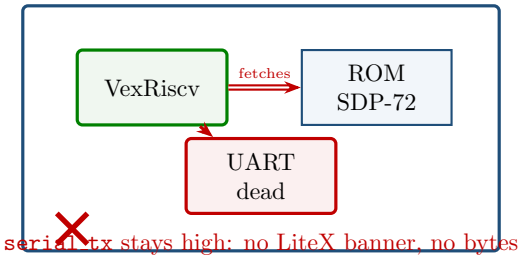
The Dead VexRiscv

A bitstream-configuration bug in the openXC7 toolchain

How a dead serial console on the Arty S7 was traced to two missing BRAM width bits — and fixed



August 17, 2026

Contents

1 The system under test

1.1 Symptom

The `demo-projects/litex-ddr-arty-s7` design — a LiteX SoC with a minimal VexRiscv CPU running on a Digilent Arty S7-50 (`xc7s50csga324-1`) — produced a completely *dead serial console* when built with the openXC7 toolchain, at every system frequency tested, down to 16 MHz. No LiteX BIOS banner, no prompt, no bytes at all. The identical RTL, built with Vivado 2024.1, booted and printed normally.

The symptom, precisely

> "Even at 16 MHz the openXC7 toolchain delivers a dead serial console, while the Vivado build works."

`serial_tx` never toggles: the CPU either never runs, runs garbage, or never reaches the UART driver. The task: find the *root cause*.

1.2 The design

For the comparison the DDR3 controller had been removed and the CPU reduced to the *minimal* VexRiscv configuration, leaving a small, easy-to-analyze SoC:

- **Clocking** — a single `PLLE2_ADV` multiplies the 100 MHz board oscillator to a 1600 MHz VCO and divides it back down to a 16 MHz system clock ($CLKFBOUT\_MULT = 16$, $DIVCLK\_DIVIDE = 1$, $CLKOUT0\_DIVIDE = 100$).
- **Reset** — an 8-stage `FDCE` shift chain on the input clock generates the PLL reset; two `FDPE` flip-flops with asynchronous preset, clocked by the PLL output and preset by $\overline{\texttt{LOCKED}}$, generate `sys_rst`.
- **CPU** — VexRiscv "minimal" (RV32I + Zicsr), reset vector 0x00000000, wishbone instruction and data buses.
- **Memories**
  - *ROM* — the LiteX BIOS, 6167 × 32 bits, inferred as **six `RAMB36E1` in SDP mode with a 72-bit read port**.
  - *register file* — the CPU's 32 × 32 register file, inferred as **two `RAMB18E1` in SDP mode** (36-bit read, 36-bit write).
  - *main RAM* (32 KiB) — eight `RAMB36E1` in TDP mode, 4-bit ports; *SRAM* (8 KiB) — two `RAMB36E1`, 18-bit ports.
- **I/O** — a LiteX UART (115200 baud), four LEDs driven by a LiteX LED chaser.

1.3 Two toolchains, one netlist

Both builds start from the *same* LiteX-generated top-level netlist (`digilent_arty_s7.v`) and the same constraints (`digilent_arty_s7.xdc`) and BIOS image (`digilent_arty_s7_rom.init`). The only difference is the implementation toolchain:

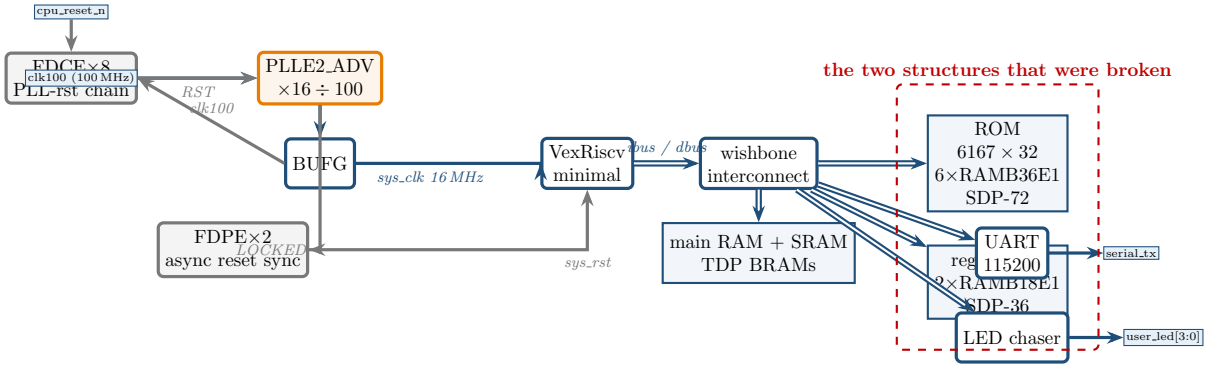


Figure 1: The LiteX SoC on the Arty S7. Clocking and reset are standard; the interesting parts are the two SDP (simple-dual-port) BRAM structures — the BIOS ROM and the CPU register file — highlighted in red.

