

FireSim over Ethernet: A Rocket Core on a Low-Cost FPGA Board

Filip Filmar and Dragiša Janković

Abstract—FireSim runs a RISC-V system generated by Chippyard on an FPGA, much faster than a software simulator, and its host program reaches the FPGA over PCIe. We run it on the Alinx AX7A200B, an Artix-7 200T board that sits beside a host on an Ethernet cable rather than in a PCIe slot. The host’s program and the board exchange raw Ethernet frames: one kind of frame for the simulator’s control registers and another for its CPU-managed streams, both answered once each and repeated safely when a frame is lost. On the board, units written in TxHDL take the frames to and from the ports of the simulator that Golden Gate generates, and the target’s memory is the board’s DDR3. The design itself comes prebuilt: a continuous integration job builds FireSim’s Rocket design and publishes it with each release of a set of Bazel rules, and the board build fetches it by checksum and needs no Scala. On October 27 and 28, 2026, FireSim’s single-core Rocket design loaded programs into DDR3, printed over its UART, took UART input, and ran a compute-bound loop at 45.7 MHz effective on a 50 MHz FPGA clock. The stream path, which FireSim’s instruction tracer uses, does not yet work on the board.

I. WHY ETHERNET

FireSim [1] turns a register-transfer design into a simulator that runs on an FPGA. Golden Gate [2], its compiler, decouples the target’s clock from the FPGA’s, so the target can stop while the host serves one of its bridges and resume without losing a cycle. The published platforms are datacenter and cloud FPGAs and the Nitefury II, an M.2 card. Each sits in a host’s PCIe slot, and the host reaches it through the Xilinx XDMA core.

The AX7A200B carries the same part as the Nitefury II, an `xc7a200tfg484-2`, and 1 GiB of DDR3. Its PCIe edge connector needs a slot it is not in: the board is on a desk, attached to its host by JTAG, a serial line and a gigabit Ethernet cable. FireSim’s host program talks to the FPGA through one class, `simif_t`, so a platform needs only its own implementation of that class. Ours sends Ethernet frames.

The cost is time per access. A register access over PCIe takes about a microsecond; over a frame and its answer it takes tens. FireSim polls registers while a bridge needs the host, so those phases run slower. Section VI measures what that costs when the target does not need the host: almost nothing.

Both authors are independent. Filip Filmar is the corresponding author, at f@hdlfactory.com; Dragiša Janković is reached at linkedin.com/in/dragisa-jankovic.

This document reports FireSim run on the Alinx AX7A200B board with Ethernet in place of PCIe. Every listing is included from the project repository `HDL/txhdl` at build time, and every number is from a build or a board run, dated where it is given. The Ethernet part is in [4], and the cover [5] lists every document.

The authors used a large language model, Claude, as an assistant in exploring the concepts and in writing and constructing the documents and the programs; every commit of the repository says so and preserves the prompts that led to it, verbatim.

Built by Bazel from commit `ae93fb1eaa` on 2026-10-29.

II. THE TOOLCHAIN

FireSim builds with `sbt` and `make`, inside a `conda` environment. Here it builds with Bazel, from 22 modules in a Bazel registry of the authors’. The target’s generators are Chippyard’s [3]. Most are thin overlays: an upstream archive at the commit Chippyard 1.15.0 pins, plus a build file. FireSim is at 1.22.0.

The target side compiles with Chisel 6.7.0 on Scala 2.13.16, the versions Chippyard uses. Golden Gate is a second Scala world, Chisel 3.6.1 on Scala 2.13.14, and a Bazel workspace has one Scala version, set by its root module. So Golden Gate is built in a repository of its own and published as one self-contained jar; the rules run it as a tool.

The rules repository, `bazel_rules_firesim`, provides four rules: elaboration, Golden Gate, the host driver, and the bitstream through Vivado. Its release workflow also builds FireSim’s Rocket design for the Nitefury II, `FireSimRocket1GiBDRAMConfig` on `BaseNitefuryConfig`, and attaches Golden Gate’s outputs to the release as a 0.9 MB archive. On October 27, 2026, the GitHub build of release `v0.2.1` took 14 minutes, and its five files were identical, byte for byte, to a cold build on the authors’ machine, which took 62 minutes. Two steps make that possible. The rules rewrite Chisel’s source locators from absolute paths in Bazel’s cache to paths inside each repository. They also package the archive with sorted names and fixed times.

TxHDL fetches the archive by its SHA-256 and builds the board from it, with no Scala in its own build. The AX7A200B uses the Nitefury configuration as it is: the simulator’s ports are the same, and both boards have 1 GiB of DDR3.

