HP LaserJet IIIP on Acorn guests: printing bug handover

MAME PR #16792 (`hplj3p`), Acorn Archimedes guests (aa310 / aa500 / aa680 / aa5000). Written 16 Nov 2026.

Status in one paragraph

Two separate things were mixed together. (1) The "reversed text" was not a bug: BBC BASIC `PRINT#` writes a binary string record, not plain text. (2) The real fault is a handshake race: `hp_lj3p` acknowledges a byte (`/ACK`) within microseconds of `/STROBE`, before the Archimedes interrupt handler has finished, so the guest loses the ACK and stalls. A 100 µs ACK delay fixed the VDU test (`ABCDEFGH` now prints) but a RISC OS desktop print still stalls and needs ON LINE toggled repeatedly. **The bug is partially fixed, not closed.** A refined patch is proposed below and has not been tested.

Corrections to earlier notes

The earlier handover (a310-print-bug-handover-1.txt) and some early advice in this investigation were wrong on these points:

- **"Reversal is a LIFO bug in hp_lj3p / ctronics"**: wrong. The guest writes the bytes reversed. See finding 1.
- **"Strobe polarity / bit order / aa310.cpp is at fault"**: all ruled out. The aa5000 fails identically through a completely different path (`upc82c710`), and the stock `aa310.cpp` handlers are fine.
- **"BUSY/ACK level-triggered wiring starves the device"**: not the cause. The IOC latches ACK correctly (see finding 3).
- The tick-timer rate (600 Hz, changed recently in the PR) is almost certainly irrelevant: handshake timing is in tens of µs, the tick is 1.67 ms. Not formally tested (see "Cheap experiments").

Findings

1. The reversed text is BASIC's `PRINT#` file format

`PRINT#ch%, "ABCDEFGH"` sends a string record: type marker `&00`, length byte, then the characters **in reverse order**. The logged bytes match exactly:

```
00 08 48 47 46 45 44 43 42 41    string, length 8, "HGFEDCBA"
00 01 0c                         string, length 1, CHR$(12)  (reversing 1
char changes nothing)
```

That is why the log showed the bytes already reversed at `peripheral5_w`, and why "control bytes arrived in the correct position". Use `VDU 2 : PRINT "..." : VDU 3` or `BPUT#ch%, byte` for plain text. Nothing needs fixing here.

2. The ACK race (root cause of "only A")

With the stock `aa310.cpp` and plain-text output (`VDU 2 : PRINT "ABCDEFGH" : VDU 12 : VDU 3`), only `A` printed. Instrumented log (times in seconds; the two CPUs keep separate local times, so ordering within a few µs is approximate):

```
[57.339074] DATA 41
[57.339082] BUSY 1
[57.339134] ACK  0      <- ~52 us after strobe (healthy)
[57.339136] BUSY 0
[57.339139] ACK  1
[57.339171] DATA 42
[57.339179] BUSY 1
[57.339177] ACK  0      <- at/before the strobe: ~2-6 us after DATA
[57.339179] BUSY 0
[57.339182] ACK  1
(no DATA 43; guest waits forever)
```

The RISC OS parallel driver is interrupt-driven: it sends the next byte from inside its ACK handler. Byte 2 was sent from inside that handler, and its ACK arrived a few µs later, before the handler finished and wrote its clear to the IOC's IRQ A request register. That clear wiped the freshly latched ACK. The driver then waits for an edge that has already happened. Pressing ON LINE makes the printer pulse ACK again, which releases exactly one more byte. PCs, MSX, Amiga etc. poll BUSY and never need ACK, so only Acorn guests expose it.

Cause in code (`hp_lj3p.cpp` , original `pctrl_w`): when the firmware writes bit 7 of the parallel control register, the device immediately did `m_pfull = false; m_ack_cb(0);` and a 5 µs timer raised `/ACK` again. The 68000 firmware is often already waiting in its IRQ2 handler, so this happens within microseconds of the strobe. A real IIIP takes much longer.