1.4 Why the bug was hard to see

Two properties made this bug particularly well hidden:

1. **Simulation said the design was fine.** The RTL boots in Verilator, and even the fully-synthesized yosys netlist boots when simulated against Xilinx's own unisim models. The bug lives entirely in *configuration bits* of the final bitstream that no functional simulation exercises (Section 4.6).

2. **The symptom has many plausible causes.** "CPU prints nothing" could be a broken PLL, a stuck reset, a corrupted ROM, a synthesis mistake, a routing bug . . . each hypothesis had to be eliminated with direct evidence rather than guesswork.

2 Methodology: bitstream archaeology

The core idea of the whole investigation is disarmingly simple: the working Vivado build and the broken openXC7 build implement the *same netlist*, so their bitstreams should be comparable. Instead of guessing, decode both bitstreams into a human-readable *feature* representation and diff them, component by component.

2.1 Decoding bitstreams with the prjxray database

The openXC7 project maintains the *prjxray database*: a machine-readable description of every configuration bit of the Series-7 fabric, produced by fuzzing real silicon. Given that database, a bitstream can be converted into *FASM* — a text file listing symbolic features such as

```
BRAM_L_X30Y35.RAMB18_Y0.READ_WIDTH_A_18
```

which reads as: "in BRAM tile `X30Y35`, RAMB18 half `Y0`, the read-width-A marker `18` is set".

2.2 Three kinds of comparison

1. **Diff against the working build.** Decode the Vivado bitstream and the openXC7 bitstream and compare features per component (PLL tile, clock tree, BRAM tiles, I/O). Placement differences are noise; *configuration* differences are signals.

2. **Diff against a purpose-built golden.** Where the Vivado build uses a different structure than the openXC7 build (it does, for the ROM: Vivado inferred 4-bit TDP BRAMs while yosys inferred 72-bit SDP BRAMs), synthesize a tiny one-cell design with the *exact* configuration under test, build it in Vivado, decode it, and use it as the reference.

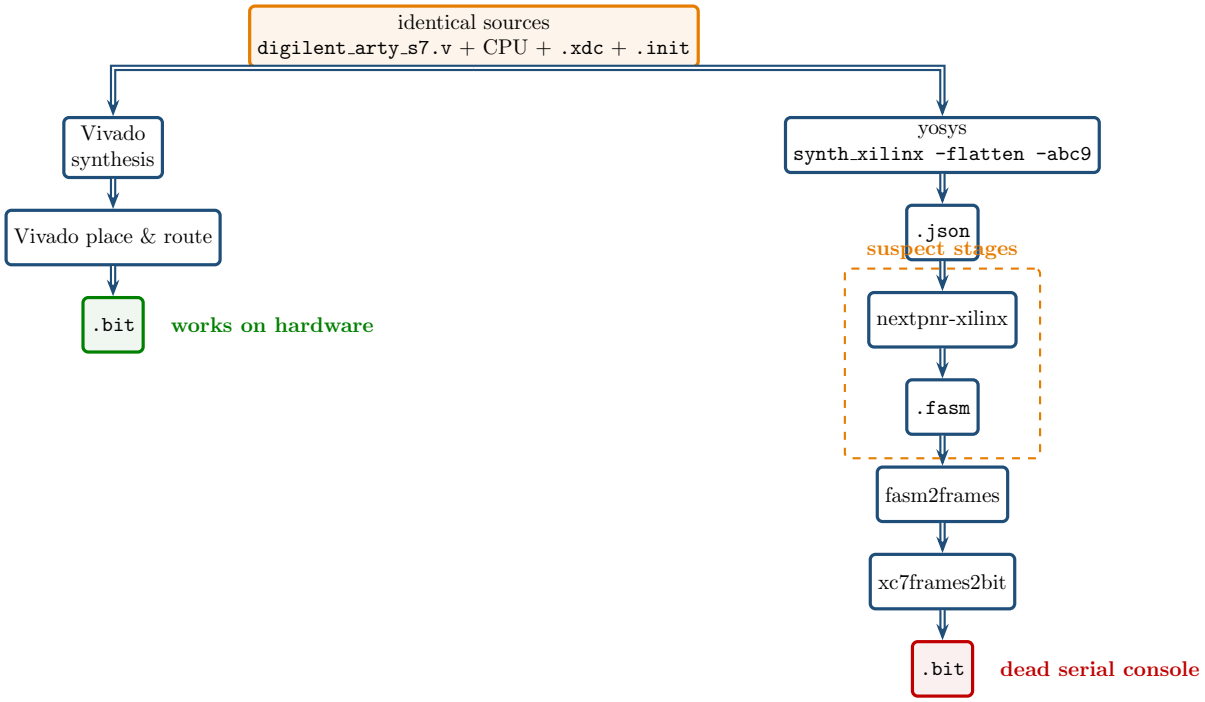


Figure 2: The two implementation flows. With identical inputs, a difference in behavior must originate in the implementation — the prime suspects are the openXC7 place-and-route and bitstream-generation stages (orange).

