LR0–31`, the microword's memory fields, `FAULT`, `FC0–3`, `STALL`, `CCLK`, …; nets `L_LR0`, `L_STALL`, …) joins slots 0, 1 and 2 only, so the microcode's memory controls never become bus signals (D49). **Slots 3–9** take any card: a card is placed by its jumpers, not its slot (D50, D52), and the slot sets only its bus-request priority. The bus definition has a slot 10 as well, and the arbiter a request pair for it; this backplane has no such position (D99).

## 3. Bused lines and per-slot lines

**Power on every socket:** pins 1–2 of every row VCC, 31–32 GND, row B's odd pins GND (a guard between rows A and C), J2 row B's even pins VCC too (D57): **32 VCC and 60 GND pins per card**.

**Bused to all ten slots**, the same pin in every slot: **J1, the cycle** (70 lines: `D0–31`, `A2–31`, `BE0–3`, `-MRD`, `-MWR`, `-ACK`, `-WAIT`; J1 has no spare pins); **J2, control** (17 lines: `CLK`, `-CLK`, `-RES`, `PWROK`, `-LOCK`, `-IRQ0–7` on A3–A15, and `-IACK`, `-FAST`, `XCLK`, `-XCLKEN` on A18–A21); and **J2's 19 spare pins** (A22–A30, C21–C30, D68), so a later signal needs no new backplane.

**Per slot: the radial request lines.** A requester's own `-BREQ` / `-BGNT` pins are J2 A16 / A17, and each slot's pair is two nets of its own:

| Nets | From | To |
|---|---|---|
| `-BREQn`, `-BGNTn` (n = 3–9) | slot *n*'s J2 A16, A17 | the same J2 C pin (C3–C18) in slot 2, the arbiter, and in slot 0, the debug card |
| `-BREQ10`, `-BGNT10` | no slot (D99) | their J2 C pins in slots 2 and 0 only |
| `-BREQ0`, `-BGNT0` | slot 0's J2 A16, A17 | slot 2's J2 C19, C20 |

**Not connected:** J2 A16–A17 in slots 1 and 2 (neither card asks for the bus), J2 C3–C20 in slots 1 and 3–9, J2 C19–C20 in slot 0, J3's signal pins in slots 3–9, and J3's spare pins there too. In slots 0, 1 and 2 J3's pins B28, B30 and C25–C30, spare when the backplane was drawn, are joined, slot to slot, as the local link is (D99). Two of the eight are now signals (D102): B28 is `-CPUIN` and B30 `-MMUIN` (nets `L_-CPUIN`, `L_-MMUIN`), each tied to ground by the card it names and pulled up on the debug card, which will not drive a line of a card that is in (debug-card-theory §9). The backplane only joins them: it has no part on either. C25–C30 (nets `L_SPARE_J3C25`, …) are six spare private lines.

## 4. Requests, grants and the lock

The backplane only carries the radial lines; the arbiter is **GAL ARB** on the MMU / data card, clocked by `CLK` (mmu-card-theory §9). At a `CLK` edge with no grant out and `-LOCK` high, the requesting slot of the highest priority, **slot 0 first, then the highest-numbered slot**, gets its grant and is master **from the next clock**. It keeps the bus as long as it holds its `-BREQ` low; `-RES` clears every grant. GAL ARB drives the `-BGNT` lines totem-pole. The disk card's DMA engine (D75) asks from GAL DKENG, the debug card through a 74HC05. **The request lines' pull-ups are on the MMU / data card**, not here: a 10k on each of `-BREQ0` and `-BREQ3–10` holds an empty slot's request high.

`-LOCK` (J2 A7) is open-drain, from a 74HC05 on the MMU / data card (a `SWAP`'s read and write, D51, D54) and one on the debug card; while it is low the arbiter grants nothing. **The masters** (the MMU / data card for the CPU, the disk card's DMA engine, the debug card) drive the address, byte enables, `-MRD` / `-MWR` and a write's data through three-state drivers; only the MMU / data card and the debug card drive `-IACK`.

## 5. The pull-ups

**The rule** (D68): every J1 / J2 bus line that **no card drives with a totem-pole output** gets a resistor to VCC here, and no other line does. The generator reads every card's drivers and fails if a line breaks the rule either way.

| Line | Pin | Ref | Value | Driven by |
|---|---|---|---|---|
| `-ACK` | J1 B28 | R2 | 1k | 74HC05, open drain: the memory, I/O, disk, video, debug cards |
| `-WAIT` | J1 B30 | R3 | 680 Ω | 74HC05: the I/O, disk, video, debug cards |
| `-LOCK` | J2 A7 | R4 | 1k | 74HC05: the MMU / data, debug cards |
| `PWROK` | J2 A6 | R5 | 10k | U1, open collector (§8.2) |
| `-FAST` | J2 A19 | R6 | 10k | the CPU card's CLOCK jumper: fitted, ground (4 MHz); open, or no CPU card, high (3 MHz) |
| `-IRQ0–7` | J2 A8–A15 | RN1 | 2.2k × 8 | 74LS145, open collector: the I/O (nine), disk, video cards |

| Line | Pin | Ref | Value | Driven by |
|---|---|---|---|---|
| `D0–31`, `A2–31`, `BE0–3`, `-MRD`, `-MWR` | J1 | RN2–RN12 | 10k × 8 | three-state: the masters; on a read, the addressed card's `D` |
| `-IACK` | J2 A18 | RN13 | 10k × 8 | three-state: the MMU / data, debug cards |

- **The open-drain lines** rise only through their resistor, so it is sized for speed: about 200 pF of line (ten sockets, the cards' inputs, 45 cm of track) and 0.5 RC to reach HCT's 2.0 V (backplane.md §2.2). `-WAIT` is let go at mid-cycle (on `-CLK`) and must be high by the next `CLK` edge: 680 Ω, about 70 ns, 7.4 mA into a 74HC05 at about 0.45 V. `-ACK` and `-LOCK` have a whole clock: 1k. `-IRQ0–7` are levels: 2.2k.
- `PWROK` (U1's open-collector output, D78) and `-FAST` (a static level, D74): 10k.
- **The three-state lines** are loose whenever no one drives them: `D` outside a read or a write; the address, byte enables and strobes while mastership changes hands, or with the MMU / data card out. 10k only holds them, so no CMOS input floats (the MMU / data card reads `D` through 74HCT253s on every cycle); a held strobe is high, no cycle. A device master drives no `-IACK` (D75), so RN13 holds it off during a disk DMA.
- **No resistor** on the lines a card drives totem-pole: `CLK`, `-CLK`, `-RES` (the CPU card's GALs CLKGEN and PANEL), `XCLK`, `-XCLKEN` (the debug card's GAL DBCLK); none on the request lines (§4) or on J3.

**The networks** are bussed SIP-9s, each on eight consecutive pins of one connector column (`pull_networks()`, D79), so that on the board every pin stands on its own net's row: RN2–RN5, RN6–RN8, RN9–RN12 on J1 columns A, B, C; RN13 on J2 A18–A25. A row with no such net leaves its resistor unused: 27 of their 96, half of RN6–RN8 (row B's odd pins are GND), seven of RN13's. They stand, with R2–R6 and the termination, in a field past slot 9, at the back, where each net's track ends.

## 6. The clock

`CLK` **and** `-CLK` (J2 A3, A4) come from the CPU card in slot 1, from GAL CLKGEN; `CLK` is high in phases 0–3 of the eight (cpu-card-theory §3). One period is one bus cycle: 333 ns at 3 MHz, or 4 MHz with the CLOCK jumper (D74). Every card works from `CLK`. Each line is one straight track past all ten slots.

**The termination** is at the far end from the CPU card, **past slot 9**: on each line **100 Ω in series with 150 pF to ground** (R7 / C1 on `CLK`, R8 / C2 on `-CLK`). It damps the reflection from the open end while an edge passes and draws no current in the steady state, so it is no load on the GAL driving the line; the values are to settle on the bench (backplane.md §2.3). The other lines are not terminated: they change while the clock is steady and are taken at its edges.