3. The IOC is correct (acorn_ioc.cpp / .h reviewed)

- `if_w` sets IRQ A bit 2 on a falling edge only, latched until the guest clears it (write to `IRQ_REQUEST_A` , mask `0x7c`).
- `il6_w` (BUSY) is a level mirror into IRQ A bit 0; the guest cannot clear it.
- Minor, unrelated: `device_reset` sets `m_if = CLEAR_LINE` (0) although idle `/ACK` is high; the first ACK after reset might not register as an edge if the printer's reset-time `ack(1)` ran earlier. Watch for "first byte after reset hangs".

4. Handler and wiring in aa310.cpp are stock and fine

`case 0x10` writes `m_cent_data_out` directly; `case 0x18` passes `BIT(data, 4)` to `write_strobe()` ; config wires `busy_handler` to `il6_w` and `ack_handler` to `if_w` . The aa310.cpp uploaded as `aa310_1.cpp` was verified identical to master (2103 lines, no hack). (An earlier upload, `aa310.cpp` , still contained the rejected buffer-and-reverse hack; it was replaced.)

Fix applied so far (partial)

In `hp_lj3p.cpp` , delay the ACK after the firmware frees the latch:

```
TIMER_CALLBACK_MEMBER(hp_lj3p_device::ack_end)
{
    if (param)
    {
        m_ack_cb(0);
        m_ack_timer->adjust(attotime::from_usec(5), 0);
    }
    else
        m_ack_cb(1);
}

// pctrl_w()
if (BIT(m_pctrl, 7) && !BIT(old, 7))
{
    m_pfull = false;
    m_ack_timer->adjust(attotime::from_usec(100), 1);   // value was my
suggestion; the exact delay used locally should be recorded
}
```

Result: the VDU test prints `ABCDEFGH`. A RISC OS desktop print now outputs pages but stalls repeatedly; ON LINE must be toggled many times, and the most seen was about 2.5 lines of text before the PNG is made and printing continues. (The `hp_lj3p.cpp` file uploaded in this session is the version *before* this change.)

Remaining problem and hypothesis

Because the fix above uses one timer for both the delay and the pulse, and BUSY still drops as soon as the firmware frees the latch, ACKs can still overlap. If the driver sends the next byte as soon as BUSY goes low (the IOC mirrors BUSY into IRQ A bit 0), the next ACK request re-arms `m_ack_timer` over a pending ACK (or a pulse in progress, leaving `/ACK` low with no new falling edge). Two bytes then produce one ACK edge, the guest waits, and ON LINE restarts it. This is a hypothesis; it has not been tested and I have not seen the RISC OS driver's ACK handler.

Proposed next patch (untested)

Release BUSY and the ACK pulse together, after the delay, so only one handshake is outstanding. Keep `m_pfull` clearing immediately: `ctrl2_r` exposes it as bit 10 ("latch full") and `update_parallel` uses it for IRQ2, so holding it would re-trigger the interrupt

and re-read the same byte. This needs a new member in `hp_lj3p.h` (not seen in this session) next to `m_pfull`:

```
bool m_ack_hold;   // /ACK pending: the host still sees BUSY
```

Initialise to false in `device_reset` (and `save_item(NAME(m_ack_hold))` if wanted). Then:

```
// update_parallel()
m_busy_cb((m_pfull || m_ack_hold || !BIT(m_pctrl, 15)) ? 1 : 0);

// pctrl_w()
if (BIT(m_pctrl, 7) && !BIT(old, 7))
{
    m_pfull = false;        // firmware sees the latch free at once
    m_ack_hold = true;      // host still sees BUSY
    m_ack_timer->adjust(attotime::from_usec(100), 1);
}

// ack_end()
if (param)
{
    m_ack_hold = false;
    update_parallel();                // BUSY drops now...
    m_ack_cb(1);                      // make sure /ACK is high first
    m_ack_cb(0);                      // ...together with the /ACK pulse
    m_ack_timer->adjust(attotime::from_usec(5), 0);
}
else
    m_ack_cb(1);
```

Diagnostics to run next

All debug lines are temporary and must be removed before any commit. `osd_printf_info` prints without `-verbose`. For ordered output use `logerror` with `-log` (writes `error.log`).