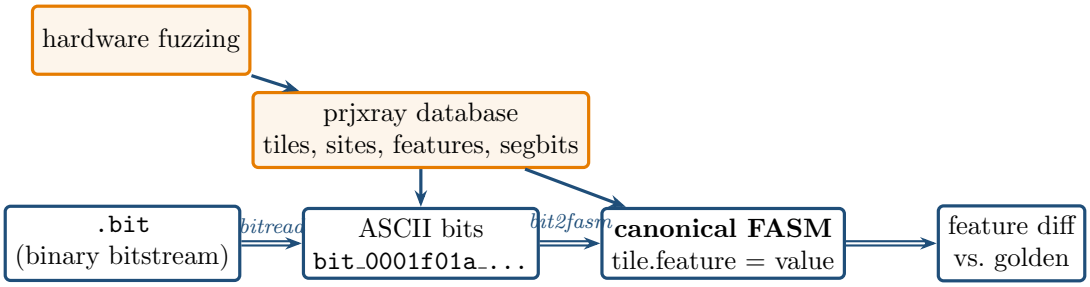


Figure 3: The decoding pipeline: a binary bitstream becomes a list of symbolic features through the fuzzing-derived prjxray database. Both bitstreams are decoded with the *same* database, so their feature lists are directly comparable even though the tools that created them differ.

3. **Prove content, not just shape.** For memory contents, reconstruct the stored data from the decoded features and compare it with the known-good input (`rom.init` / the yosys INIT parameters).

2.3 Simulation as a second front

In parallel, every intermediate artifact was executable: the RTL, and even the yosys-synthesized netlist (written back to Verilog), can be run in a logic simulator with behavioral models of the Xilinx primitives. A *booting* simulation (the LiteX BIOS prints a banner, i.e. `serial_tx` starts toggling about 450 mus after reset) is a strong statement that an artifact is *functionally* correct — but, as this bug shows, it says nothing about bitstream-level configuration.

3 The trail of hypotheses

Diagnosis proceeded hypothesis by hypothesis. Each candidate root cause was tested with direct evidence; four attractive hypotheses were eliminated before the real one was found.

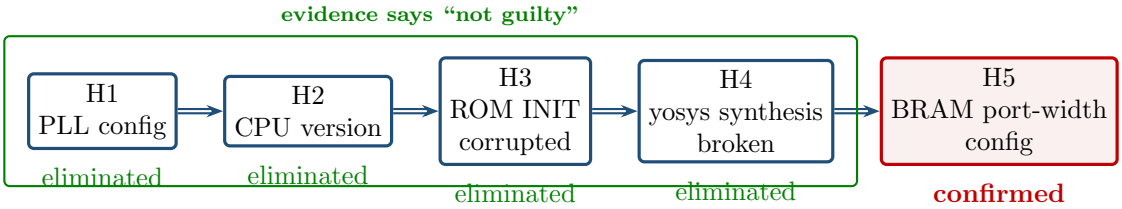


Figure 4: The hypothesis chain. Four plausible culprits were cleared with direct evidence; the fifth — a configuration-bit error in the BRAM tiles — explained every observation.

3.1 H1: the PLL is misconfigured (eliminated)

The design gates `sys_rst` on the PLL's `LOCKED` output, so a PLL that fails to lock would hold the whole SoC in reset — exactly the observed symptom. The openXC7 flow programs the PLL's loop-filter and lock tables from its own tables, so an error there seemed plausible.

Test. Decode the PLL tile (`PLLE2_ADV` feature block) of both bitstreams and compare every configuration feature.

PLL parameter	meaning	Vivado bits	openXC7 bits
DIVCLK_DIVIDE	pre-divider	1 (no-count)	1 (no-count)
CLKFBOUT_MULT	VCO multiplier	16 (high 8 / low 8)	16 (high 8 / low 8)
CLKOUT0_DIVIDE	sys-clk divider	100 (50 / 50)	100 (50 / 50)
COMPENSATION	feedback mode	`Z_ZHOLD`	`Z_ZHOLD`
`LKTABLE[39:0]`	lock detector	`FFE71FA401`	`FFE71FA401`
`TABLE[9:0]`	loop filter	`3D4`	`3D4`
`FILTREG1_RESERVED`	filter config	`08`	`08`

Table 1: PLL configuration: bit-identical between the working Vivado build and the broken openXC7 build. VCO = 100 MHz · 16 = 1600 MHz, CLKOUT0 = 1600/100 = 16 MHz.

The clock tree matched as well: the same `CLKFBOUT2IN` feedback loop, the same `CLKOUT0` → `CLKPLL0` → BUFG route, and the same global-clock distribution. **H1 eliminated.**

3.2 H2: the two builds use different CPU netlists (eliminated)

A source-parity check surfaced a real difference: the openXC7 Makefile compiled `vexriscv/VexRiscv_Min.v` (generated by SpinalHDL 1.6.0 in 2022), while the Vivado build read LiteX's bundled `VexRiscv_Min.v` (SpinalHDL 1.9.4, 2024). An old CPU netlist with a latent bug would produce exactly this symptom, toolchain-independent.