III. THE FRAMES

Golden Gate’s top module, `FPGATop`, has three ports a host reaches. The control port is AXI4 with 32-bit data, used one 32-bit word at a time. The CPU-managed port is AXI4 with 128-bit data and 64-bit addresses, through which the host drains and fills the bridges’ streams. The memory port is AXI4 with 64-bit data, 16-bit identifiers and 30-bit addresses: the target’s DRAM, which the board provides.

The host reaches the first two with raw Ethernet frames, one request per frame and one answer per request (Table I). The control frames are those of TxHDL’s remote peripheral [4], under `EtherType 0x88b5`. The stream frames carry one 16-byte beat each under `0x88b6`, the other `EtherType` IEEE leaves for local experiment.

A frame can be lost, so the host sends a request again when its answer does not come, under the same tag. Neither port tolerates a request performed twice: writing some control registers starts an action, and a stream read removes data from a queue. So each master keeps the tag and the answer of the last request it performed, and answers a repeat of that tag from them without touching its port. A host run cannot know the tag the board kept from an earlier run, so it starts with a

TABLE I

THE TWO KINDS OF FRAME: THE BYTE OFFSETS OF EACH FIELD, AFTER THE 14-BYTE ETHERNET HEADER. REQUEST AND ANSWER SHARE A LAYOUT; THE MAC PADS EACH FRAME TO 60 BYTES.

Field	Control (0x88b5)	Stream (0x88b6)
kind	14: 0x02, 0x82	14: 0x03, 0x83
device	15	15
tag	16	16
flags	17: write; error	17: write, no-op; error
address	18 to 21	18 to 25
data	22 to 25, a word	26 to 41, a beat
strobe	26	42 to 43
bytes used	27	44

request that repeats harmlessly: a read of a status register on the control path, and a no-op on the stream path, which the stream master answers without its port and whose tag it keeps.

IV. ON THE BOARD

Figure 1 shows the board’s design. The MAC’s two halves and the PHY’s RGMII interface are TxHDL’s Ethernet design [4]. Bytes cross between the PHY’s 125 MHz and the simulator’s 50 MHz in gray-coded channel FIFOs.

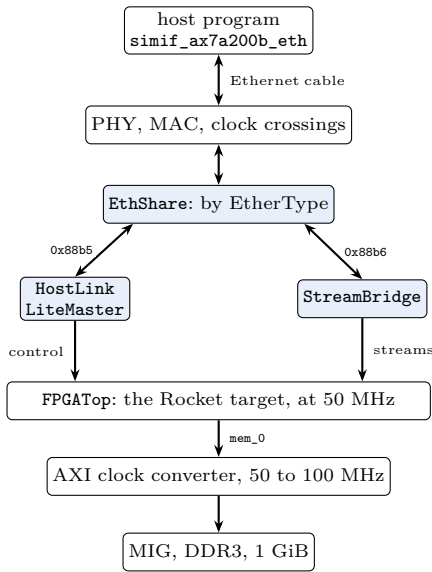


Fig. 1. FireSim’s Rocket design on the AX7A200B. The shaded units are TxHDL’s, lowered as one unit, `rocket_link`.

The three shaded units are lowered together as one unit, `rocket_link`, so the channels between them are buffered inside it. Joined by wires in the hand-written top instead, each channel’s valid would depend on its own ready, a combinational loop. `EthShare` reads each frame’s `EtherType` and hands control frames to `HostLink`, whose `LiteMaster` performs them on the control port, and every other frame to `StreamBridge`, which drops those that are not stream frames. Listing 1 is `StreamBridge`: `StreamLink` reads and writes the frames, and `BeatMaster` performs each request as an AXI4 burst of one beat. Every unit has a test against a software model, and the build simulates each lowered netlist against the same run under `nvc` and `Verilator`.

```

/// ['StreamLink'] and ['BeatMaster'] as one unit: frames in and out on
/// one side, the AXI4 port on the other. Joined inside the unit for
/// the reason ['HostBridge'] (super::host::HostBridge) is.

```

```

#[derive(Trace, Default)]
pub struct StreamBridge<const DEV: usize> {
    /// The frames.
    pub link: StreamLink<DEV>,
    /// The port.
    pub master: BeatMaster,
}

#[lower]
impl<const DEV: usize> Unit for StreamBridge<DEV> {
    async fn run(
        &mut self,
        (rx, b, r): (Rx<EthByte>, Rx<B<4>>, Rx<R<128, 4>>),
        (tx, aw, ar, w): (
            Tx<EthByte>,
            Tx<Aw<64, 4>>,
            Tx<Ar<64, 4>>,
            Tx<W<128, 16>>,
        ),
    ) {
        let (ask_tx, ask_rx) = chan::<BeatAsk, DefaultClock>();
        let (ans_tx, ans_rx) = chan::<BeatAnswer, DefaultClock>();
        join2(
            self.link.run(rx, ans_rx), (ask_tx, tx)),
            self.master.run(b, r, ask_rx), (aw, ar, w, ans_tx)),
        .await;
    }
}

```

Listing 1. `StreamBridge`: the frames on one side, the 128-bit AXI4 port on the other.