**The clock's other lines**, on J2: with `-FAST` (A19) low the I/O card's timer divides by 4, so it ticks at 1 MHz at either clock (D74). `XCLK` and `-XCLKEN` (A20, A21; D76) come from the debug card's GAL DBCLK; the CPU card's GAL CKSEL takes `XCLK` as its oscillator while `-XCLKEN` is low (12 MHz, or one edge per host STEP). With no debug card, the CPU card's own 10k resistors hold `XCLK` low and `-XCLKEN` high. With no CPU card, the debug card makes `CLK` and `-CLK` itself (GAL DBCLK) and can drive `-RES`, through three-state drivers (D64).

## 7. Interrupts and the acknowledge cycle

`-IRQ0–7` (J2 A8–A15) are eight bused open-drain levels, held high by RN1. A device's level is set by jumpers on its card (D52) and decoded onto its line by a 74LS145, open collector: one per device on the I/O card, one on the disk card, one on the video card. The CPU card takes the levels through a 74ACT574 at `-CLK`'s rise (`card_cpu.py`); the debug card listens. No level depends on a slot. A level belongs to **one card**; one card may drive several (D57).

**The acknowledge cycle** (D57) is a bus cycle like a read, with `-IACK` (J2 A18) low in place of `-MRD` and `A4:A2` = the level; the MMU / data card runs it for the CPU. The card driving that level pulls `-ACK` and puts its vector, window × 16 + the asking device, on `D7:D0`. **No answer is not a bus error:** R2 keeps `-ACK` high, and the vector reads `0xFF` through RN2 and the MMU / data card's own 10k on `D0–7` (5k together). The kernel then reads the cards' interrupt status registers.

# 8. Power

## 8.1 The supply, the bolts and the capacitors (D78)

**A dedicated +5 V switching supply**, 30 A at least, **40 A recommended** (e.g. Mean Well LRS-200-5): the machine with four memory cards is 16.9 A typical and 30.3 A at its datasheet worst case, the backplane itself 5 mA, 79 mA at most (power budget). It is switched on the mains side; the backplane has no power switch or jumper. **+12 V is not on the backplane:** the IDE drives take it from the supply's 12 V rail or a second supply, and their +5 V from the supply directly.

**The entry is two pairs of M4 bolts, both at the back** (D99): P1 (+5V) and P2 (GND) in the gap between the J1 and J2 rows, P3 (+5V) and P4 (GND) in the gap between the J2 and J3 rows, over the field past slot 9, where no channel runs: a plated 4.3 mm hole ringed with vias into its plane, a ring lug on 10 AWG wire under the bolt's head. Both pairs come from the same supply, so each carries about half the current. **The planes**, In1 GND and In4 VCC, reach every socket's VCC and GND pins; a card's 32 VCC and 60 GND contacts are rated 1–2 A each, against the largest card's 7.8 A at most. **H1–H18**, M3 plated holes for the subrack, are on GND (D99). The board is 14 mm taller than the cards at the top and at the bottom (394.7 mm against 366.7), and fourteen of the holes are in those two strips, outside the cards' ends: the corners, one in line with every second gap between slots (1–2, 3–4, 5–6, 7–8: 76.2 mm apart) and one over the field. Four more are mid-board, in the gaps between the socket rows.

**The capacitors:** a 100 nF beside each of the 30 sockets and a 10 µF a slot (C3–C42, in the gaps between the J1, J2 and J3 sockets), and a 1000 µF 16 V beside each pair of bolts (C43 by P1 / P2, C44 by P3 / P4); every card has its own as well. **D1**, an LED through R1 (1k), labelled +5V, shows the supply is on: the two stand at the front edge, where they can be seen (D99), the only parts there.

## 8.2 PWROK: the supervisor (D78)

**U1, a TL7705A** (TL7705ACP, DIP-8), stands at the back with the other parts (D99), on `PWROK`'s row, so the line runs through every slot and on to R5 and to it:

| Pin | Net | Use |
|---|---|---|
| 8 VCC, 7 SENSE | VCC | it senses +5 V itself: threshold 4.55 V |
| 2 `-RESIN` | VCC | tied high |
| 1 REF | C45, 100 nF to GND | the datasheet's reference capacitor |
| 3 CT | C46, 10 µF to GND | the delay, 1.3 × 10⁴ × CT: about 130 ms |
| 5 `-RESET` | `PWROK` | open collector, pulled up by R5 (10k) |
| 6 `RESET` | — | unused |

`PWROK` is low until +5 V is above 4.55 V, then for about 130 ms more, and low again whenever +5 V dips below the threshold. The CPU card's GAL PANEL registers `RESET = RESBTN or not PWROK` and drives `-RES` onto the bus (boot §1), so every card is held in reset until power is good. The debug card listens to `PWROK`. C51 (100 nF) decouples U1.

## 9. An empty slot

Nothing need be fitted in any slot. An empty socket stays on every bused net, and the bus sees **no** `-ACK` at the addresses its card would answer: R2 holds `-ACK` high and the MMU / data card makes the cycle a **bus error**, so a missing card never hangs the bus (the kernel finds the cards this way, D52). It sees **no wait** (R3), **no interrupt** (RN1; an acknowledge cycle on a level no card drives reads `0xFF`), and **no request**: the MMU / data card's 10k holds the slot's `-BREQn` high, so GAL ARB never grants it. In slots 3–9, J3 is power only. Three are special:

- **Slot 0 empty** (no debug card): the MMU / data card's 10k holds `-BREQ0` high; the CPU card's resistors hold `XCLK` low and `-XCLKEN` high, so the CPU card keeps its own oscillator.
- **Slot 1 empty** (the debug card alone, D64): R6 holds `-FAST` high, the 3 MHz setting; `CLK`, `-CLK` and `-RES` have no driver but the debug card.
- **Slot 2 empty:** no arbiter, and no pull-ups on the request lines; the backplane's 10k still holds `D`, the address, byte enables and strobes.

## 10. Verified

`python3 generators/card_backplane.py`, in `make test`, checks the electrical rules (`p32net.check`: every pin connected or declared unused, no pin on two nets, no net with one end, at most one totem-pole driver); every pin of every socket in every slot against `busnet()`, every bus net and spare on two sockets or more; **every card in every slot it may use** (debug 0, CPU 1, MMU / data 2, memory, I/O, disk, video 3–9), each pin the card uses wired there; the pull-up rule against every card's drivers, both ways; and termination only on lines a card drives totem-pole. It prints `backplane OK: 10 slots (0-9), 30 sockets, 194 bus nets, 122 parts; every card fits its slots`. `p32bus.py --check`, also in `make test` (which keeps docs/bus.md current), checks each signal's one pin, the local link's `RSRC`, `RDST` and `MEM` widths against the microword, and that each radial line reaches its own slot, slot 2 and, from slots 3–10, slot 0.

**The board** (D77, D79): `p32backplane.py` places **and routes** it, with no autorouter: **477 × 394.7 mm, 6 layers** (F.Cu, In1 GND, In2, In3, In4 VCC, B.Cu; D99), the sockets turned 180° so that pin 1 is at the bottom as on the cards and column c is on the front side of column a (so a card's components face the front, by KiCad's footprints of the two connectors), each bused net one straight track. It asserts each socket's pins where the cards put theirs, each network on eight rows of one column, and no two nets' tracks overlapping on a layer; KiCad's DRC found **0 violations, 0 unconnected**. **Not yet checked:** the termination's values (bench), and the sockets' orientation against a real card (backplane.md §4).

## 11. Open questions

1. **Interrupt priority.** Settled 2026-11-11: bus.md's signal table said `-IRQn` "(IRQ0 = timer; 7 highest)"; it now says what its Interrupts text, isa-system.md, cpu-card-theory §14 and the emulators do: the lowest-numbered level first.
2. **The clock rates.** bus.md's cycle text said "5 MHz planned, 3 MHz as the safe setting"; since 2026-11-11 it says 3 MHz, or 4 MHz with the jumper (D72, D74). Still open: backplane.md §2.2 budgets `-WAIT` at 5 and 3 MHz only, and its 680 Ω rise (about 70 ns) is over its own 5 MHz budget (about 60 ns).
3. **When the arbiter grants.** bus.md: "when the CPU's next microcycle does not use the bus"; GAL ARB has no such input and grants at any `CLK` edge with no grant out and `-LOCK` high (mmu-card-theory §9). What becomes of a CPU cycle stretched by `-WAIT` when a grant falls on one of its clock edges is not said.