Test. Simulate *both* CPU versions in the full SoC:

Both boot and print. (Along the way, an Icarus-Verilog artifact — an *always @(*)* sensitivity mis-evaluation that held the bus decoder at zero — briefly produced a false "hang"; Verilator, used as a second opinion, exposed it as a simulator artifact, a reminder to cross-check simulators.) **H2 eliminated.**

3.3 H3: the ROM contents are corrupted (eliminated)

The next suspect: the BIOS image stored in the BRAM initialization bits. If yosys or nextpnr mis-encodes the INIT values, the CPU fetches garbage from a perfectly healthy clock/reset system.

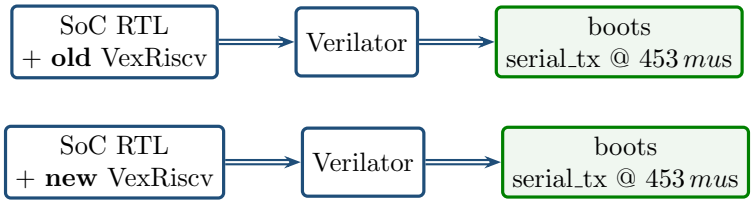


Figure 5: Both CPU netlists execute the BIOS and drive the UART identically in simulation. The version difference is a source-parity wart, not the cause.

Test. Reconstruct the ROM contents from the openXC7 bitstream and compare them bit-for-bit with yosys's INIT parameters. This required solving a small puzzle first: how the 256-bit INIT parameters of a `RAMB36E1` map onto the INIT features of its two `RAMB18` halves in the decoded FASM.

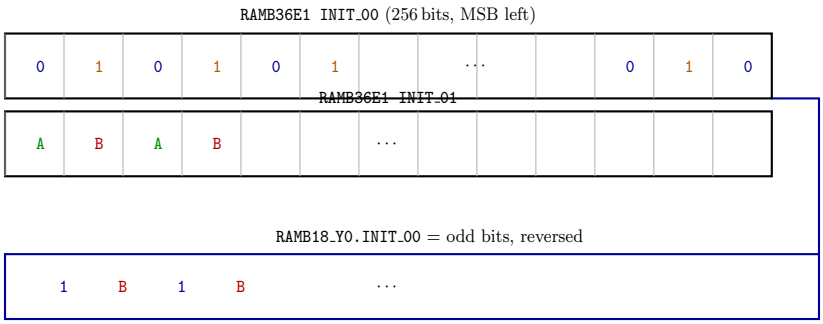


Figure 6: The RAMB36-to-RAMB18 INIT interleave: the decoded `RAMB18_Y0.INIT_XX` feature is the odd bits of the RAMB36 `INIT_XX`/`INIT_XX+1` parameters (LSB-first). The transform was first *validated* on the working Vivado build — where the parameters and the decoded bits are both known — and then applied to the openXC7 data.

Result. All 144 INIT/INITP parameters of the six ROM cells matched the openXC7 bitstream bit-for-bit. The BIOS is stored correctly. **H3 eliminated** — but the exercise produced two durable assets: the validated INIT transform, and a proven method for content verification.

3.4 H4: yosys synthesis breaks the logic (eliminated)

With the clock, the reset chain, the CPU netlists and the ROM contents all cleared, the next suspect was the synthesized logic itself: maybe *synth_xilinx* or ABC9 mis-compiles the CPU or the wishbone interconnect.

Test. Write the synthesized netlist back to Verilog and simulate it against *Xilinx's own* unisim behavioral models (not nextpnr's view of the world, but Vivado's):

Figure 7: The yosys-synthesized netlist executes the BIOS correctly when simulated against the unisim models — the same models whose semantics Vivado relies on. Synthesis is exonerated.

The netlist boots. **H4 eliminated.** By elimination, the defect had to live in the place-and-route/bitstream stage — and the decoded FASM already showed a suspicious asymmetry in the BRAM tiles (section 4).

4 Root cause: SDP BRAM port widths

4.1 What the decoded bitstreams showed

Decoding the BRAM tiles of both bitstreams revealed a configuration difference in exactly the two structures that matter for the CPU. In SDP (simple-dual-port) mode, the 36/72-bit data path is built from *both* halves of the memory, so the "opposite side" of the port — the side yosys leaves unconfigured (parameter 0) — must still be programmed wide in the bitstream:

Structure	config feature	Vivado golden	openXC7 (broken)	fixed
ROM 6×RAMB36E1 SDP, read 72	`READ_WIDTH_A` (each half)	`_18`	`_18`	`_18`
	`READ_WIDTH_B` (each half)	`_18`	`_1`	`_18`
	`SDP_READ_WIDTH_36`	set	set	set
	`WRITE_WIDTH_A/B`	`_1` / `_1`	`_1` / `_1`	`_1` / `_1`
regfile 2×RAMB18E1 SDP, R/W 36	`READ_WIDTH_A` (Y0 / Y1)	`_1` / `_18`	`_1` / `_18`	`_1` / `_18`
	`READ_WIDTH_B` (each half)	`_18`	`_1`	`_18`
	`WRITE_WIDTH_A` (each half)	`_18`	`_1`	`_18`
	`WRITE_WIDTH_B` (each half)	`_18`	`_18`	`_18`
	`SDP_READ/WRITE_WIDTH_36`	set	set	set

Table 2: The smoking gun. Three features were programmed as width `_1` where hardware needs `_18`. Red cells: the defect. The "Vivado golden" column comes from purpose-built one-cell Vivado designs (section 4.4).

