IronForge Specification

AI-Assisted Embedded Linux Development on Native Windows

Arty A7-100T / MicroBlaze / SPI JFFS2 and Windows NFS

Revision 11-16-2026

1. Overview and Qualification Status

IronForge is an AI-based development environment for building embedded Linux on an FPGA from native Windows. An agent begins with tracked `start.md`, examines the workspace and requested change, and uses the repository's explicit build and board operations. `install.cmd`, `ironforge.cmd`, tracked configurations, patches and checked build receipts perform the actual preparation, compilation, packaging and deployment work. The agent plans and explains the work; the tools execute defined steps and reject stale or mismatched inputs.

Windows PowerShell starts the engine. AMD Vivado/Vitis provide FPGA generation, JTAG, MicroBlaze cross-compilers and device-tree tools. MSYS2 supplies host build utilities required by upstream projects. Linux executes on the FPGA's MicroBlaze processor. No WSL, Linux virtual machine or container is required. VS Code is an optional interface to the same commands.

The implemented target is the Digilent Arty A7-100T, `xc7a100tcsg324-1`, with a 100 MHz little-endian MMU MicroBlaze, 256 MiB DDR3, EthernetLite, UARTLite and 16 MiB SPI NOR. Arm-based AMD SoCs are a planned extension; this repository does not implement or qualify an Arm target.

The two root modes are board-local writable JFFS2 and Windows-hosted NFS. Both originate from the one editable `rootfs/` tree. The current flash design has five partitions and stores `/boot/uImage` and `/boot/system.dtb` inside JFFS2. The current image passed complete programming/verification and FPGA reconfiguration boot with Windows NFS stopped. Current serial/runtime evidence is distinguished from the earlier seven-partition performance baseline.

Item	Status at 11-16-2026	Scope
Native Windows build workflow	Implemented and exercised on the development PC	Complete fresh-PC installation/build untested
FPGA and DDR diagnostic	Baseline hardware evidence recorded	Repeat after hardware/tool changes
Current NFS Linux boot	SPI-loaded U-Boot fetch, writable root and services passed	Existing Windows host; same current kernel/DTB
SPI JFFS2 root and local services	Current five-partition boot and HTTP/MQTT/SSH passed	FPGA reconfiguration; physical cold cycle untested
Current JFFS2 `fsload` boot payload	Kernel and DTB loads passed	Current-image serial/runtime evidence retained
Physical cold power-cycle startup	Untested	FPGA reconfiguration is a separate test

Item	Status at 11-16-2026	Scope
Linux `reboot`	Halts	No implemented restart controller
Standalone initramfs acceptance	Pending	Generated archive is not a qualified alternate startup path
Generic public flash release	Not provided	Local image contains private SSH provisioning

A build, fixture test or image inspection establishes its stated checks. It does not establish power-cycle reliability, storage endurance, production security or a compatible build on every Windows/AMD release. AI assistance does not change these evidence requirements or guarantee identical binaries.

Quick start

After installing AMD tools with classic MicroBlaze support, read `start.md` and run from PowerShell in the repository root:

```
.\install.cmd
.\ironforge.cmd doctor
.\install.cmd -Plan -Build
```

The final command previews a build. Section 5 gives the full onboarding sequence. Board commands in sections 13 and 14 change the selected board and require an explicit cable identity.

2. Table of Contents

- 1. Overview and Qualification Status
- 3. Agent Entry Flow and System Requirements
- 4. Architecture and Boot Paths
- 5. Fresh Windows PC Installation
- 6. Downloads, Versions and Compiled Components
- 7. Patch Application and Recovery
- 8. Workspace and Command Line API
- 9. Build Order and Configuration
- 10. FPGA Modules, Clocks and Reset
- 11. Memory Map and Flash Layout
- 12. Root Filesystem and Windows Metadata
- 13. NFS Bring-up and Operation
- 14. SPI Image, Programming and Recovery
- 15. Linux Applications and Runtime API
- 16. Build Evidence and Packaging Gates
- 17. Source Modules and Host Scripts
- 18. Rebuild and Update Workflows
- 19. Verification and Recorded Measurements
- 20. Debugging, Limitations and Licensing

Reading and maintaining this specification

Paths are relative to the repository unless stated otherwise. Host examples run in native Windows PowerShell. Commands at `IronForge>` run in U-Boot; commands at a Linux shell run on the board. Replace every angle-bracket placeholder. The PC LAN address, Arty address, cable serial and COM port are distinct selections.

`platforms/arty-a7-100t/platform.json` defines target addresses and flash partitions. `platforms/sources.lock.json` defines upstream archive identities, hashes and extracted directory names. `platforms/host-tools.lock.json` defines downloaded host bootstrap archives. Code implements the checks/actions; local receipts and logs identify what ran on a particular workstation.

Qualification rows distinguish baseline evidence from outstanding tests. Changes to boot images, partition layout, hardware or runtime behavior require matching evidence before the changed implementation is described as qualified. Run dates belong in test evidence, not invented from the document revision.

Architecture drawings are editable under `documentation/diagrams/src/`; rendered PNGs appear in section 4. Regenerate and review them after architecture changes. No Google Slides reference URL is recorded in this repository. Documentation rendering is separate from target image building.

3. Agent Entry Flow and System Requirements

AI and tool responsibilities

`start.md` is the tracked entry document for an agent or developer opening the project. Read it before choosing commands, then inspect repository instructions, current platform/locks, local environment and the requested change. Establish which board and boot mode are intended, which artifacts already match their inputs, and which evidence is missing. Use previews and read-only inspection where appropriate before a hardware operation.

The agent can explain failures, edit tracked source/configuration, choose an appropriate rebuild scope and report results. The build engine performs archive verification, patching, configuration, compilation and image checks. Board scripts select the intended cable and execute explicit operations. An agent must not replace those checks with a claim that a file looks current, silently bypass verification, or infer a hardware pass from successful compilation.

Authorization comes from the user's task and applicable workspace instructions. A source/document edit does not imply permission to erase another board. Preserve the selected-cable boundary and other connected devices. Do not copy private settings/keys into public documentation or treat a locally provisioned flash image as a release artifact. Report exact tests run, skips, failures and untested behavior.

Host requirements

Requirement	Purpose and restriction
Native Windows and Windows PowerShell	Installer, CLI launchers, serial and firewall helper
Short local ASCII checkout	No spaces, symlinks or junctions; installer path limit 120 characters
AMD Vivado/Vitis	Separate installation with target device support and classic MicroBlaze Linux/bare-metal compilers
Required AMD licenses	Supplied/accepted by the user, not IronForge
MSYS2	Bash, make, host compiler and build utilities; private bootstrap available
Internet for first preparation	Locked archives and rolling MSYS2 packages
Windows OpenSSH Client	`ssh.exe` and `ssh-keygen.exe` for demo credentials/client access

Requirement	Purpose and restriction
Optional VS Code	Explorer actions and native terminal menuconfig
Optional documentation tools	Node.js, browser and drawing exporters, separate from target build

Discovery selects available tools; it does not establish compatibility with every AMD release. The recorded environment uses AMD 2026.1. CPU/compiler ABI changes require matching hardware/software rebuild and qualification. MSYS2 runs Windows host programs; it is not a Linux target runtime or VM.