The target’s DRAM is the board’s DDR3, through AMD’s MIG 7-series controller as TxHDL’s own designs use it, with one change: its AXI port is 64 bits wide with 16-bit identifiers, the shape of `FPGATop`’s memory port, and the MIG widens to its 256-bit native port inside. An AXI clock converter crosses from the simulator’s 50 MHz to the MIG’s 100 MHz user clock. The simulator, the link and its crossings are held in reset until the MIG has calibrated.

Vivado 2025.2 places and routes the design on the `xc7a200tfbg484-2` in 31 minutes. It meets timing, with 0.096 ns of setup slack and 0.036 ns of hold slack, and uses 79 971 LUTs (60%), 38 687 flip-flops (14%), 202 block-RAM tiles (55%) and 16 DSPs. The Nitefury II’s own PCIe build of the same design uses 92 354 LUTs.

V. THE HOST

The host program is `FireSim`’s, from its sources: its main loop, the bridges’ drivers of Chipyard’s `FireChip`, and the design’s generated header from the prebuilt archive. Only the platform file is TxHDL’s. It implements `simif_t`’s register read and write and `FireSim`’s `CPUMangedStreamIO` on two raw sockets, one per `EtherType`. It sends each request to every port of the interface cabled to the board, waits up to 20 ms for the answer, and sends it again up to 50 times.

A raw socket needs the `cap_net_raw` capability. A binary given a file capability runs in Linux’s secure mode, in which the loader ignores `LD_LIBRARY_PATH`, so the driver links `Spike`, which FireSim’s bridges use, statically. It is one file of 3.7 MB that needs only the C library’s shared objects.

VI. RESULTS

Table II lists the board runs. FireSim’s GCD example, whose bridges are registers only, passed first, on October 27, 2026. The Rocket design followed the same day. Its driver loaded each program into the DDR3 through FireSim’s TSI bridge, Rocket’s boot ROM jumped to it, and its exit through `tohost` ended the run.

`bench.riscv` times a loop of 50 million iterations with the core’s own counters: 350 000 004 instructions in 600 000 056 cycles. The whole run, 687 634 397 target cycles, took about 15.03 s of the host’s time on a 50.007 MHz FPGA clock: FireSim reports an effective target clock of 45.7 MHz, a ratio of 1.09

TABLE II
BOARD RUNS, OCTOBER 27 AND 28, 2026.

Program	What it checks	Result
GCD	registers	[GCD]PASS
hello.riscv	DDR3, UART output	passed
echo.riscv	UART input	passed
bench.riscv	speed	45.7 MHz
hello, TracerV	the stream path	failed

FPGA cycles per target cycle, and the driver rounds the time to 15.0 s. That is about 23 million target instructions per second. While the target runs on its own, the link costs nearly nothing: the host steps in only while a bridge needs it, and this loop needs none.

`echo.riscv` echoed a line typed into the driver’s standard input, through FireSim’s UART bridge, and stopped at a `q`.

The stream path did not work on the board. With FireSim’s TracerV bridge set to write a trace, `hello.riscv` still passed, but the trace file held only its header. The next run got no answer to its first step of the simulation, and the board needed reprogramming. Each read that FireSim’s stream engine receives takes one 16-byte beat out of a 64-byte queue entry, which is how `StreamBridge` reads, so the frames’ shape is not the cause; the cause is still open (issue 2007).

VII. WHAT IS NOT DONE

- The stream path on the board (issue 2007). Until it works, a design whose bridges use CPU-managed streams, as TracerV does, cannot run.
- Linux on the Rocket design: a kernel and a root file system, and FireSim’s block-device bridge, whose every access is a register exchange over the cable.
- The bitstream as a prebuilt. Only Golden Gate’s outputs are published; each board build still runs Vivado.
- PCIe. The board stays on its cable for now.

REFERENCES

- [1] S. Karandikar *et al.*, “FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud,” in *Proc. ISCA*, 2018.
- [2] A. Magyar *et al.*, “Golden Gate: Bridging the resource-efficiency gap between ASICs and FPGA prototypes,” in *Proc. ICCAD*, 2019.
- [3] A. Amid *et al.*, “Chipyard: Integrated design, simulation, and implementation framework for custom SoCs,” *IEEE Micro*, vol. 40, no. 4, 2020.

- [4] F. Filmar and D. Janković, “Ethernet in TxHDL,” project repository `HDL/txhdl`, `//docs:eth`, 2026.
- [5] F. Filmar and D. Janković, “TxHDL documents,” project repository `HDL/txhdl`, `//docs:cover`, 2026.