4. `XCLK` **is not terminated**, though it is a clock (12 MHz, or single edges, D76) driven from slot 0 into the CPU card's oscillator select; backplane.md §2.3's reason for terminating only `CLK` and `-CLK` does not cover it.
5. **Slot 2 empty:** `-BGNT0` and `-BGNT3–10` are driven, and `-BREQ0` and `-BREQ3–10` held, only by the MMU / data card; with the debug card alone nothing drives the grants or holds the requests. Intended?
6. **Stale comments and names.** (The mounting holes: settled by D99, which puts them at the corners, along both edges and between the sockets, in the netlist, the board and backplane.md alike.) p32bus.py's comments give J3 as "slots 1-2" and the "arb" lines as "slot 2 only"; `busnet()` wires both to slot 0 too. D78 and backplane.md §2.1 call U1's pins 5 and 2 `RESET` and `RESIN`, the netlist `-RESET` and `-RESIN` (pin 6, `RESET`, is unused). cpu-card-theory §14 (a 74HCT574 on `-IRQ0–7`) and mmu-card-theory §6 (74HCT541s on the address) predate D72's 74ACT574 and 74ACT541s.

# The power budget

docs/power-budget.md

*Generated by* `generators/p32power.py` *(D69) from every card's netlist — edit the generator, not this file.*

Every card's **+5 V** current: **typical** (the datasheets' typical figures, or our estimate at the bus clock) and **maximum** (each part at its datasheet maximum and fastest rated frequency, all at once: an upper bound that is never reached together). Parts behind a card's 3.3 V regulator count at 5 V (a linear regulator passes its current through). Rows marked *estimate* have no datasheet figure yet; the Tang Nano 20K and the Picos are to measure.

**CMOS logic** (74HCT, HC, ACT, LVC) draws microamps standing still; its current is switching: C_pd × V × f for each gate and its loads, and HCT's extra current for an input between the rails (a GAL's or TTL's high). Estimated as 1 mA a chip at the bus clock (4 mA at most), 4 mA a chip for ACT on the dot or capture clock (10 mA at most).

## The cards

| Card | Typical | Maximum | The largest share |
|---|---|---|---|
| CPU | 4.45 A | 7.82 A | 35 × GAL: 4.55 A at most |
| MMU / data | 1.79 A | 3.37 A | 7 × CY7C199: 1.08 A at most |
| memory | 0.31 A | 0.79 A | 3 × GAL: 0.39 A at most |
| I/O | 2.28 A | 3.90 A | 16 × GAL: 2.08 A at most |
| disk | 1.79 A | 2.86 A | 15 × GAL: 1.95 A at most |
| video | 1.16 A | 2.28 A | 5 × GAL: 0.65 A at most |
| debug | 3.17 A | 5.97 A | 24 × CY7C199: 3.72 A at most |
| backplane | 5 mA | 79 mA | the +5V LED and the supervisor; the pull-ups with every pulled line low |

## The machine

| Machine | Typical | Maximum |
|---|---|---|
| every card, 1 memory card | 15.0 A | 27.1 A |
| every card, 4 memory cards | 15.9 A | 28.7 A |

Wired from the supply directly, not through the backplane, but from the same +5 V rail (D78):

| Drives | Typical | Maximum |
|---|---|---|
| 2 CompactFlash cards (in CF-to-IDE adapters) | 0.2 A | 0.4 A |
| 2 IDE hard drives, 3.5" (their +5 V; +12 V for the motor) | 1.0 A | 1.6 A |

**The supply** (D78): a dedicated +5 V supply on the backplane's M4 bolts. The full machine (four memory cards) is **16 A typical, 29 A at most** on +5 V; with its drives on the same rail, 17 A typical, 31 A at most. So **30 A at least, 40 A recommended**: the worst case adds every part's datasheet maximum, which never happen together, but a cheap supply sags near its rating. The drives' +12 V (the hard drives' motors) comes from the supply's own 12 V rail or a second, 12 V supply; it is not on the backplane.

**The connectors carry it:** each card has 32 VCC and 60 GND pins (DIN 41612 contacts, rated 1–2 A each), against the largest card's 7.8 A at most. The backplane takes its 29 A at most on two pairs of M4 bolts (+5V and GND, one pair at each end, 10 AWG wire), about half each.

**The 3.3 V regulators** (LD1117V33, 800 mA, TO-220): the CPU card's feeds the ALU's 74LVC latches and buffers (tens of mA); the I/O card's the WIZ850io (132 mA) and its 3.3 V buffers, about 150 mA: 0.25 W in the regulator, no heatsink needed. The Picos and the Tang Nano make their own 3.3 V from 5 V.

**Where the current goes:** the GALs (ATF22V10C, about 85 mA each, whether they switch or not) are 7.5 A of the 15.9 A typical. Microchip's quarter-power and zero-power 22V10s (CQ, CQZ) draw a fraction of that but are slower (15–20 ns); which GALs could take them is a question for the timing pass against the datasheets (backlog).

## The CPU card

| Part | Count | Typical | Maximum | Source |
|---|---|---|---|---|
| GAL | 35 | 2.98 A | 4.55 A | datasheet: ATF22V10C-7 / -10, ICC (commercial) |
| CY7C199 | 8 | 0.64 A | 1.24 A | datasheet: max: CY7C199-15 at 1/tRC (66 MHz); typ: the CY7C199D's specified maximum |
| 74F283 | 8 | 0.29 A | 0.44 A | estimate: the F family's adders |
| TANG_NANO_20K | 1 | 0.10 A | 0.30 A | estimate: the ALU bitstream, SDRAM idle, the BL616: to measure |
| SST39SF010A | 9 | 90 mA | 0.27 A | datasheet: read; standby 30 uA typ (max read to confirm on the datasheet) |
| 74ACT574 | 16 | 64 mA | 0.16 A | estimate: switching at 12.5-24 MHz |
| 74ACT541 | 12 | 48 mA | 0.12 A | estimate: switching at 12.5-24 MHz |
| 74ABT377 | 4 | 20 mA | 0.12 A | datasheet: 30 mA with its outputs low |
| 74F153 | 4 | 48 mA | 80 mA | datasheet |
| 74HCT541 | 16 | 16 mA | 64 mA | estimate: switching at the bus clock |
| 74LVC574 | 13 | 13 mA | 52 mA | estimate: switching at the bus clock, 3.3 V |
| 74ACT157 | 4 | 16 mA | 40 mA | estimate: switching at 12.5-24 MHz |
| 74HCT157 | 8 | 8 mA | 32 mA | estimate: switching at the bus clock |
| 74HCT257 | 8 | 8 mA | 32 mA | estimate: switching at the bus clock |
| 74HCT153 | 8 | 8 mA | 32 mA | estimate: switching at the bus clock |
| 74HCT161 | 8 | 8 mA | 32 mA | estimate: switching at the bus clock |
| OSC24M | 1 | 15 mA | 30 mA | estimate: a DIP-14 can oscillator |
| 74ACT32 | 3 | 12 mA | 30 mA | estimate: switching at 12.5-24 MHz |
| OSC32M | 1 | 15 mA | 30 mA | estimate: a DIP-14 can oscillator |
| 74HCT377 | 5 | 5 mA | 20 mA | estimate: switching at the bus clock |
| 74F138 | 1 | 13 mA | 20 mA | datasheet |
| 74ACT08 | 2 | 8 mA | 20 mA | estimate: switching at 12.5-24 MHz |
| 74HCT253 | 4 | 4 mA | 16 mA | estimate: switching at the bus clock |
| 74HCT4002 | 4 | 4 mA | 16 mA | estimate: switching at the bus clock |
| LED | 3 | 9 mA | 15 mA | estimate: with its resistor |
| 74HCT138 | 3 | 3 mA | 12 mA | estimate: switching at the bus clock |
| LD1117V33 | 1 | 5 mA | 10 mA | datasheet: quiescent |
| 74HCT540 | 2 | 2 mA | 8 mA | estimate: switching at the bus clock |
| 74HCT574 | 2 | 2 mA | 8 mA | estimate: switching at the bus clock |

| Part | Count | Typical | Maximum | Source |
|---|---|---|---|---|
| 74LVC244 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock, 3.3 V |
| 74HCT148 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| 74HCT00 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| 74HCT14 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| **total** | | **4.45 A** | **7.82 A** | |