Board and network requirements

Use an Arty A7-100T with USB JTAG/UART, adequate power and Ethernet. NFS needs a reachable PC IPv4 address and normal scripted boot expects DHCP. SPI startup requires JP1 fitted for Master SPI; JTAG can override startup configuration. Probe the actual flash before choosing the Vivado configuration-memory part because board revisions may differ.

Serial is 115200 baud, 8N1, no flow control, DTR/RTS disabled. Keep one process owning the COM port. Close the terminal before the serial flash probe. The demo is for a trusted LAN: HTTP and MQTT do not supply authentication/TLS. There is no verified boot, encrypted root storage, automatic restart or automatic update rollback.

4. Architecture and Boot Paths

Windows holds the agent/tool interface, source/build trees, compilers, Vivado, native NFS server and optional clients. The FPGA holds the processor, memory subsystem and peripherals. Ethernet carries NFS traffic when selected, HTTP/MQTT/SSH and iperf2. USB supplies JTAG and UART independently of Ethernet.

Figure 4-1. Windows development and board runtime architecture (`ironforge-demo` , sheet 1). Review drawing content against the five-partition implementation when regenerating it.

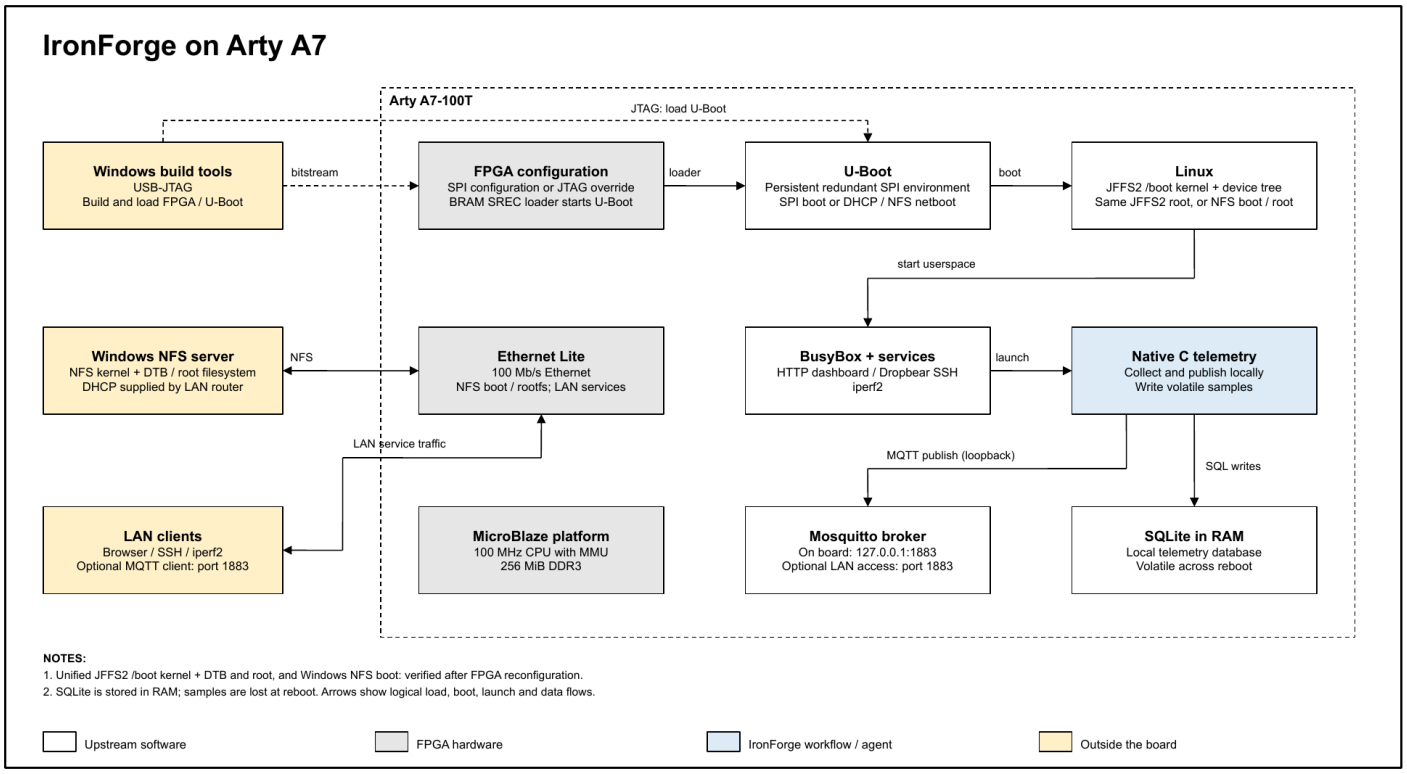


Figure 4-2. SPI and NFS boot paths (`ironforge-demo`, sheet 2).