4.2 Why width-1 kills the ROM reads

The ROM cells are `RAMB36E1` in SDP mode with $READ\_WIDTH\_A = 72$: each address returns a 72-bit word that is assembled from *four* 18-bit data paths — the A and B read ports of both RAMB18 halves.

4.3 The register file suffers the same defect

The CPU register file is a `RAMB18E1` in SDP mode with a 36-bit read and a 36-bit write. Both ports span the A *and* B sides of the memory: the read is `DOADO/DOPADOP` (lower 18) plus `DOBDO/DOPBDOP` (upper 18); the write is `DIADI/DIPADIP` (lower 18) plus `DIBDI/DIPBDIP` (upper 18).

4.4 How the defect was proven: purpose-built goldens

The Vivado build could not serve as the reference for these two structures, because Vivado had inferred *different* memory shapes (4-bit TDP ROM cells, a RAMB36-based register file). The decisive step was to build tiny one-cell Vivado designs that instantiate *exactly* the yosys configurations, and decode their bitstreams:

4.5 The minimal reproducer

Alongside the goldens, the defect was demonstrated from the other side with a nine-line Verilog design (`tmp/bramtest/top.v`) that provokes exactly the yosys inference the LiteX ROM relies on: a read-only 6167×32 block RAM (the LiteX ROM's depth, so yosys chooses the same SDP mapping):

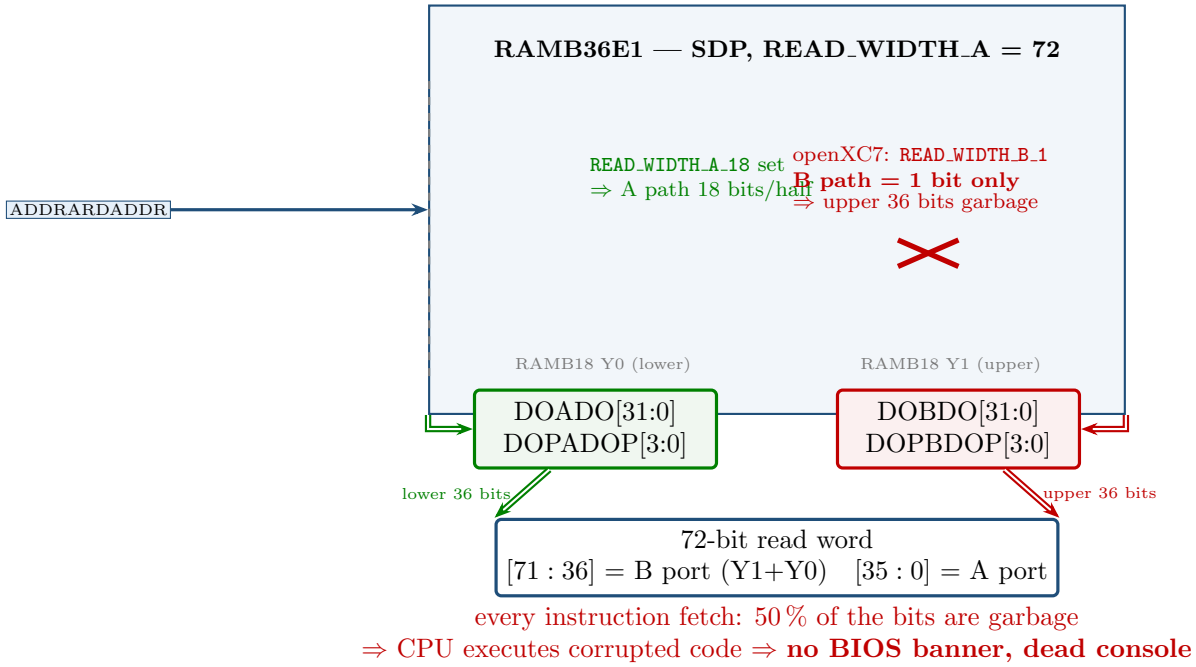


Figure 8: The SDP-72 ROM read. Hardware assembles each 72-bit word from the A port (lower 36 bits) *and the B port* (upper 36 bits). The openXC7 bitstream configured the B-side read path 1 bit wide, so the upper half of every fetch was garbage — the CPU never decoded a valid instruction.

```
module top(input clk, input [12:0] addr, output [31:0] q);
  (* ram_style = "block" *) reg [31:0] mem [0:6166];
  integer i;
  initial begin
    for (i = 0; i < 6167; i = i + 1) mem[i] = i;
  end
  reg [31:0] rd;
  always @(posedge clk)
    rd <= mem[addr];
  assign q = rd;
endmodule
```

Listing 1: The minimal reproducer: a read-only 6167 × 32 block RAM.