## The MMU / data card

| Part | Count | Typical | Maximum | Source |
|---|---|---|---|---|
| CY7C199 | 7 | 0.56 A | 1.08 A | datasheet: max: CY7C199-15 at 1/tRC (66 MHz); typ: the CY7C199D's specified maximum |
| GAL | 8 | 0.68 A | 1.04 A | datasheet: ATF22V10C-7 / -10, ICC (commercial) |
| 74F161 | 8 | 0.30 A | 0.44 A | datasheet: as the 74F163 |
| 74ABT377 | 7 | 35 mA | 0.21 A | datasheet: 30 mA with its outputs low |
| 74F521 | 4 | 80 mA | 0.12 A | estimate: the F family's comparators |
| 74ACT541 | 11 | 44 mA | 0.11 A | estimate: switching at 12.5-24 MHz |
| 74HCT157 | 27 | 27 mA | 0.11 A | estimate: switching at the bus clock |
| 74HCT541 | 24 | 24 mA | 96 mA | estimate: switching at the bus clock |
| 74HCT253 | 8 | 8 mA | 32 mA | estimate: switching at the bus clock |
| 74HCT161 | 8 | 8 mA | 32 mA | estimate: switching at the bus clock |
| 74ACT157 | 2 | 8 mA | 20 mA | estimate: switching at 12.5-24 MHz |
| 74HCT377 | 5 | 5 mA | 20 mA | estimate: switching at the bus clock |
| 74HCT257 | 4 | 4 mA | 16 mA | estimate: switching at the bus clock |
| 74HCT138 | 3 | 3 mA | 12 mA | estimate: switching at the bus clock |
| 74ACT08 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| LED | 1 | 3 mA | 5 mA | estimate: with its resistor |
| 74HCT00 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| 74HCT32 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| 74HC05 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| **total** | | **1.79 A** | **3.37 A** | |

## The memory card

| Part | Count | Typical | Maximum | Source |
|---|---|---|---|---|
| GAL | 3 | 0.26 A | 0.39 A | datasheet: ATF22V10C-7 / -10, ICC (commercial) |
| AS6C4008 | 32 | 40 mA | 0.24 A | datasheet: one bank of 4 active (typ: a third of the cycles); the others deselected (4 / 50 uA) |
| SST39SF020A | 4 | 0 mA | 0.12 A | datasheet: deselected after booting (30 uA); max: all four read |
| 74HCT245 | 4 | 4 mA | 16 mA | estimate: switching at the bus clock |
| 74HCT541 | 3 | 3 mA | 12 mA | estimate: switching at the bus clock |
| LED | 1 | 3 mA | 5 mA | estimate: with its resistor |
| 74HCT138 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| 74HC05 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| **total** | | **0.31 A** | **0.79 A** | |

## The I/O card

| Part | Count | Typical | Maximum | Source |
|---|---|---|---|---|
| GAL | 16 | 1.36 A | 2.08 A | datasheet: ATF22V10C-7 / -10, ICC (commercial) |
| SOCK2X8 | 2 | 0.22 A | 0.46 A | estimate: a CH376S module (12 / 30 mA) and its USB drive (100 / 200 mA, the design's budget) |
| 74F163 | 8 | 0.30 A | 0.44 A | datasheet |
| PICO | 1 | 60 mA | 0.30 A | estimate: Pico 2 W: Wi-Fi on (peaks while sending) |
| WIZ850IO | 1 | 0.13 A | 0.15 A | datasheet: 132 mA (WIZnet / DigiKey), through the LD1117 |
| 74LS145 | 9 | 63 mA | 0.12 A | datasheet |
| MAX232A | 4 | 32 mA | 80 mA | datasheet: 8 / 10 mA unloaded, plus two RS-232 loads |
| 74HCT540 | 12 | 12 mA | 48 mA | estimate: switching at the bus clock |
| TL16C550 | 4 | 32 mA | 40 mA | datasheet |
| 74HCT541 | 9 | 9 mA | 36 mA | estimate: switching at the bus clock |
| OSC1M8432 | 1 | 10 mA | 20 mA | estimate: a DIP-14 can oscillator |
| 74F138 | 1 | 13 mA | 20 mA | datasheet |
| 74HCT245 | 4 | 4 mA | 16 mA | estimate: switching at the bus clock |
| 74HCT374 | 4 | 4 mA | 16 mA | estimate: switching at the bus clock |
| DS12C887 | 1 | 7 mA | 15 mA | datasheet: max from the distributors' listings |
| 74HCT574 | 3 | 3 mA | 12 mA | estimate: switching at the bus clock |
| LD1117V33 | 1 | 5 mA | 10 mA | datasheet: quiescent |
| 74HCT688 | 2 | 2 mA | 8 mA | estimate: switching at the bus clock |
| 74HCT14 | 2 | 2 mA | 8 mA | estimate: switching at the bus clock |
| LED | 1 | 3 mA | 5 mA | estimate: with its resistor |
| 74HCT125 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| 74LVC244 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock, 3.3 V |
| 74HCT244 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| 74HC05 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| CD74HC4050 | 1 | 1 mA | 2 mA | estimate |
| **total** | | **2.28 A** | **3.90 A** | |

## The disk card

| Part | Count | Typical | Maximum | Source |
|---|---|---|---|---|
| GAL | 15 | 1.27 A | 1.95 A | datasheet: ATF22V10C-7 / -10, ICC (commercial) |
| 74F163 | 12 | 0.44 A | 0.66 A | datasheet |
| 74HCT541 | 20 | 20 mA | 80 mA | estimate: switching at the bus clock |
| 74HCT245 | 13 | 13 mA | 52 mA | estimate: switching at the bus clock |
| 74HCT574 | 10 | 10 mA | 40 mA | estimate: switching at the bus clock |
| LED | 5 | 15 mA | 25 mA | estimate: with its resistor |
| 74HCT540 | 4 | 4 mA | 16 mA | estimate: switching at the bus clock |
| 74LS145 | 1 | 7 mA | 13 mA | datasheet |
| 74HCT688 | 2 | 2 mA | 8 mA | estimate: switching at the bus clock |
| 74HC05 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |

| Part | Count | Typical | Maximum | Source |
|---|---|---|---|---|
| 74HCT153 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| 74HCT85 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| **total** | | **1.79 A** | **2.86 A** | |

## The video card

| Part | Count | Typical | Maximum | Source |
|---|---|---|---|---|
| GAL | 5 | 0.42 A | 0.65 A | datasheet: ATF22V10C-7 / -10, ICC (commercial) |
| MINIDIN6F_RA | 2 | 0.10 A | 0.50 A | estimate: a PS/2 device; both share the 0.5 A polyfuse |
| 74F299 | 4 | 0.27 A | 0.38 A | estimate: the F family's shift registers; to confirm on the datasheet |
| AS6C1008 | 2 | 60 mA | 0.12 A | estimate: taken as the AS6C4008's; to confirm on its datasheet |
| HD63484 | 1 | 60 mA | 0.10 A | estimate: CMOS at 6.3 MHz: Hitachi's manual gives no figure; to measure |
| 27C256 | 2 | 20 mA | 60 mA | estimate: an EPROM read all the time; to confirm on the datasheets (27C256, AT27C256R-70) |
| 74F163 | 1 | 37 mA | 55 mA | datasheet |
| 74F74 | 3 | 33 mA | 48 mA | estimate: the F family's flip-flops; to confirm on the datasheet |
| OSC25M | 1 | 15 mA | 30 mA | estimate: a DIP-14 can oscillator |
| 74ACT32 | 3 | 12 mA | 30 mA | estimate: switching at 12.5-24 MHz |
| 74ACT574 | 3 | 12 mA | 30 mA | estimate: switching at 12.5-24 MHz |
| 74LS145 | 2 | 14 mA | 26 mA | datasheet |
| 74ACT573 | 2 | 8 mA | 20 mA | estimate: switching at 12.5-24 MHz |
| 74ACT245 | 2 | 8 mA | 20 mA | estimate: switching at 12.5-24 MHz |
| 74ACT273 | 2 | 8 mA | 20 mA | estimate: switching at 12.5-24 MHz |
| ATMEGA328P | 1 | 10 mA | 20 mA | estimate: active at 16 MHz, 5 V |
| 74F138 | 1 | 13 mA | 20 mA | datasheet |
| 74HCT541 | 4 | 4 mA | 16 mA | estimate: switching at the bus clock |
| 74F21 | 2 | 8 mA | 16 mA | estimate: the F family's gates; to confirm on the datasheet |
| 74ACT04 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74ACT10 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74ACT11 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74ACT08 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74ACT00 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74ACT02 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74ACT86 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74ACT74 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74HCT245 | 2 | 2 mA | 8 mA | estimate: switching at the bus clock |
| 74HCT688 | 2 | 2 mA | 8 mA | estimate: switching at the bus clock |
| 74HCT574 | 2 | 2 mA | 8 mA | estimate: switching at the bus clock |
| LED | 1 | 3 mA | 5 mA | estimate: with its resistor |
| 74HCT244 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| 74HCT74 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| 74HC05 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |

| Part | Count | Typical | Maximum | Source |
|---|---|---|---|---|
| **total** | | **1.16 A** | **2.28 A** | |