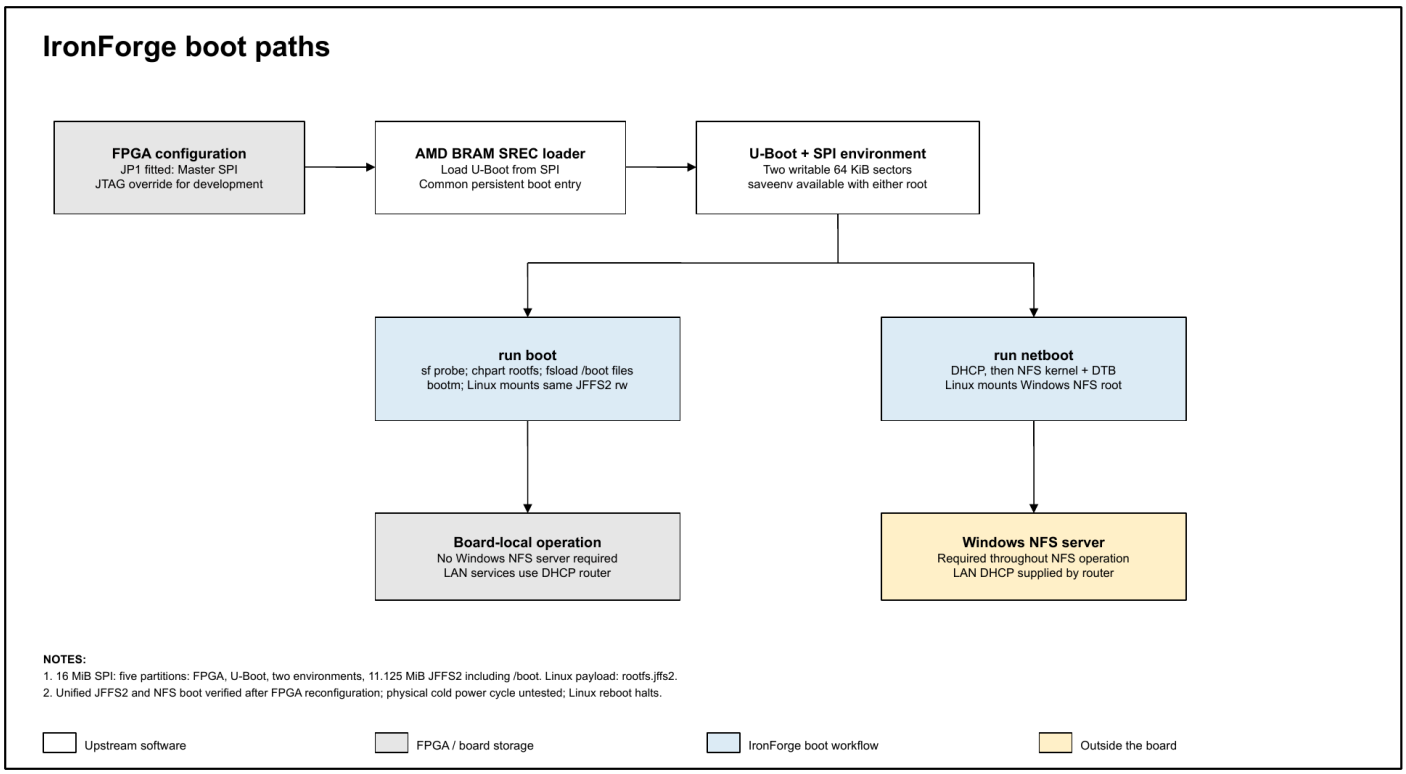
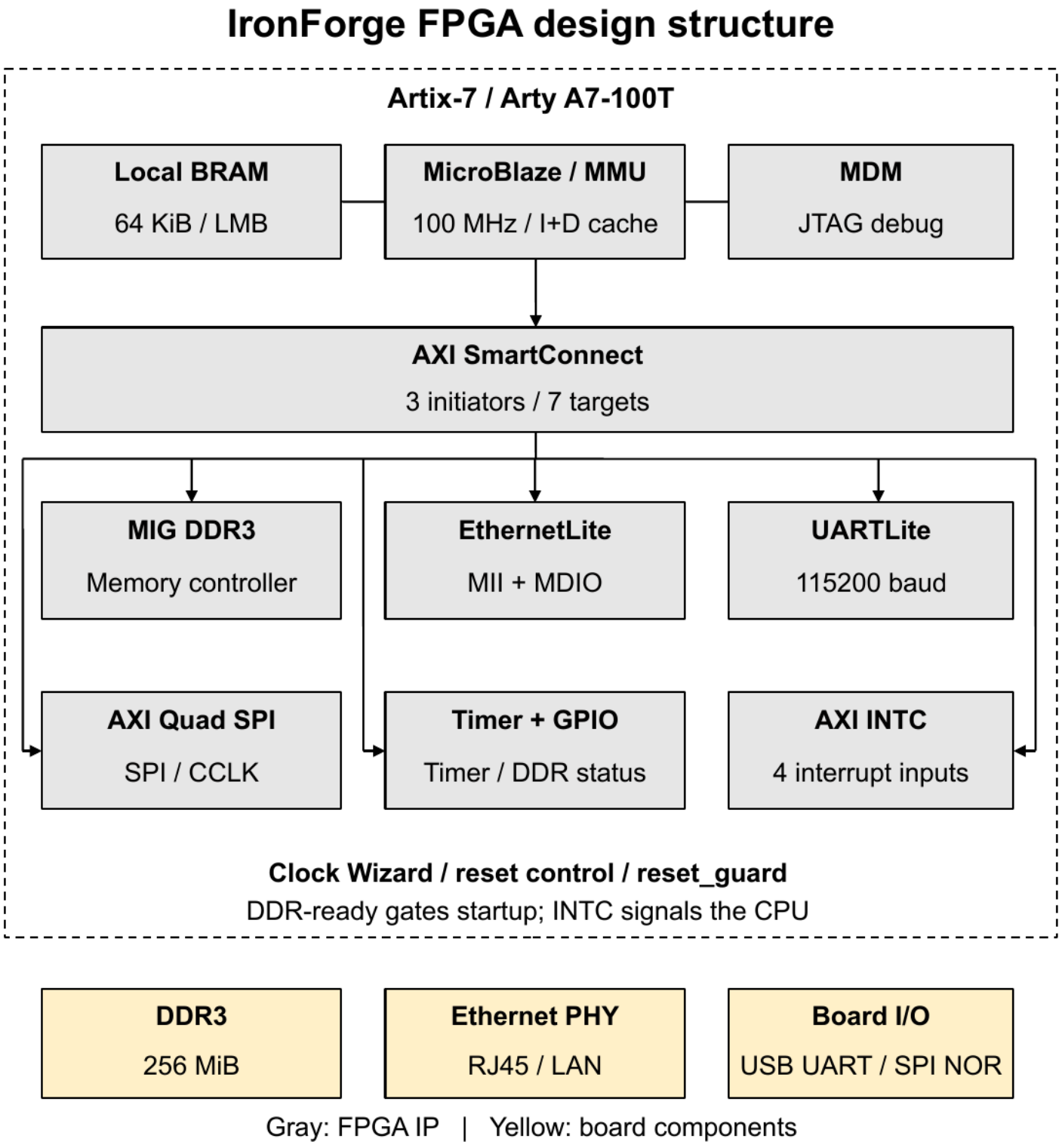


Figure 4-3. Implemented FPGA design structure (`ironforge-fpga` , sheet 1), drawn from `hardware/arty-a7-100t/build.tcl` . The MicroBlaze connects to local BRAM and AXI peripherals; the board's DDR3, Ethernet PHY, USB UART and SPI NOR sit outside the FPGA boundary.



Local SPI startup

FPGA configuration reads the beginning of SPI NOR. The flash bitstream contains the AMD SREC loader in BRAM. After reset and DDR calibration, the loader reads U-Boot SREC from its bounded partition, checks records/destinations, writes DDR and transfers control. U-Boot has a three-second countdown and defaults to `run boot`.

The current environment's `boot` command probes SPI, selects the rootfs MTD partition and loads `/boot/uImage` and `/boot/system.dtb` through JFFS2 `fsload`, then calls `bootm` with a writable JFFS2 root. Kernel and DTB have no separate flash partitions. Linux mounts the same rootfs. Windows NFS is unnecessary for local operation. The current image passed this boot path after FPGA reconfiguration with Windows NFS stopped.

NFS startup

JTAG can load U-Boot after FPGA programming and diagnostic validation. Interrupt autoboot and enter `run netboot`. U-Boot obtains a client address using DHCP, preserves the chosen server, fetches `/ironforge/boot/uImage` and `/ironforge/boot/system.dtb` over NFS, and starts Linux with NFSv3/TCP root exported by Windows as `/ironforge`. This path remains independent of JFFS2 file loading.

Both commands replace full Linux boot arguments and clear `fdt_high` so U-Boot relocates the DTB into aligned available RAM. `saveenv` persists to redundant SPI sectors in either mode. Previously saved environment can override compiled defaults; inspect it when validating changed scripts.

Artifact separation

`artifacts/system.bit` is the normal JTAG bitstream. Flash packaging uses `updatemem` to create a separate BRAM-loader bitstream. `rootfs/` is the sole editable filesystem and NFS export. `build/filesystem/` contains generated filesystem images/staging, not another editable rootfs. JTAG debug access is isolated by cable selection; Ethernet services do not provide JTAG access.

5. Fresh Windows PC Installation

Install AMD Vivado/Vitis separately, including device support, classic MicroBlaze Linux and bare-metal toolchains and applicable licenses. Install Windows OpenSSH Client through Optional Features if absent. Clone or extract the project to a short ASCII path such as `C:\IronForge`. VS Code is optional. Avoid spaces, network/synchronized paths, symlinks and junctions.