yosys maps it to five `RAMB18E1` cells in SDP mode with $READ\_WIDTH\_A = 36$ — the same defect class as the LiteX ROM (a 36-bit read assembled from the A *and* B sides), at minimal scale. Running the openXC7 place-and-route on it made the defect visible in five lines of FASM:

```
--- unfixed nextpnr (0.9.2), one cell of five:
BRAM_L_X6Y85.RAMB18_Y1.READ_WIDTH_A_18
BRAM_L_X6Y85.RAMB18_Y1.SDP_READ_WIDTH_36
(no READ_WIDTH_B feature: defaults to width 1)

--- fixed nextpnr (commit f1c77134), delta: +5 lines, 0 removed:
BRAM_L_X6Y85.RAMB18_Y1.READ_WIDTH_B_18
BRAM_L_X6Y95.RAMB18_Y0.READ_WIDTH_B_18
BRAM_L_X6Y95.RAMB18_Y1.READ_WIDTH_B_18
BRAM_L_X6Y100.RAMB18_Y0.READ_WIDTH_B_18
BRAM_L_X6Y115.RAMB18_Y0.READ_WIDTH_B_18
```

Listing 2: The minimal reproducer's FASM delta: unfixed nextpnr emits no `READ_WIDTH_B` feature at all (the B-side read path stays at width 1); the patched toolchain adds exactly one `READ_WIDTH_B_18` per SDP cell.

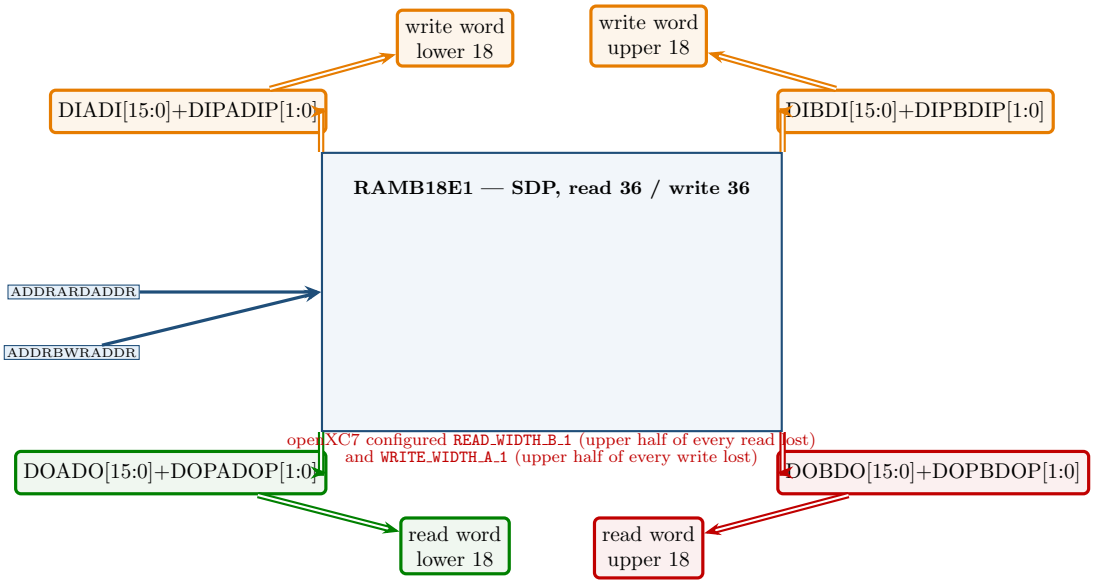


Figure 9: The SDP-36 register file. Every 36-bit read needs `READ_WIDTH_B_18` and every 36-bit write needs `WRITE_WIDTH_A_18`; the openXC7 bitstream left both at width 1, so the CPU's register reads and writes were half-garbage from the first instruction onward.

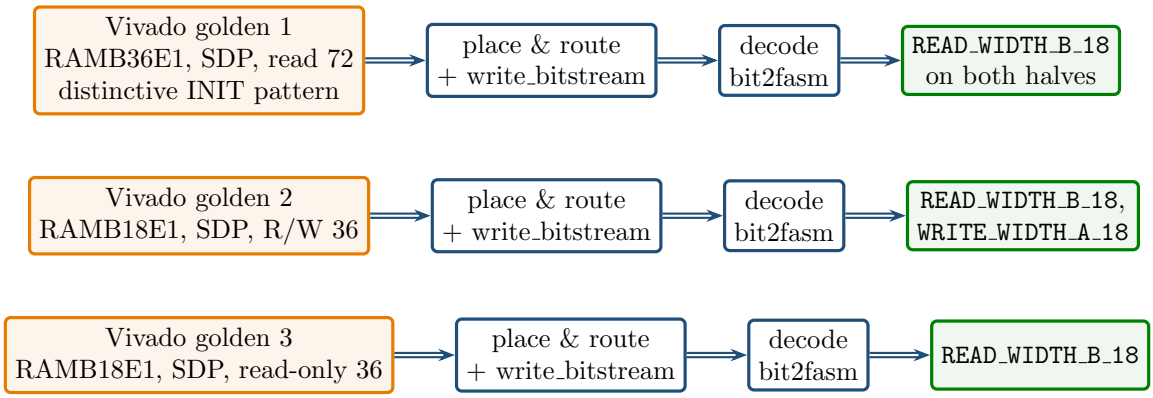


Figure 10: Three purpose-built Vivado golden designs pinned down the hardware truth: Vivado always programs the opposite-side width markers to `_18` in SDP mode — the openXC7 bitstream did not.