## The debug card

| Part | Count | Typical | Maximum | Source |
|---|---|---|---|---|
| CY7C199 | 24 | 1.92 A | 3.72 A | datasheet: max: CY7C199-15 at 1/tRC (66 MHz); typ: the CY7C199D's specified maximum |
| GAL | 6 | 0.51 A | 0.78 A | datasheet: ATF22V10C-7 / -10, ICC (commercial) |
| 74F161 | 8 | 0.30 A | 0.44 A | datasheet: as the 74F163 |
| TIL311 | 2 | 0.21 A | 0.36 A | datasheet: a hex display: its logic 60 / 90 mA and its LEDs 45 / 90 mA (TI SODS001D) |
| 74ACT574 | 22 | 88 mA | 0.22 A | estimate: switching at 12.5-24 MHz |
| 74HCT541 | 27 | 27 mA | 0.11 A | estimate: switching at the bus clock |
| 74HCT574 | 19 | 19 mA | 76 mA | estimate: switching at the bus clock |
| PICO | 1 | 25 mA | 60 mA | estimate: Pico 2: USB to the Mac |
| 74HCT157 | 8 | 8 mA | 32 mA | estimate: switching at the bus clock |
| OSC24M | 1 | 15 mA | 30 mA | estimate: a DIP-14 can oscillator |
| 74HCT273 | 7 | 7 mA | 28 mA | estimate: switching at the bus clock |
| 74HCT154 | 4 | 4 mA | 16 mA | estimate: switching at the bus clock |
| LED | 3 | 9 mA | 15 mA | estimate: with its resistor |
| 74HCT08 | 3 | 3 mA | 12 mA | estimate: switching at the bus clock |
| 74HC05 | 3 | 3 mA | 12 mA | estimate: switching at the bus clock |
| 74HCT138 | 3 | 3 mA | 12 mA | estimate: switching at the bus clock |
| 74ACT157 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74ACT00 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74ACT04 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74ACT74 | 1 | 4 mA | 10 mA | estimate: switching at 12.5-24 MHz |
| 74LVC245 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock, 3.3 V |
| 74HCT32 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| 74HCT74 | 1 | 1 mA | 4 mA | estimate: switching at the bus clock |
| **total** | | **3.17 A** | **5.97 A** | |

# Timing

docs/timing.md

*Generated by* `generators/p32timing.py` *(D71) from every card's netlist and* `generators/p32delays.py`*'s datasheet figures — edit the generators, not this file.*

A static timing analysis of the cards, alone and together over the backplane. The gate-level simulations (D61–D67) check that the cards are wired to do the right thing; this checks that they do it in time. Method and assumptions: §4. **Worst case** is every chip at its datasheet maximum over temperature (4.5 V, 50 pF, -40..+85 °C); **typical** at its 25 °C typical (`p32delays.typical`). A path fails if its data can arrive after the clock edge that takes it, less that register's setup time.

## 1. Results

| Analysis | Corner | Endpoints | Failing at the design clock | Fastest clock every path allows |
|---|---|---|---|---|
| the machine at 3 MHz | worst case | 2659 | 0 | oscillator 40.4 ns: a **3.09 MHz** bus clock (design: 3 MHz) |
| the machine at 3 MHz | typical | 2659 | 0 | oscillator 25.4 ns: a **4.93 MHz** bus clock (design: 3 MHz) |
| the machine at 4 MHz (the clock jumper) | typical | 2659 | 0 | oscillator 25.4 ns: a **4.93 MHz** bus clock (design: 4 MHz) |
| the disk card's DMA cycles (3 MHz) | worst case | 1229 | 0 | 247.0 ns (design 333.3 ns) |
| the disk card's DMA cycles (3 MHz) | typical | 1229 | 0 | 144.4 ns (design 333.3 ns) |
| the debug card alone (12 MHz timing mode) | worst case | 536 | 0 | 41.0 ns (design 41.7 ns) |
| the debug card alone (12 MHz timing mode) | typical | 536 | 0 | 26.5 ns (design 41.7 ns) |

**Read data on the bus.** A slave's read data must be on the backplane's D lines this long before the cycle's last edge (the path from D through the MMU / data card's load steering to the register that takes it):

- the machine at 3 MHz, worst case: **140 ns** (into mmu/U76 D0)
- the machine at 3 MHz, typical: **71 ns** (into mmu/U76 D0)
- the machine at 4 MHz (the clock jumper), typical: **71 ns** (into mmu/U76 D0)

A device that answers from a clock of its own is held to `READ_SETUP`, 140 ns (`p32timing_config.py`): the video card's chip, whose word is on D at least that long before the cycle ends at every phase of its clock (`generators/sim_video_timing.py`). This pass fails if the machine needs more.

## 2. What fails

Each group is the endpoints of one kind of chip on one card that fail in one kind of bus cycle, with the edges' distance; its worst path in full (each step: the delay, the chip, its pins, the net reached).

## 3. What is not timed, and why

**Case analysis.** The machine is timed three times: a **read** (the slaves drive D, the MMU / data card takes it in through its load steering, which alone drives R), a **write** (the MMU / data card drives D from R, which only the store data drives; the slaves take D in) and an **internal** cycle (nothing on D; R from any source but those two). Each card's chips on D point one way in each; signals made only from `-MWR` (or only from `-MRD` / `-IACK`) do not move in the other cycles, and a register clocked by such a strobe takes nothing then. Paths never cross a bus twice (out of D onto R and back, say). A device's DMA drivers (the disk card's, D75) drive the backplane only in its own cycles as master: off in the CPU's, and timed in a run of their own with the slave cards answering (the CPU and the MMU / data card held).

**Phases.** The CPU card's GAL CLKGEN makes every bus and CPU clock from the 24 MHz oscillator, so the analysis knows each one's edges in the eight-phase microcycle, and for every GAL on the oscillator which inputs it listens to at which edges.

**The video card** (revision 2, the HD63484 card: D103, D106, D107) is in the machine's runs for its bus side: the decode, `-ACK` and `-WAIT`, the cycle's GAL, the CS flip-flop on `-CLK`, its own registers onto D, the command latch. Its chip and its ATmega328P have clocks of their own, so nothing is timed into or out of them here: what the chip asks of the card is checked with delays at every phase of its clock, and the memory side and the dots by YACC1-D's own check of that circuit at eight corners, both in `generators/sim_video_timing.py` (docs/video-card-acrtc.md §4.8).

**the machine at 3 MHz:**

- cpu/U3 changes only at edge 0 of the frame: GAL PANEL: the front panel's state changes at a commit, or with a button.
- the debug card's capture SRAMs are read back, and its trigger RAMs set up, only while stopped (not a path).
- the opcode mux takes R only after a FETCH (a read cycle) (not a path).
- the opcode mux takes R only after a FETCH (a read cycle) (not a path).
- the I/O card's byte devices: two clocks, -WAIT holding the first (D72) (1 more edge allowed).
- the I/O card's byte devices: two clocks, -WAIT holding the first (D72) (1 more edge allowed).
- a register-file write uses the write address (WPH selects it in phases 0-2) (not a path).
- the -IRQ lines are asynchronous to the CPU (IPEND samples them) (not a path).
- instructions come only from memory: no fetch from a device card (not a path).
- the ALU's result: read in a later microcycle than ALUGO's (D21) (1 more edge allowed).