Open ordinary PowerShell:

```
Set-Location C:\IronForge
.\install.cmd
.\ironforge.cmd doctor
```

Default installation checks/downloads locked archives, extracts versioned sources, prepares MSYS2 host tools, runs tool checks, builds patched native WinNFSd and installs the local VS Code extension when VS Code is present. No Administrator shell is needed. `-SkipExtension` selects CLI tools only.

Dynamic AMD and MSYS discovery

The installer and engine use explicit selections, saved settings, environment, PATH, installed directories and registry information. The explicit AMD release root contains `Vivado`, `Vitis`, `gnu` and `tps`:

```
.\install.cmd -AmdRoot '<AMD-release-directory>'
.\install.cmd -MsysRoot '<existing-msys64-directory>' -SkipExtension
```

CLI `--amd-root` and `--msys-root` precede the subcommand. Environment alternatives are `IRONFORGE_AMD_ROOT` and `IRONFORGE_MSYS_ROOT`; `XILINX_VIVADO` and `XILINX_VITIS` also participate. AMD discovery checks the native Vivado launcher and classic Linux compiler paths. `doctor` exercises Vivado/compiler/device-tree versions and static target libc linkage, then records resolved paths in `local/environment.json`.

Missing MSYS2 can bootstrap privately under `downloads/tools/msys64`. IronForge makes no global PATH, service or shortcut changes. The first private bootstrap performs a full online package upgrade. Existing shared MSYS2 installations are not automatically upgraded. An incomplete private bootstrap is deliberately not overwritten; follow the reported recovery instruction.

Preview and build

```
.\install.cmd -Plan -Build
.\install.cmd -Build -ServerAddress '<PC-LAN-IPv4>' `
  -CableSerial '<Arty-cable-serial>' -SerialPort '<COM-port>'
```

Replace `<COM-port>` with a value such as `COM7`. `-Plan` and `-WhatIf` preview; `-Help` lists options. `-Build` runs hardware, diagnostic, core software, applications, offline demo setup, boot-image packaging and complete local SPI packaging in order. It never programs the board. A server selection also prepares NFS files/commands.

Settings merge into ignored `local/settings.json`, preserving unrelated keys. Installer logs and step status use `logs/install-<timestamp>*.log` and `.json`. Stop-on-first-failure preserves the reported problem. Correct it and rerun; verified downloads are reused.

`-Offline` needs both cached verified archives and fully prepared MSYS2 packages. A base installer archive alone cannot bootstrap offline. MSYS2 repositories are rolling; local receipts record installed package versions. A complete fresh-PC installation and build remains untested. These commands describe the intended onboarding path and implemented steps, not a completed fresh-PC qualification.

6. Downloads, Versions and Compiled Components

IronForge builds real upstream Linux, U-Boot, BusyBox and application projects with native Windows cross-compilers and host tools. Tracked patches adapt the Windows build and selected board/boot integration. They do not replace the kernel, bootloader or application implementations with an emulation layer. The exact selected versions and candidate/unstable labels are retained in the inventory below.

Download and extraction procedure

`platforms/sources.lock.json` records each input's version or commit, exact internet URL, SHA-256 and destination. Archives use their explicit `filename` when supplied; otherwise the URL supplies the cache name. The `directory` field selects the extracted upstream folder, for example a versioned Linux or U-Boot directory. Vendor board XML/MIG files and prebuilt peers are downloaded inputs too, but are distinguished from compiled source.

`setup` checks existing cached bytes against the locked SHA-256 before reuse, fetches missing inputs into `downloads/`, checks downloaded bytes, then extracts/prepares source destinations and records identity. A mismatch is a failure; it is not permission to accept new bytes under an old lock. Preparation checks portable paths and extraction destinations. After changing a lock, rerun setup before rebuilding. Preserve local upstream edits before replacing an extraction.

Downloads use HTTPS and a `.partial` file, promoted to the cache name only after a matching SHA-256. A corrupt existing cache entry fails with instructions to remove that specific file; it is not silently replaced. Extraction stages under `local/extracting/`, validates the expected archive root and rejects escaping paths or incomplete destinations. Internal source links are materialized as Windows files/directories without symlink privileges. Known unused Linux netfilter/litmus-test case collisions receive recorded disambiguated names; the build rejects NETFILTER when that adaptation applies. The original verified archive retains the unmodified source. Extraction receipts record these Windows adaptations and dangling-link cases.

```
.\ironforge.cmd setup --download-only
.\ironforge.cmd setup
.\ironforge.cmd setup --offline
```

The first form fills/verifies the cache without extracting. The second prepares source folders; the third requires all necessary cached inputs. `setup --host-tools` additionally prepares MSYS2. Hash verification establishes identity against the repository lock; it is not an independent audit of upstream

code or a signature-verification claim.

`platforms/host-tools.lock.json` selects the MSYS2 bootstrap archive. Its release/checksum/documentation URLs and SHA-256 are included below. The base archive is locked; packages installed afterward come from MSYS2's rolling repositories through pacman and are recorded in local receipts. Do not claim the lock fixes every future package version or that a cached base archive guarantees offline installation.

Extensible compiled-source and download inventory

This bounded section is generated from current locks, tracked patch files, in-tree components and AMD-generated software. It includes compiled target code, compiled Windows peers/image tools and vendor loader/BSP components. Repository code is identified by repository revision; generated AMD code follows the selected installed release. Build receipts record the actual input/tool identities used on a workstation. Update the locks and regenerate this section when adding another compiled project.

Compiled components

Source ID	Exact version / commit	Source folder	Function
linux	`7.3.4`	`linux-7.3.4/`	MicroBlaze operating-system kernel and device drivers.
busybox	`1.38.0`	`busybox-1.38.0/`	Compact shell, Unix utilities, init and HTTP server for the target root filesystem.
u-boot	`2026.10`	`u-boot-2026.10/`	Boot monitor loading Linux and its device tree from NFS or SPI JFFS2.
winnfsd-source	`d833130fab6ed1fcf7fe89ecc628ad1fdf464c86`	`winnfsd-d833130fab6ed1fcf7fe89ecc628ad1fdf464c86/`	Native Windows NFS server exporting the development root filesystem.
dropbear	`2026.94`	`dropbear-2026.94/`	Target SSH server and locally compiled Windows key-generation helper.
sqlite	`3.54.1`	`sqlite-autoconf-3540100/`	Embedded SQL database used for telemetry history.

Source ID	Exact version / commit	Source folder	Function
mosquitto	`2.0.23`	`mosquitto-2.0.23/`	Target MQTT broker and locally compiled Windows publish/subscribe clients.
iperf	`2.2.2`	`iperf-2.2.2/`	Target iperf2 network-throughput server; Windows peer is a separate downloaded executable.
zlib	`1.3.1`	`zlib-1.3.1/`	Compression library compiled for the native Windows JFFS2 image builder.
mtd-utils	`2.3.0`	`mtd-utils-2.3.0/`	Native Windows mkfs.jffs2 tool for packaging the SPI root filesystem.