The two directions bracket the bug: the Vivado goldens prove what the hardware *should* be programmed with, and the minimal reproducer proves what the openXC7 flow *does* program — and that the patched flow now programs exactly what the goldens prescribe.

4.6 Why simulation masked the bug

The final puzzle: why did every simulation boot? Because the behavioral models *derive* the missing widths. Yosys's SDP-72 cells leave $READ\_WIDTH\_B = 0$ as a convention; the unisim model interprets this as "B side takes its width from the A side" and produces correct read data. On real silicon, however, the B-side data path is a physical mux whose width comes from the `READ_WIDTH_B` configuration bits — and with those bits at width-1, the upper data lines are simply not driven.

The bug in one sentence

Simulation uses *parameters*; silicon uses *configuration bits*. nextpnr translated the parameter correctly for the primary side of every SDP port, but translated the opposite side — which carries half the data — as "unused, width 1".

5 The fix and its verification

5.1 The change

The defect lives in `write_bram_width()`, the nextpnr-xilinx function that turns a BRAM cell's width parameters into FASM features (`xilinx/fasm.cc`). It treated the opposite-side parameters of SDP ports as unused and emitted the width-1 default. The fix emits the width-18 markers the hardware actually needs, exactly as Vivado's golden bitstreams do. The port-width parameters are fetched once, and every condition in the `if` statements is a named boolean constant whose name states its meaning, so the intent is readable without decoding the expressions.

```
1  void write_bram_width(CellInfo *ci, const std::string &name, bool is_36, bool is_y1)
2  {
3      // SDP mode spans both data sides of the memory, so the "opposite-side"
4      // width markers must also be configured wide, exactly like Vivado's
5      // golden bitstreams:
6      //   - RAMB36E1 READ_WIDTH_A=72: the 72-bit read is the A port (lower
7      //     36 bits) PLUS the B port (upper 36 bits) -> READ_WIDTH_B_18 on
8      //     BOTH RAMB18 halves.  yosys leaves READ_WIDTH_B at 0 and its
9      //     unisim model derives the B width from READ_WIDTH_A, so the
10     //     width-1 default looked correct in simulation; on silicon it
11     //     kills the B-side read path and the upper half of every read
12     //     returns garbage (litex-ddr-arty-s7 ROM: CPU fetches corrupted
13     //     instructions -> dead serial console while the equivalent Vivado
14     //     build works).
15     //   - RAMB18E1 READ_WIDTH_A=36: the 36-bit read is DOADO/DOPADOP
16     //     (lower 18) PLUS DOBDO/DOPBDOP (upper 18) -> READ_WIDTH_B_18.
17     //   - RAMB18E1 WRITE_WIDTH_B=36: the 36-bit write takes DIADI/DIPADIP
18     //     (lower 18) PLUS DIBDI/DIPBDIP (upper 18) -> WRITE_WIDTH_A_18
19     //     (VexRiscv regfile: register writes lose half their data
20     //     otherwise).
21     //
22     // Fetch the port-width parameters once so that every condition below
23     // is a self-documenting named boolean.
24     const int read_width_a  = int_or_default(ci->params, ctx->id("READ_WIDTH_A"), 0);
25     const int write_width_b = int_or_default(ci->params, ctx->id("WRITE_WIDTH_B"), 0);
26     const int this_width    = int_or_default(ci->params, ctx->id(name), 0);
27
28     // The width parameter being emitted is unset: yosys leaves the
29     // opposite side of an SDP port at 0, although that side still carries
30     // data on silicon.
31     const bool this_width_param_is_unset = (this_width == 0);
32     // The parameter being emitted is the B-side read width.
33     const bool this_param_is_read_width_b = (name == "READ_WIDTH_B");
34     // The parameter being emitted is the A-side write width.
35     const bool this_param_is_write_width_a = (name == "WRITE_WIDTH_A");
36     // On a RAMB36E1 the B side carries half of a 72-bit SDP read; on a
37     // RAMB18E1 it carries half of a 36-bit SDP read.
38     const bool b_side_reads_half_the_word =
39             (is_36 ? (read_width_a == 72) : (read_width_a == 36));
40     // On a RAMB18E1 the A side carries half of a 36-bit SDP write.
41     const bool a_side_writes_half_the_word = (!is_36 && (write_width_b == 36));
42
43     if (this_width_param_is_unset && this_param_is_read_width_b && b_side_reads_half_the_word)
        {
```

```
44          write_bit("READ_WIDTH_B_18");
45          return;
46      }
47      if (this_width_param_is_unset && this_param_is_write_width_a &&
        a_side_writes_half_the_word) {
48          write_bit("WRITE_WIDTH_A_18");
49          return;
50      }
51      int width = this_width;
52      if (width == 0)
53          width = 1; // golden encodes unused ports as width 1
54      /* ... existing emission logic, unchanged ... */
55  }
```

Listing 3: The fix in `write_bram_width()` (nextpnr-xilinx `xilinx/fasm.cc`, commit `f1c77134`, 49 insertions / 1 deletion).