**the machine at 4 MHz (the clock jumper):**

- cpu/U3 changes only at edge 0 of the frame: GAL PANEL: the front panel's state changes at a commit, or with a button.
- the debug card's capture SRAMs are read back, and its trigger RAMs set up, only while stopped (not a path).
- the opcode mux takes R only after a FETCH (a read cycle) (not a path).
- the opcode mux takes R only after a FETCH (a read cycle) (not a path).
- the I/O card's byte devices: two clocks, -WAIT holding the first (D72) (1 more edge allowed).
- the I/O card's byte devices: two clocks, -WAIT holding the first (D72) (1 more edge allowed).
- a register-file write uses the write address (WPH selects it in phases 0-2) (not a path).
- the -IRQ lines are asynchronous to the CPU (IPEND samples them) (not a path).
- instructions come only from memory: no fetch from a device card (not a path).

- the ALU's result: read in a later microcycle than ALUGO's (D21) (1 more edge allowed).

**the disk card's DMA cycles (3 MHz):**

- the I/O card's byte devices: two clocks, -WAIT holding the first (D72) (1 more edge allowed).
- the I/O card's byte devices: two clocks, -WAIT holding the first (D72) (1 more edge allowed).

**the debug card alone (12 MHz timing mode):**

- the timing mode samples the lines asynchronously, as a logic analyser does (not a path).
- the capture SRAMs are read back, and the trigger RAMs set up, only while stopped (not a path).
- the card listens or drives, never both (not a path).

**Crossings.** Paths between oscillators are not timed: the `-IRQ` lines are sampled by IPEND, the debug card has its own oscillator for its timing mode, the UARTs and the Picos their own clocks.

## 4. Open-drain lines: the release

An open-drain line is asserted by a driver pulling it low (timed above); it is released by its pull-up, a rise of 0.51 RC to HCT's 2.0 V:

| Line | Pull-up | Capacitance | Rise |
|---|---|---|---|
| `-ACK` | 1k | 200 pF | 102 ns |
| `-BREQ0` | 10k | 50 pF | 255 ns |
| `-IRQ0` | 2.2k | 200 pF | 224 ns |
| `-IRQ1` | 2.2k | 200 pF | 224 ns |
| `-IRQ2` | 2.2k | 200 pF | 224 ns |
| `-IRQ3` | 2.2k | 200 pF | 224 ns |
| `-IRQ4` | 2.2k | 200 pF | 224 ns |
| `-IRQ5` | 2.2k | 200 pF | 224 ns |
| `-IRQ6` | 2.2k | 200 pF | 224 ns |
| `-IRQ7` | 2.2k | 200 pF | 224 ns |
| `-LOCK` | 1k | 200 pF | 102 ns |
| `-WAIT` | 680 | 200 pF | 69 ns |

`-WAIT` must be up within half a bus clock of its release (167 ns at 3 MHz, 100 at 5 MHz, less the gates: D68); `-ACK` before the next cycle's decision; a radial `-BREQ` before the arbiter's next look.

## 5. Method and assumptions

- **Delays:** `generators/p32delays.py`, each part's datasheet and the figures used (§6). An output into a backplane line carries 200 pF, not the datasheet's 50: about 0.06 ns a pF more for HCT, 0.02 for ACT, F, LVC and the GALs, and 3 ns for the line. A radial or local line: 50 pF.
- **Clocks:** each register's clock pin is traced back to its oscillator through the most direct route (its clock input, not the decode that gates it); its insertion delay counts, so skew between cards does too.

- **The ALU's FPGA** clocks its result latches at most 200 ns after `ACLK` (D21: a division takes 18 of its clocks), and the result is read in a later microcycle.
- **Not checked here:** hold times (no datasheet gives minimum delays for most HCT parts), the FPGA's inside, metastability at the synchronizers.

## 6. The datasheets

| Part | Source |
|---|---|
| 27C256 | YACC1-D tests/video/v30_netsim.py: Atmel AT27C256R-70 doc 0014 (the font's 27C256 taken as that fast) |
| 74ABT377 | https://www.ti.com/lit/ds/symlink/sn74abt377a.pdf (BiCMOS, TTL-level outputs: no ACT / F 377 is made in DIP) |
| 74ACT00 | https://www.ti.com/lit/ds/symlink/sn74act00.pdf |
| 74ACT02 | YACC1-D tests/video/v30_netsim.py: TI CD74ACT02 |
| 74ACT04 | https://www.ti.com/lit/ds/symlink/sn74act04.pdf |
| 74ACT08 | https://www.ti.com/lit/ds/symlink/sn74act08.pdf |
| 74ACT10 | YACC1-D tests/video/v30_netsim.py: TI SN74ACT10 |
| 74ACT11 | YACC1-D tests/video/v30_netsim.py: TI SN74ACT11 |
| 74ACT157 | https://www.ti.com/lit/ds/symlink/cd74act157.pdf |
| 74ACT245 | YACC1-D tests/video/v30_netsim.py: TI SN74ACT245 |
| 74ACT273 | YACC1-D tests/video/v30_netsim.py: TI SCHS249C |
| 74ACT32 | https://www.ti.com/lit/ds/symlink/sn74act32.pdf |
| 74ACT541 | https://www.onsemi.com/download/data-sheet/pdf/74act541-d.pdf |
| 74ACT573 | YACC1-D tests/video/v30_netsim.py: TI SN74ACT573 (a transparent latch: D or LE to Q) |
| 74ACT574 | https://www.ti.com/lit/ds/symlink/sn74act574.pdf |
| 74ACT74 | https://www.ti.com/lit/ds/symlink/sn74act74.pdf |
| 74ACT86 | YACC1-D tests/video/v30_netsim.py: TI SN74ACT86 |
| 74F138 | https://www.ti.com/lit/ds/symlink/sn74f138.pdf |
| 74F151 | https://www.ti.com/lit/ds/symlink/sn74f151b.pdf |
| 74F153 | https://www.ti.com/lit/ds/symlink/sn74f153.pdf |
| 74F161 | https://www.ti.com/lit/ds/symlink/sn74f161a.pdf |
| 74F163 | https://www.ti.com/lit/ds/symlink/sn74f163a.pdf |
| 74F21 | YACC1-D tests/video/v30_netsim.py: TI SN74F21 |
| 74F253 | https://www.ti.com/lit/ds/symlink/sn74f253.pdf |
| 74F257 | https://www.ti.com/lit/ds/symlink/sn74f257.pdf |
| 74F283 | https://www.ti.com/lit/ds/symlink/sn74f283.pdf |
| 74F299 | YACC1-D tests/video/v30_netsim.py: TI SDFS071B |
| 74F521 | https://www.ti.com/lit/ds/symlink/sn74f521.pdf (the '688's pinout) |
| 74F74 | YACC1-D tests/video/v30_netsim.py: TI SDFS046A |
| 74HC05 | https://www.ti.com/lit/ds/symlink/sn74hc05.pdf (open drain: pulling down, tPHL 21; its release rises through the pull-up) |
| 74HC280 | https://www.ti.com/lit/ds/symlink/cd74hc280.pdf |
| 74HCT00 | https://www.ti.com/lit/ds/symlink/sn74hct00.pdf |