Repository / generated component	Version identity	Function
Native C demo	Repository revision; `tools/target/ironforge-demo.c`	Telemetry, HTTP CGI and SQLite history service.
Ethernet diagnostic helper	Repository revision; `tools/target/ethctl.c`	PHY speed advertisement control.
BRAM diagnostic firmware	Repository revision; `hardware/arty-a7-100t/diagnostic/`	DDR and Ethernet PHY bring-up checks.
AMD SREC SPI loader and standalone BSP	Installed AMD release plus generated source/input hashes in `artifacts/loader.build.json`	First-stage flash loader generated from the exported XSA; compiled and embedded in FPGA BRAM.

In-tree code has no independent release number: identify the exact checkout with `git rev-parse HEAD` and review uncommitted changes. The generated AMD loader/BSP has no archive version in the source lock; its exact identity depends on the installed vendor release and local build receipt. Do not infer that release from a downloaded component version.

Exact source downloads

Values below are copied directly from `platforms/sources.lock.json`. Status labels describe lock declarations, not completed qualification. License labels are inventory data; review upstream license files and redistribution obligations.

linux

- Version / commit: `7.3.4`
- Folder: `linux-7.3.4/`
- Download: https://cdn.kernel.org/pub/linux/kernel/v7.x/linux-7.3.4.tar.xz
- SHA-256: `b11b6df8295e8e991ff040b71aaa4994a8b55caa23b6b092907f195b5840fd1f`
- License: GPL-2.0-only
- Status: candidate

busybox

- Version / commit: `1.38.0`
- Folder: `busybox-1.38.0/`
- Download: https://busybox.net/downloads/busybox-1.38.0.tar.bz2
- SHA-256: `34f9ea6ff8636f2c9241153b9114eefa9e65674a45318ae1ef95bb5f31c53bb2`
- License: GPL-2.0-only
- Status: candidate; upstream unstable

u-boot

- Version / commit: `2026.10`
- Folder: `u-boot-2026.10/`
- Download: https://codeload.github.com/u-boot/u-boot/tar.gz/refs/tags/v2026.10
- SHA-256: `f943a88389c8fd70a6b0d061a88267465e143c7ca6ea2b75837ab20f5b60c739`
- License: GPL-2.0-or-later
- Status: candidate

winnfsd-source

- Version / commit: `d833130fab6ed1fcf7fe89ecc628ad1fdf464c86`
- Folder: `winnfsd-d833130fab6ed1fcf7fe89ecc628ad1fdf464c86/`
- Download: https://codeload.github.com/winnfsd/winnfsd/tar.gz/d833130fab6ed1fcf7fe89ecc628ad1fdf464c86

- SHA-256: `d96ca5229566d3b1bad0a1c1f83f78070f983126cd0db4af7c87d2434e643443`
- License: GPL (upstream declaration)
- Status: candidate with local READ patch

dropbear

- Version / commit: `2026.94`
- Folder: `dropbear-2026.94/`
- Download: https://www.ucc.gu.uwa.edu.au/~matt/dropbear/releases/dropbear-2026.94.tar.bz2
- SHA-256: `e098034a843699200c8c977a991fff73159735bf795d5f72ef672c41a6b1ae81`
- License: MIT (upstream multi-license LICENSE file)
- Status: candidate; maintained stable release

sqlite

- Version / commit: `3.54.1`
- Folder: `sqlite-autoconf-3540100/`
- Download: https://sqlite.org/2026/sqlite-autoconf-3540100.tar.gz
- SHA-256: `926ef8d3eab93fbf6e06011901273eec076b25594a4e1cdadd50c987f4f6d64e`
- License: blessing
- Status: stable; candidate native Windows cross-build

mosquitto

- Version / commit: `2.0.23`
- Folder: `mosquitto-2.0.23/`
- Download: https://mosquitto.org/files/source/mosquitto-2.0.23.tar.gz
- SHA-256: `47d66fc98d9f8714bdc700dfe690dbe80c9fbe93c00a21b04dc61d00e1667b34`
- License: EPL-2.0 OR BSD-3-Clause
- Status: stable; candidate native Windows cross-build

iperf

- Version / commit: `2.2.2`
- Folder: `iperf-2.2.2/`
- Download: https://downloads.sourceforge.net/project/iperf2/iperf-2.2.2.tar.gz
- SHA-256: `4008e1dc7cdeb68605747e11e954a5b25f4646ab6d71207d979acec5de7d385a`
- License: BSD-2-Clause
- Status: stable; candidate native Windows cross-build

zlib

- Version / commit: `1.3.1`
- Folder: `zlib-1.3.1/`
- Download: https://zlib.net/fossils/zlib-1.3.1.tar.gz
- SHA-256: `9a93b2b7dfdac77ceba5a558a580e74667dd6fede4585b91eefb60f03b72df23`
- License: Zlib
- Status: native mkfs.jffs2 host dependency

mtd-utils

- Version / commit: `2.3.0`
- Folder: `mtd-utils-2.3.0/`
- Download: https://infraroot.at/pub/mtd/mtd-utils-2.3.0.tar.bz2
- SHA-256: `2db102908b232406ccb20719c0f43b61196aef4534493419fbf98a273c598c10`
- License: GPL-2.0-or-later
- Status: native mkfs.jffs2 host tool

Downloaded inputs that are not compiled source

Input	Version / commit	Destination	Exact download URL	SHA-256
board-board.xml	`d8d43ed9f3f7e367ee3de82782ed601d88305677`	`boards/arty-a7-100/E.0/1.1/board.xml`	https://raw.githubusercontent.com/Digilent/vivado-boards/d8d43ed9f3f7e367ee3de82782ed601d88305677/new/board_files/arty-a7-100/E.0/1.1/board.xml	`6e781d710db6073490828a950f4a426a5e66d02623ef13b40c8757c9a665c91d`
board-preset.xml	`d8d43ed9f3f7e367ee3de82782ed601d88305677`	`boards/arty-a7-100/E.0/1.1/preset.xml`	https://raw.githubusercontent.com/Digilent/vivado-boards/d8d43ed9f3f7e367ee3de82782ed601d88305677/new/board_files/arty-a7-100/E.0/1.1/preset.xml	`ff95b9b976323fefa588873588dfa2bdce1673759c0a9e91c12ee9f2a2955bce`

Input	Version / commit	Destination	Exact download URL	SHA-256
board-part0_pins.xml	`d8d43ed9f3f7e367ee3de82782ed601d88305677`	`boards/arty-a7-100/E.0/1.1/part0_pins.xml`	https://raw.githubusercontent.com/Digilent/vivado-boards/d8d43ed9f3f7e367ee3de82782ed601d88305677/new/board_files/arty-a7-100/E.0/1.1/part0_pins.xml	`50aa844aa9c78af6342de68ae3c980f7618716447b28282878d5a106f86fed31`
board-mig.prj	`d8d43ed9f3f7e367ee3de82782ed601d88305677`	`boards/arty-a7-100/E.0/1.1/mig.prj`	https://raw.githubusercontent.com/Digilent/vivado-boards/d8d43ed9f3f7e367ee3de82782ed601d88305677/new/board_files/arty-a7-100/E.0/1.1/mig.prj	`3f72593b115e717bbc9a2e219cc321b36c2b7a5786ec32d9c6178390560862ec`
iperf-host	`2.2.2`	`apps/iperf/iperf.exe`	https://downloads.sourceforge.net/project/iperf2/iperf-2.2.2-win64.exe	`de60d0116757dde60e70d4f113c93a21dc48a36c0b30087551dac11e8eab8d1d`

Digilent board XML/MIG inputs configure hardware generation. The Windows iperf2 peer is downloaded as a binary. Neither belongs to the compiled-source table.