5.2 What changes in the design

Rebuilding the demo with the patched toolchain changes the FASM by exactly sixteen added lines and nothing else:

Feature added	where	why
`READ_WIDTH_B_18`	ROM tiles × 6, both halves (12 lines)	upper 36 bits of every 72-bit fetch
`READ_WIDTH_B_18`	register file, both halves (2 lines)	upper 18 bits of every 36-bit read
`WRITE_WIDTH_A_18`	register file, both halves (2 lines)	upper 18 bits of every 36-bit write

Table 3: The complete delta: 16 FASM lines.

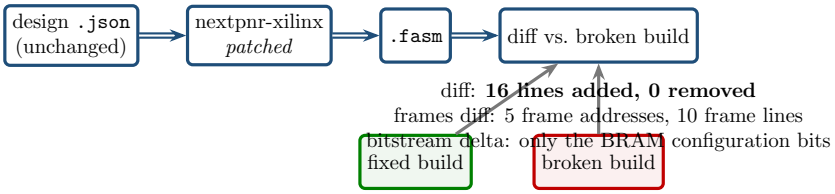


Figure 11: Verification at three levels: the FASM diff is exactly the 16 expected lines, the frames diff is confined to the BRAM configuration frames, and the decoded configuration of every affected tile now matches the Vivado goldens feature-for-feature.

5.3 Hardware confirmation

The patched toolchain produced a new bitstream (`digilent_arty_s7.fixed2.bit`); flashed onto the Arty S7 it brought the board to life:

Hardware result

"The test bitstream works!" — the LiteX BIOS banner appears on the serial console and the SoC runs, confirming the diagnosis end-to-end.

6 Lessons learned

6.1 The technical lesson

1. **Simulation is not silicon.** Every simulation — RTL and gate-level, with yosys's own models and with Xilinx's unisim models — reported a healthy design. The defect was purely in configuration bits that the behavioral models ignore or derive. For Series-7 work, "it simulates" is necessary but never sufficient; bitstream-level configuration must be checked independently.

2. **"Unused" ports may still carry data.** In SDP mode the opposite side of a port is not unused — it is half of the data path. Parameter conventions that mark it as 0 are model conventions, not hardware truth.

3. **Golden bitstreams are the ground truth.** The most decisive evidence in the whole investigation came from decoding tiny purpose-built Vivado designs. When tool behavior is in doubt, ask the silicon what Vivado programs.

4. **Verify transforms on known-good data.** The RAMB36-to-RAMB18 INIT interleave was validated against the working Vivado pair before being trusted on the broken openXC7 data — a habit that prevented a wrong conclusion in either direction.

5. **Cross-check simulators.** Icarus Verilog produced a spurious "hang" (a missed *always @(*)* sensitivity); Verilator disagreed. One simulator is an opinion; two are evidence.

6.2 The process lesson

1. **Keep sources identical between A/B builds.** The two builds compiled different VexRiscv netlists, which had to be investigated and cleared before the comparison was valid. A two-line Makefile alignment would have removed that detour.

2. **Write the hypothesis list down.** Five hypotheses, each with an explicit test and an explicit verdict — the chain in fig. 4 kept the investigation moving instead of thrashing.

3. **Record the eliminating evidence.** Each eliminated hypothesis produced a reusable artifact (the validated INIT transform, the netlist simulation harness, the golden designs); the final proof was assembled almost entirely from them.

4. **Minimal reproducers beat big ones.** The one-cell Vivado goldens settled the question in minutes where whole-design comparisons had failed for hours.

6.3 Artifacts

Artifact	Location
nextpnr-xilinx fix	branch `fix-sdp72-read-width`, commit `f1c77134`
working bitstream	`tmp/wt-dp/litex-ddr-arty-s7.mine/digilent_arty_s7.fixed2.bit`
Vivado golden designs	`tmp/vivsdp`, `tmp/vivsdp36`, `tmp/vivsdp36ro`
minimal reproducer	`tmp/bramtest/top.v` (6167×32 ROM $\rightarrow$ 5×RAMB18E1 SDP-36)
simulation harnesses	`tmp/sim-s7` (RTL, netlist, unisim)
decoded bitstreams	`tmp/vivado.fasm`, `tmp/openxc7.fasm`, `tmp/fixed2.fasm`

One-paragraph summary

A LiteX/VexRiscv SoC built with openXC7 sat silent while the same netlist built with Vivado booted. Bitstream decoding showed the PLL, the clock tree and the ROM contents to be perfect, and simulation showed the synthesized logic to be perfect. The defect was

found by building one-cell Vivado "golden" designs for the two SDP BRAM structures (the 72-bit-read BIOS ROM and the 36-bit register file): nextpnr-xilinx configured the *opposite-side* port-width markers of those SDP ports as 1-bit instead of 18-bit, so hardware drove only half of each memory word. Sixteen FASM lines — fourteen `READ_WIDTH_B_18` and two `WRITE_WIDTH_A_18` — restored the CPU's data paths, and the board came alive.