| Part | Source |
|---|---|
| 74HCT08 | https://www.ti.com/lit/ds/symlink/sn74hct08.pdf |
| 74HCT125 | https://www.ti.com/lit/ds/symlink/sn74hct125.pdf |
| 74HCT138 | https://www.ti.com/lit/ds/symlink/sn74hct138.pdf |
| 74HCT14 | https://www.ti.com/lit/ds/symlink/sn74hct14.pdf |
| 74HCT148 | https://www.ti.com/lit/ds/symlink/sn74hc148.pdf (no HCT datasheet found: the HC part's figures) |
| 74HCT153 | https://www.ti.com/lit/ds/symlink/cd74hct153.pdf |
| 74HCT154 | https://www.ti.com/lit/ds/symlink/cd74hct154.pdf |
| 74HCT157 | https://www.ti.com/lit/ds/symlink/sn74hct157.pdf |
| 74HCT161 | https://www.ti.com/lit/ds/symlink/cd74hct161.pdf |
| 74HCT163 | https://www.ti.com/lit/ds/symlink/cd74hct163.pdf |
| 74HCT175 | https://www.ti.com/lit/ds/symlink/cd74hct175.pdf |
| 74HCT244 | https://www.ti.com/lit/ds/symlink/sn74hct244.pdf (its -40..+125 °C column: no 85 °C one) |
| 74HCT245 | https://www.ti.com/lit/ds/symlink/sn74hct245.pdf |
| 74HCT253 | https://www.ti.com/lit/ds/symlink/cd74hct253.pdf |
| 74HCT257 | https://www.ti.com/lit/ds/symlink/sn74hct257.pdf |
| 74HCT273 | https://www.ti.com/lit/ds/symlink/sn74hct273.pdf |
| 74HCT32 | https://www.ti.com/lit/ds/symlink/sn74hct32.pdf |
| 74HCT374 | https://www.ti.com/lit/ds/symlink/sn74hct374.pdf |
| 74HCT377 | https://www.ti.com/lit/ds/symlink/cd74hct377.pdf |
| 74HCT393 | https://www.ti.com/lit/ds/symlink/cd74hct393.pdf (CP to Q3, the ripple's last bit) |
| 74HCT4002 | https://assets.nexperia.com/documents/data-sheet/74HC_HCT4002.pdf |
| 74HCT540 | https://www.ti.com/lit/ds/symlink/sn74hct540.pdf |
| 74HCT541 | https://www.ti.com/lit/ds/symlink/sn74hct541.pdf |
| 74HCT574 | https://www.ti.com/lit/ds/symlink/sn74hct574.pdf |
| 74HCT595 | https://www.ti.com/lit/ds/symlink/sn74hct595.pdf (its -40..+125 °C column) |
| 74HCT597 | https://www.ti.com/lit/ds/symlink/cd74hct597.pdf |
| 74HCT688 | https://www.ti.com/lit/ds/symlink/cd74hct688.pdf |
| 74HCT74 | https://www.ti.com/lit/ds/symlink/sn74hct74.pdf |
| 74HCT85 | https://www.ti.com/lit/ds/symlink/cd74hct85.pdf |
| 74LS145 | https://www.ti.com/lit/ds/symlink/sn74ls145.pdf (25 °C max only, 50; x1.2 for temperature) |
| 74LVC244 | https://www.ti.com/lit/ds/symlink/sn74lvc244a.pdf |
| 74LVC245 | https://www.ti.com/lit/ds/symlink/sn74lvc245a.pdf |
| 74LVC574 | https://www.ti.com/lit/ds/symlink/sn74lvc574a.pdf |
| AS6C1008 | YACC1-D tests/video/v30_netsim.py: Alliance AS6C1008-55 v1.2 |
| AS6C4008 | Alliance Memory AS6C4008 v1.4 (-55) |
| CD74HC4050 | https://www.ti.com/lit/ds/symlink/cd74hc4050.pdf |
| CY7C199 | Cypress 38-00239-E (CY7C199-15) |
| DS12C887 | ESTIMATE: its datasheet could not be fetched (Intel-mode read delay taken as 120) |

| Part | Source |
|---|---|
| GAL-10 | https://ww1.microchip.com/downloads/en/DeviceDoc/doc0735.pdf (ATF22V10C-10) |
| GAL-7 | https://ww1.microchip.com/downloads/en/DeviceDoc/doc0735.pdf (ATF22V10C-7; tCO 5.5 in DIP, footnote 2) |
| SST39SF010A | Microchip DS20005022C, table 11 (-55: the control store, D18) |
| SST39SF020A | Microchip DS20005022C, table 11 (-70: the boot ROM) |
| TIL311 | TI SODS001D (the latch: data 50 ns before the strobe rises and 40 ns after; the strobe low 40 ns) |
| TL16C550 | https://www.ti.com/lit/ds/symlink/tl16c550c.pdf (RD to data 45; address and CS 7 before the strobe, data 15 before WR ends) |

# Speeding up the CPU: where it can be done

docs/speed.md

**Status:** ANALYSIS (2026-10-27; Ken: "look where there can be changes to speed up cpu"). Nothing here is built. The figures come from the timing analysis (docs/timing.md, `generators/p32timing.py`) with what-if changes, and from `tools/isamix.sh` on two recorded P32W Unix sessions.

Speed is the clock times the work done per cycle. Today: **3 MHz guaranteed** (the datasheets' worst case, D72), about **4.9 MHz at typical delays**, and **3.2 microcycles an instruction**: about 0.94 MIPS at 3 MHz.

## 1. The clock

What the timing analysis says each step would give (worst case: every chip at its datasheet maximum, 85 °C; typical: 25 °C):

| Change | Worst case | Typical | Cost |
|---|---|---|---|
| As built (D72) | 3.09 MHz | 4.93 MHz | — |
| **A faster oscillator, chosen by a jumper: done, D74** | — | 4 MHz, 19% in hand (4.5 MHz ruled out: the timer's 1 MHz needs a whole divider) | a 32 MHz oscillator in a socket, a 74ACT157 and a jumper on the CPU card; the `-FAST` bus line, so the I/O card's timer divides by 4 |
| Swap every remaining 74HCT part on the failing paths for 74ACT / 74F (about 93 more chips, the device cards' decoders and the MMU / data card's muxes included) | about 3.6 MHz | — | many chips, more supply current, faster edges |
| The control store and the decode tables in 15 ns SRAM, loaded from the flash at reset | 3.09 MHz (other paths limit) | 5.50 MHz | 7 + 2 × CY7C199 and a loader (a counter and a GAL) |
| Both of the above together | 4.09 MHz | 6.44 MHz | both |
| **5 MHz at worst case** | needs the structure changed, below | | |

**Why worst case stops near 4 MHz:** one path crosses the whole machine in one microcycle, from IR through the decode table's flash (the access size) to the byte enables, the addressed card's decode and `-ACK`, the bus-error decision (FAULT), the next microaddress and the control store's flash. Two flash reads and a bus round trip don't fit 200 ns. Structural fixes:

1. **Latch the decode table's outputs with IR** (look it up during FETCH, from R, as the opcode mux already does): the size and the class are ready when the cycle starts.
2. **Decide a bus error a cycle later:** a fault latch that the next microcycle traps on, so `-ACK` no longer feeds the microaddress in the same cycle. The microcode's abort discipline (D10) has to stay exact: the faulting cycle's writes must still be cancelled.
3. **Translation a cycle earlier** (microarch.md 8): the TLB looked up as MAR loads, so the bus cycle starts with its physical address.

With those and the SRAM control store, 5 MHz is in reach at worst case; measured, the work per cycle is unchanged, so **5 MHz is 1.67× today**.

**The cheapest step by far is the jumper (D74, done):** most machines will run at 4 MHz at room temperature. `make test` can't prove that; a bench run of the test programs at the higher clock (the debug card's listen mode shows any late signal) would.

## 2. The work per cycle

`tools/isamix.sh` on two sessions (cycles without `WAIT`):

| Session | Instructions | Cycles an instruction |
|---|---|---|
| shell, files (session1) | 24.2 M | 3.20 |
| MicroPython (session15) | 17.5 M | 3.18 |

| Change | Cycles saved | Cost |
|---|---|---|
| **A DMA disk card** (the engine is on the card now, D75; the driver change is still to do) | the CPU's copy loop: `LHU` / `SH` are **12.7%** of the shell's cycles and **21.4%** of Python's (with the loop's `ADDI` / `BNE` more) | a DMA engine on the disk card, a driver change |
| **An overlapped fetch**: the next instruction fetched in an instruction's last cycle when that cycle doesn't use the bus | **11–12%** (every ALU op, `LUI`, `LI32`, a branch not taken: 3 cycles → 2) | a second path from the MMU / data card's data into IR (today the fetched word reaches IR over R, which the last cycle also uses for its result), the shadow PC ahead by one, the microcode's END words; exceptions must stay precise |
| Leaf calls with a link register (`JAL` / `JR lr`) | 1.3% | an ISA and compiler change |
| Post-increment used more by the compiler | 1.2% | compiler |
| Compare with a constant and branch | 0.7% | ISA, compiler |
| `LDM` / `STM` | 0.6% | a register-list sequencer |

The ISA additions left over from D59 together are under 4%; the DMA disk and the overlapped fetch are the two that matter.