Host bootstrap (separate lock)

AMD Vivado/Vitis, MicroBlaze compilers, vendor libraries and licensed IP are installed separately by the user. IronForge does not download or relicense that SDK. MSYS2 bootstrap values come from `platforms/host-tools.lock.json`; subsequent online package repositories are rolling, with installed versions recorded locally.

- msys2 2026-09-27: https://github.com/msys2/msys2-installer/releases/download/2026-09-27/msys2-base-x86_64-20260927.sfx.exe SHA-256: `ad336cccfda47758b5e15cda993fbba421115cb0b126697daef1ee4dfe37209f`

Tracked patch stack

20 patch files. Archive-source patches apply in lexical filename order within each component, as implemented by `software.apply_patches` and `nfs_build.build`. The AMD loader patch applies separately with exact context to a freshly generated copy, never the installed SDK. Patch SHA-256 values below use the UTF-8/LF bytes stored by Git under `.gitattributes`, so Windows editor newline choices do not change this inventory. Local build receipts separately hash the actual working files.

Component	Patch (lexical order within component)	SHA-256
AMD generated loader	`amd-srec-0001-bounded-load.patch`	`db49608f9b363ed3a66a415d36734a1ba312c6843af612e76a1303d343939ab1`
busybox	`busybox-0004-httpd-cgi-index.patch`	`e37024e0f8b3d7bf58551402041c983328ce3c74d2290258825e2b793addf840`
busybox	`busybox-windows-host-executables.patch`	`0538f65fa7885a9222b74fe00c27ccc4f55a53e9106ae35daa8c59dcd541f903`
busybox	`busybox-windows.patch`	`28fc45814fee3a90231eed83dbb9287b9934fcb57d213afbe32f082d2ad8d898`
linux	`linux-host-cpio.patch`	`93bd3ae49f8b6c4fe6433f725e699b4f8846875784e1eb9b09759537e2cb721b`
linux	`linux-windows.patch`	`89c0e7b89639baff061a76438e516b9e886804a9b37dbdb19da2c57c385f09f2`
mosquitto	`mosquitto-0001-win32-reportevent-const.patch`	`f98540502949341c79f4b0775a0f265231283a41faf6da5ba4ea4aab1c2d8696`
mosquitto	`mosquitto-0002-winsock-setsockopt-pointer.patch`	`aa6557fada6192f497a4023539e45acf83d28867315c2825bbf0f4d00ce0f583`
mtd-utils	`mtd-utils-0001-msys-host-includes.patch`	`e2355e3e2d468fa147cf4474025e50daf6aa7aa2686159acc702e96dfd77c431`

Component	Patch (lexical order within component)	SHA-256
u-boot	`u-boot-0005-ironforge-netboot-env.patch`	`1867a56898756c0214186b8188d41337ce1ff62346c3ee91bdfb0248b4605f4a`
u-boot	`u-boot-0006-conditional-legacy-nor-env.patch`	`79070c1e29878dd1cfa2284a16f545ab45a8563ca114d9f266eccebda5a2bbde`
u-boot	`u-boot-0007-jffs2-spi-mtd.patch`	`3b8ebd588441629c1157c343638ac40c27757ae54911b57c0b450743021da3bf`
u-boot	`u-boot-0008-jffs2-crc-declaration.patch`	`211a16580a722f4047eb57926918c015a5395243999db336563a5002d4c09a19`
u-boot	`u-boot-external-dtb.patch`	`1e6919ee6b0044b73513851007d846fd0c437af7b66158317436dabac3bfbcd2`
u-boot	`u-boot-pvr.patch`	`d2b90d16aaf5629d948b4c1e2bea70ca00db08950aff5c1cdf5089a65ec7ca13`
u-boot	`u-boot-windows.patch`	`656bb3b136a0c2f3ec3905a3c537f908b5655ac9e1f2fa011b5b245354cc318a`
winnfsd-source	`winnfsd-0001-read-opaque-length.patch`	`c462a1eb80b973c0c2485bf2ddd7f542ebe4f1859349de719e94b02b96902d73`
winnfsd-source	`winnfsd-0002-umntall.patch`	`0e507779831e96d360524b1844830bf208afaba4470729e9d11330e47e3cd195`
winnfsd-source	`winnfsd-0003-find-handles.patch`	`8b7c5015744cc797388545697c74b0ae6cc52fe3e88c7eabfcb63ff26876a786`
winnfsd-source	`winnfsd-0004-tcp-records.patch`	`0c3e9b9f8c79dda864778e74a639ea665552c2d8cefc439801ba32f2ddbdec3a`

7. Patch Application and Recovery

Archive-source patches are tracked under `patches/`. Component configuration/build applies the matching `<component>-*.patch` files in filename order. Native WinNFSd is patched during its server build. Setup verifies/extracts source; the component build performs patching. The inventory in section 6 lists the exact current filenames and hashes.

`software.py` invokes MSYS `patch` with `-p1 --batch`. The flow tests forward applicability and uses a reverse dry-run to recognize an already applied exact patch. A clean forward application is performed with `--forward`; an already applied exact patch is skipped. A failed forward and reverse dry-run stops the operation for conflict review. An older applied revision of a subsequently changed patch can therefore fail on an otherwise plausible source folder.

Patch purposes

Group	Scope
Linux	Native Windows host execution/build portability and host cpio preparation
BusyBox	Windows host/executable behavior and HTTP CGI directory-index fix
U-Boot	Windows portability, external DTB/PVR integration, selected boot environment, SPI environment and JFFS2 SPI MTD reads
WinNFSd	Opaque-length handling, UMNTALL, file/directory handle behavior and TCP records
Mosquitto	Native Win32 compiler interface compatibility
mtd-utils	MSYS host include compatibility for image writer
Generated AMD SREC loader	Bounded flash/DDR access and reduced serial progress output

The SPI JFFS2 adapter `u-boot-0007-jffs2-spi-mtd.patch` is a board/storage integration change, approximately 227 additions and 15 deletions in the inspected revision; it is larger than a one-line portability fix. `u-boot-0008-jffs2-crc-declaration.patch` adds the owning CRC declaration include needed by sorted JFFS2 code. Exact inventory/patch content, rather than a blanket claim that every patch is tiny, defines the current changes.

Conflict recovery

Read the failing component log and inspect local upstream edits. Preserve the affected source folder under `local/` or another deliberate backup location. Rerun setup to extract a clean locked copy, then rebuild. The verified archive cache is reusable. Do not overwrite local modifications blindly, hand-edit away patch failures, or copy a receipt from an old build to bypass a check.

Core/server receipts record patch hashes. `logs/<component>-patch.log`, or `logs/NFS-patch.log`, records the latest executed patch command, not a cumulative patch-stack narrative. Determine the full applied set from the receipt/current files. Patch changes invalidate dependent build/package evidence.

AMD generated-copy exception

The SREC loader is generated fresh from the installed AMD software repository into `build/loader/`. `loader.py` applies `amd-srec-0001-bounded-load.patch` through exact-context logic to the generated application's `bootloader.c`. It does not run the ordinary archive-source patch pipeline or modify the installed AMD tree. Changed vendor source context stops generation for review. The freshly generated BSP is built with native AMD standalone tools and recorded with loader inputs.