1. During a desktop print, log `DATA` (aa310.cpp `case 0x10`), `ACK` / `BUSY` (lambdas on the `busy_handler` / `ack_handler` in the machine config) and count `DATA` against `ACK 0` up to the first stall. Fewer ACKs than bytes supports the overlap hypothesis.

2. In `hp_lj3p.cpp` `strobe_w`, add a `LOG` for the currently silent dropped-byte case (strobe falling while `m_pfull` is true). Dropped bytes would show as missing characters rather than stalls.
3. To confirm the guest-clears-ACK race directly: uncomment `//#define VERBOSE 1` in `acorn_ioc.cpp` and filter `error.log` for `IOC W 05`, `DATA`, `ACK`, `BUSY`. Look for the write of `04` (clear ACK bit 2) landing *after* an `ACK 0`.

Cheap experiments

- Try the ACK delay at 10 µs, 20 µs, 50 µs and 1 ms. Record the smallest value that works reliably, and choose a comfortable multiple above it. A thin margin over a timing race is fragile.
- Temporarily set the system tick back to the old rate (`32_MHz_XTAL / 192000`, about 167 Hz). If the stall is unchanged, the tick is innocent. Better: bisect against the commit before "Run the system tick at 600 Hz".

Test matrix

Working before the changes (reported): Tandy Model 100, Amiga 1200, Amiga 500, maclc475, pcipc, ibm5170, Atari ST, MSX (hx21), apple2e (BASIC with DOS system disk), NEC PC-6001, NEC PC-8801. Not working: aa5000 / a310 (this bug), trs80 and ti99_4a (no usable parallel port).

After any fix retest: the Acorn machines (VDU test and a RISC OS desktop print on a310 and aa5000), then at least ibm5170, hx21, Amiga, maclc475 and TRS-80 Model 100. Hosts that watch ACK are the likeliest to react to the ACK change.

Useful tests on Acorn:

```
VDU 2 : PRINT "ABCDEFGH" : VDU 12 : VDU 3

ch% = OPENOUT("printer:")
FOR i% = 1 TO 8 : BPUT#ch%, 64+i% : NEXT
BPUT#ch%, 12
CLOSE #ch%
```

To make the single-pass PNG appear without touching the panel, a form feed (12) must be sent.

Files

- `src/devices/machine/hp_lj3p.cpp` (and `.h`, not seen): handshake lives in `strobe_w`, `pctrl_w`, `update_parallel`, `ack_end`, `ctrl2_r` (latch-full bit)
- `src/mame/acorn/aa310.cpp`: stock handlers at `peripheral5_w` cases `0x10` / `0x18`; config around line 1167 (aa500), 1273 (aa310) wires BUSY to `il6_w` and ACK to `if_w`
- `src/devices/machine/acorn_ioc.cpp` / `.h`: `if_w` (falling-edge latch), `il6_w` (level), `registers_w` `IRQ_REQUEST_A` (guest clear)
- `src/devices/machine/upc82c710.cpp`: aa5000 parallel path (not changed, same symptoms, so it is not at fault)

Housekeeping before committing

- Revert every temporary print (the `DATA` / `BUSY` / `ACK` lines in `aa310.cpp` and anything added to the printer). `aa310.cpp` should end up identical to master.
- This work follows the PR's convention of disclosing AI-written code in commit messages. Changes proposed here were written with Claude; say so in the commit.
- Suggested commit message: *hp_lj3p: Delay /ACK after the firmware frees the data latch.* Body: ACK was pulsed within microseconds of /STROBE, before the Archimedes IRQ handler had finished, so the IOC's latched ACK was cleared and the driver stalled; release BUSY and pulse /ACK together after a delay.

Open questions

- What delay value is in your local `hp_lj3p.cpp`, and did you test other values?
- How many bytes is a desktop print sending before the first stall (DATA vs ACK count)?
- Does RISC OS send the next byte on BUSY low (IRQ A bit 0) or only on ACK? The answer decides how much BUSY ordering matters.
- Is the `/ACK` pulse width (5 µs) acceptable to other hosts, and what does a real IIIP do?