## 3. In order of value for the effort

1. **A clock jumper** (4 MHz besides 3): +33%, a part or two. **Done: D74.**
2. **DMA on the disk card:** 13–21% fewer cycles where the disk is busy. **The hardware is built in (D75)**; what remains is the driver.
3. **The control store in SRAM:** typical 5.5 MHz; a step towards 5 MHz worst case.
4. **The structural changes for 5 MHz at worst case** (§1, 1–3): 1.67×.
5. **The overlapped fetch:** 11–12%, the largest datapath change of these.

Together: 5 MHz with the fetch overlapped and DMA is about **2× today's speed** (1.67 × 1.13 × about 1.1), about 2 MIPS.

## 4. The benchmark: a clean make of the commands

Software speed is scored on one job (Ken, 2026-11-05): `make clean; make` in `/usr/src/cmd`, every command rebuilt on P32W itself, all 101 byte for byte the host's. Long and repeatable, and it uses the kernel, the disk and the compiler together. To run it, on a copy of the disk (`emulators.md` §4.7):

```
cd /usr/src/cmd
cycles sh -c "make clean; make"
```

The times are the processor's alone, at 3 MHz, with no drive access time.

| Date | Change | Cycles | Time | User | Sys | Disk commands |
|---|---|---|---|---|---|---|
| 2026-11-05 | the baseline (D85) | 26,214,809,902 | 2:25:38 | 7,832 s | 890 s | 76,518 |
| 2026-11-05 | `dirlookup` a block at a time, `ls` keeps the last names (D87) | 25,805,258,346 | 2:23:22 | 7,838 s | 751 s | 76,513 |
| 2026-11-05 | `ctype` macros (D89) | 23,480,869,849 | 2:10:27 | 7,062 s | 752 s | 76,515 |
| 2026-11-05 | `putc` into the buffer (D89) | 22,687,214,493 | 2:06:02 | 6,789 s | 762 s | 76,770 |
| 2026-11-05 | `as`: the first character before `strcmp` (D89) | 21,746,096,713 | 2:00:49 | 6,478 s | 759 s | 76,778 |
| 2026-11-05 | `cc`'s lexer: the first character before `strncmp` (D91) | 19,330,720,936 | 1:47:24 | 5,672 s | 761 s | 76,745 |
| 2026-11-05 | `cc`'s `is`: the same (D91) | 18,514,042,450 | 1:42:51 | 5,399 s | 761 s | 76,590 |
| 2026-11-05 | `strlen`, `strcmp`, `strncmp`, `strchr` in assembly (D91) | 17,873,170,257 | 1:39:18 | 5,192 s | 756 s | 76,580 |
| 2026-11-05 | `cc`: integer constants in 32 bits (D93) | 16,963,919,772 | 1:34:15 | 4,887 s | 758 s | 76,474 |
| 2026-11-05 | the clock and scheduler walk only the slots in use (D93) | 16,396,397,709 | 1:31:06 | 4,719 s | 738 s | 76,524 |
| 2026-11-05 | `ialloc` starts at the lowest free inode (D93) | 16,186,841,412 | 1:29:56 | 4,717 s | 671 s | 64,425 |
| 2026-11-05 | `ld` reads only the archive members it takes (D93) | 15,846,258,991 | 1:28:02 | 4,675 s | 598 s | 47,367 |
| 2026-11-05 | `as`: one output buffer, operands from blocks (D93) | 15,011,735,330 | 1:23:24 | 4,396 s | 599 s | 47,328 |
| 2026-11-05 | runs of text in `printf`, `fputs` and `gen.c` (D93) | 14,391,050,348 | 1:19:57 | 4,194 s | 596 s | 47,576 |
| 2026-11-05 | `make` without `sh`, spare page tables (D93) | 14,157,678,312 | 1:18:39 | 4,193 s | 519 s | 46,838 |
| 2026-11-06 | `cc`: token tests inline, one call for a binary expression (D94) | 13,000,164,799 | 1:12:13 | 3,805 s | 522 s | 46,730 |
| 2026-11-06 | `cc`: punctuators by first character, `VPUSH` (D94) | 12,352,459,311 | 1:08:38 | 3,587 s | 524 s | 46,716 |
| 2026-11-06 | `printf`: plain `%s` and `%d` first (D94) | 11,912,110,108 | 1:06:11 | 3,434 s | 530 s | 46,868 |
| 2026-11-06 | `as`: the opcode once, registers, comments, case (D94) | 10,925,984,911 | 1:00:42 | 3,106 s | 530 s | 46,969 |
| 2026-11-06 | `as`: plain numbers skip the evaluator (D94) | 10,396,524,636 | 0:57:46 | 2,931 s | 529 s | 46,930 |

The user programs were 90% of the baseline, and those are mostly `cc` and `as`. For `sh.c` (1,654 lines, 12,256 of assembly) `cc -c` took 346 s, about 8 lines of C a second: 203 s to compile and 142 s to assemble. With D89 it takes 277 s, with D91 233 s, with D93 183 s and with D94 129 s. The link takes 7.5 s. (`cc -S` is slower, 638 s at the baseline, because it copies each statement's source line into the assembly, rescanning the file each time. A build does not use it.)

The profile (`p32uemu -G -q`, `tools/uprof.py`, D88) was flat, with no one function to blame. In the compile, writing the assembly out as text (`fputc`, `format`, `strlen`, `put`, `udivmod`, `fputs`, `towrite`) was about a third. In `as`, the `ctype` functions were 21% and `strcmp` 15%. Those are gone now, and the largest part left is structural: `cc` turning everything into text (a quarter of its time) and `as` parsing it back. Together they are about half of all the compile's time, and `cc` writing objects itself would be a redesign.

## What a program costs before its own code (D92)

A program that does nothing is the floor every command in the benchmark stands on, and it is far above zero. At 3 MHz, with the code of D91:

| `cc -o x x.c` of | Compile | Assemble | Link | In all |
|---|---|---|---|---|
| one line, no headers | 1.8 s | 1.1 s | 4.6 s | 7.1 s |
| `hello.c` | 4.3 s | 1.2 s | 4.6 s | 9.6 s |
| an empty `main`, seven common headers (185 lines after `cpp`) | 9.5 s | 1.1 s | 4.6 s | 14.8 s |

The last row, times 101, is 25 minutes, a quarter of the benchmark. `cc`'s own start is 0.13 s of user time: the cost is loading programs, reading headers and linking. For the smallest command the kernel is 72% of the time. D93 brings the three rows down to 4.1, 6.6 and 11.6 s, and D94 to 4.2, 5.9 and 9.6 s. Headers, about 5.5 s a program, are still the largest

part, a quarter of that in `malloc`.

Over the whole benchmark the kernel takes **21.4% of the cycles** (3.8 billion of 17.9), where `time` says 12.7%. `time` counts 50 Hz ticks, and a tick cannot land in a short trip with interrupts off, such as a TLB refill (`trapvec`, 3.6%). By function, `uprof.py -k`:

| Function | Share of all cycles | What it is |
|---|---|---|
| `trapvec` | 3.6% | mostly the TLB refill, in software |
| `cf_in` | 3.1% | the disk's sector copy, 2,440 cycles a sector |
| `clockintr` | 2.5% | walks all 64 process slots, 50 times a second |
| `scheduler` | 1.8% | the same walk, at every switch |
| `memset` | 1.7% | zeroing pages |
| `memcpy` | 1.0% | |
| `bget` | 0.9% | the buffer cache's list |

About 90% of the 230,662 sectors read are three things, by the files' sizes and the count of files made: `sh`, `cc`, `as` and `ld` loaded again for every program (about half), `libc.a` read whole for every link (a quarter), and `ialloc`, which scanned the inode table from the start for every file created (a sixth). D93 dealt with the last two: 154,166 sectors now, and 46,838 disk commands, not 76,580. The first is still there: a cache of programs would take most of it.