On 11-16-2026, all 19 current archive-source patches across six components passed isolated touched-file replay twice against fresh archive inputs. `logs/patch-replay-results.json` records the results. The separate generated AMD loader exact-context patch is covered by the actual flash package build and loader tests. This is 19 archive-source patches plus one generated-loader patch; touched-file replay is not a complete fresh-PC installation/build qualification.

8. Workspace and Command Line API

Location	Contents
`src/`, `hardware/`, `platforms/`, `patches/`	Tracked engine/extension, hardware, configuration and upstream changes
`tools/`, `tests/`	Tracked host/target helpers and checks
`downloads/`	Ignored archive cache, vendor board files and private host tools
Source-lock `directory` folders	Ignored versioned upstream extractions
`build/<logical-component>/`	Ignored objects, final `.config`, generated tools and preparation state
`rootfs/`	Sole editable target filesystem/NFS export; generated files ignored
`rootfs/boot/`	Generated `uImage` and `system.dtb`
`build/filesystem/`	Generated JFFS2/initramfs and metadata staging
`dts/`	Generated platform DTS/DTB
`artifacts/`, `artifacts/apps/`	Deployable outputs and checked build/image receipts
`artifacts/flash-backups/`	Timestamped complete SPI backups and operation receipts
`local/`	Machine settings, credentials, environment/state and preparation receipts
`logs/`	Installer, build, tool, serial and operation evidence

Source paths derive from lock `directory` values. Logical build paths such as `build/linux/` remain stable when source versions change. Do not create unversioned source aliases or duplicate editable rootfs trees. Do not delete `rootfs/` as a routine clean step: it contains editable files and local provisioning.

`ironforge.cmd` / `ironforge.ps1` launch the Python CLI. Global `--amd-root` / `--msys-root` options precede the subcommand. Resource locks protect build/package/board resources from conflicting IronForge operations.

Command	Responsibility
`setup`, `doctor`	Prepare verified sources/tools; inspect actual environment
`hardware [--generate-only] [--jobs N]`	Generate or implement/export FPGA design
`diagnostic`	Compile BRAM DDR/PHY diagnostic
`configure linux	u-boot
`menuconfig <package>`	Interactive native Kconfig UI
`software <component> [--jobs N]`	Build core/application, `apps` or `all`
`package`	Validate inputs and assemble boot/rootfs outputs
`board identify --cable <serial>`	Select/remember intended cable
`board program	memtest
`serial`	Native monitor on configured COM port
`netboot build-server	prepare
`flash package	inspect
`demo setup	status

`software` accepts `busybox`, `u-boot`, `linux`, `all`, `apps`, `dropbear`, `sqlite`, `mosquitto`, `iperf`, `demo`. `all` builds core then apps; `apps` omits core. Jobs default to four where exposed; some recipes use conservative fixed parallelism.

VS Code and package extension

The optional local extension presents the IronForge Explorer and invokes the same backend/settings. Menuconfig runs in an interactive native terminal. `platforms/packages.json` version 1 contains unique `id`, `label`, `source_id` or workspace-relative `source_dir`, separate `build_dir` and optional `menuconfig_target` / `menuconfig_program`. Locked `source_id` takes precedence. Paths stay in the workspace. The runner supplies `O=`, `ARCH=microblaze`, the cross compiler and MSYS tools; packages must support that interface. Optional `configure` / `build` flags expose existing actions; an entry alone does not implement a new recipe.

9. Build Order and Configuration

Setup prepares sources/tools. Hardware exports XSA/bitstream/MMI. Diagnostics use the bare-metal compiler. Core builds prepare upstream host programs, generated device tree, configuration and target outputs. Application recipes compile static target executables/libraries and Windows peers. Demo setup prepares persistent configuration/keys. Package installs checked generated files. Flash package generates AMD loader/BSP, embeds BRAM code, writes JFFS2 and assembles SPI.

```
.\ironforge.cmd hardware
.\ironforge.cmd diagnostic
.\ironforge.cmd software all
.\ironforge.cmd demo setup
.\ironforge.cmd package
.\ironforge.cmd flash package
.\ironforge.cmd flash inspect
```

Compiling applications can proceed while the old NFS system runs; installation into `rootfs/` requires the stopped-system sequence in section 18.

Configuration derivation

Component	Fresh upstream start	Tracked fragment	Resolution
Linux	`mmu_defconfig`	`platforms/arty-a7-100t/linux.cfg`	`olddefconfig`
U-Boot	`microblaze-generic_defconfig`	`platforms/arty-a7-100t/u-boot.cfg` plus local/layout settings	`olddefconfig`
BusyBox	`allnoconfig`	`platforms/arty-a7-100t/busybox.cfg`	`oldconfig` with defaults

Fragments override upstream defaults; they are not full final configurations. Dependencies and hidden defaults come from the selected upstream version. U-Boot incorporates server settings and environment geometry derived from the platform. The generator creates a DTB matching addresses, clocks, interrupt types, PHY selection and flash partitions.

An existing `build/<component>/.config` survives ordinary rebuilds. Use:

```
.\ironforge.cmd configure linux
.\ironforge.cmd menuconfig linux
.\ironforge.cmd software linux
.\ironforge.cmd package
```

Preserve final `.config` or promote intentional changes into tracked fragments to reproduce them on another PC. Before deliberately reinitializing configuration, save the current file/source edits. Compare final configurations after source upgrades; fragment text alone does not show all resolved settings.

Linux includes MMU, ELF/scripts, devtmpfs/proc/sysfs/tmpfs, UART console, EthernetLite/DP83848, DHCP, NFSv3 root, SPI NOR MTD/block and zlib JFFS2. Size settings disable modules, IPv6, USB, display/audio, PCI and unrelated filesystems/debug facilities. Kernel symbols and `/proc/config.gz` remain available for diagnosis. The platform does not expose every upstream feature.

BusyBox is static and provides init, ash, file/process tools, mounts, DHCP, network diagnostics and HTTP/CGI. Standalone/no-fork ash reduces process startup for built-in applets. U-Boot includes Ethernet/DHCP/NFS, SPI NOR/redundant environment, JFFS2, DTB and legacy gzip uImage support. SPL/EFI are unused.

Default build metadata uses current UTC wall-clock time. Set `SOURCE_DATE_EPOCH` to a Unix timestamp for controlled timestamps. Matching binaries also requires repository, archives, AMD/MSYS tools, settings and final configurations to match. Runtime Linux time is separate; no RTC/time synchronization is configured for this demo.

10. FPGA Modules, Clocks and Reset

`hardware/arty-a7-100t/build.tcl` generates the design. Vendor IP resolves from the installed Vivado definitions; exact IP versions belong to that project/tool release. Board pins and MIG geometry come from verified Digilent board files and tracked constraints.

Processor and memory

MicroBlaze uses little endian, MMU mode 3/two zones, interrupts/debug, hardware divide/multiply, barrel shift, pattern compare and hardware floating point. Instruction/data caches are 16 KiB each with 32-byte/16-byte lines respectively. DDR `0x80000000..0x8fffffff` is cacheable; data-cache writeback is disabled. Platform compiler flags match these capabilities; mismatch can cause illegal instructions or incorrect execution.

MicroBlaze automation adds 64 KiB LMB BRAM for vectors, diagnostics and first-stage loader. The diagnostic runs there with caches disabled while overwriting DDR. The loader runs there while placing U-Boot in DDR. Exported `system.mmi` identifies BRAM for `updatemem`.

MIG 7-series controls 256 MiB DDR3. The script retains vendor geometry/pins and changes MIG system-clock input to the buffered clock-wizard output. Calibration crosses into the CPU domain through the reset guard. SmartConnect connects three processor AXI initiators to DDR and six peripheral targets, with MIG UI as a second clock domain. A separate processor-reset instance controls DDR-facing AXI reset in that domain.

Peripheral cores

AXI UARTLite provides the 115200 console used by loader, U-Boot and Linux. UART RX is asynchronous and uses the vendor synchronizer plus an explicit timing exception. UART is the first diagnostic path for failed network/filesystem startup.

AXI Timer provides the Linux timing interrupt, level-high at controller input zero. AXI INTC gathers timer, EthernetLite, UARTLite and SPI and interrupts the processor. The kind mask `0x0e` makes Ethernet/UART/SPI rising edge and timer level. The generated DT matches it.

AXI EthernetLite supplies 10/100 MII, MDIO and transmit/receive ping-pong buffers. It connects to the DP83848 through MII/MDIO, 25 MHz reference and reset. It is not a gigabit/DMA MAC. Diagnostics scan PHY IDs and record the discovered address. Correct edge-triggered receive notification matters: an earlier all-level configuration lost notifications.

AXI Quad SPI is a standard 8-bit master, one slave select, 16-entry FIFO, divider 16. The 100 MHz source yields 6.25 MHz SCK. STARTUPE2 drives dedicated configuration CCLK internally; ordinary pins carry data/select and WP#/HOLD# stay high. IRQ remains wired, but Linux's generated node omits the optional interrupt property to select polling and avoid measured per-FIFO overhead.

AXI GPIO exposes one synchronized readiness input. XLConcat gathers IRQs. XLConstant ties unused SPI inputs/control to defined levels. The MicroBlaze Debug Module supplies JTAG debug access within the selected cable flow; no network debug service is installed on Linux.

Clocks and reset

Clock	Frequency	Use
Board oscillator	100 MHz	Clock wizard input
Wizard output 1	100 MHz	CPU, AXI, MIG system input, SPI source
Wizard output 2	200 MHz	MIG reference
Wizard output 3	25 MHz	PHY reference
MIG UI	Generated by MIG	DDR AXI; inspect resolved project value

`reset_guard.v` asynchronously asserts its reset-release synchronizer when board reset or clock lock drops; release takes two edges. A second two-stage synchronizer samples calibration. `cpu_ready` requires released reset and synchronized calibration. PHY reset requires reset release, MAC reset release and a saturating 20,000,000-cycle delay, nominally 200 ms. Synchronizer/counter registers initialize to zero.

The hardware build checks setup/hold/pulse width, timing coverage, bus skew, asynchronous output bounds and critical DRCs before exporting outputs. Timing, CDC, clock interaction, I/O, exception and SPI/MDIO reports remain under `build/hardware/`. Explicit external asynchronous exceptions are deliberate; review the checked categories/reports when interpreting timing output.

11. Memory Map and Flash Layout

CPU address map

Resource	Base	Span / role
Local BRAM	`0x00000000`	64 KiB; vectors/diagnostic/loader
Status GPIO	`0x40000000`	64 KiB AXI window
UARTLite	`0x40600000`	64 KiB AXI window
EthernetLite	`0x40c00000`	64 KiB AXI window
AXI INTC	`0x41200000`	64 KiB AXI window
AXI Timer	`0x42000000`	64 KiB AXI window
AXI Quad SPI	`0x44a00000`	64 KiB AXI window
DDR3	`0x80000000`	`0x10000000` / 256 MiB

Vendor drivers/product guides define peripheral register fields and side effects. The 64 KiB mappings are decode windows, not a claim that every byte is implemented. This specification defines integration addresses rather than duplicating every vendor register definition.

Boot RAM regions

Item	Address / bound	Use
Decompressed kernel	`0x80000000`	Kernel destination
Kernel limit / uImage load	`0x82000000`	Kernel window bound and compressed image staging
DTB staging	`0x84000000`	Relocatable boot DTB
Initramfs staging	`0x84200000`	Separate fallback archive
U-Boot link/load/entry	`0x87e00000`	SREC destination
Loader destination limit	`0x88000000`	Exclusive upper bound
U-Boot reserved-start profile	`0x8c000000`	High-memory separation

Image/ELF checks enforce relevant architecture, load and size bounds. These regions are not scratch space while a live bootloader/kernel/decompressor owns them. Diagnostics overwrite all DDR and cannot run against a live OS.

Five-partition SPI map

`platform.json` defines 16 MiB total capacity, 64 KiB erase size, bus zero, chip-select zero and 6.25 MHz maximum clock.

Partition	Offset	Size	Last byte	Payload / Linux protection
`fpga`	`0x000000`	`0x300000`	`0x2fffff`	Configuration/BRAM loader; read-only
`u-boot`	`0x300000`	`0x1c0000`	`0x4bffff`	U-Boot SREC; read-only
`u-boot-env`	`0x4c0000`	`0x010000`	`0x4cffff`	Primary environment; writable
`u-boot-env-redund`	`0x4d0000`	`0x010000`	`0x4dffff`	Redundant environment; writable
`rootfs`	`0x4e0000`	`0xb20000`	`0xffffff`	11.125 MiB JFFS2 including `/boot`; writable

There are no separate kernel/DTB partitions. Layout validation checks geometry, alignment and coverage; generated DTS and U-Boot storage settings use the same profile. Do not reuse an old partition index by hand: current rootfs is the fifth MTD partition, normally `mtdblock4`.

Environment sectors contain checksummed data and redundant active-state handling. `saveenv` writes them in either boot mode. Linux read-only annotations reduce accidental runtime writes but do not protect against full-image JTAG programming. Current layout boot and saved-environment persistence across FPGA reconfiguration are verified.

12. Root Filesystem and Windows Metadata

Windows files do not directly encode Linux UID/GID, executable modes, symlinks or device nodes. IronForge retains portable files in the one editable `rootfs/` tree and records generated target metadata in `.ironforge-metadata.json`. This JSON file is packaging output, not a second user-maintained filesystem or an interface for hand-authoring arbitrary symlinks. Archive/image writers consume its records instead of inferring Linux semantics from Windows permission bits. Assembly requires no administrator account or native Windows symlink privilege.

BusyBox names: Windows, NFS and Linux images

BusyBox contains many applets in one executable and selects an applet from the invoked program name (`argv[0]`). The same intended applet names have different representations at the host/export boundary and in Linux image formats:

Representation	What `/bin/ls` and other generated applet names contain	Storage and execution behavior
Windows `rootfs/`	NTFS hardlinks to `rootfs/bin/busybox` when `os.link` succeeds; ordinary copies when it fails	Hardlinked names share one NTFS file record/inode and data allocation; copies consume separate storage
Windows NFS export	Regular executable file names backed by those hardlinks or copies	They are not exported applet symlinks; Linux invokes the requested name and BusyBox dispatches using `argv[0]`
Generated newc CPIO/initramfs	Linux `S_IFLNK` entries whose link-target bytes are `/bin/busybox`	Linux resolves the symbolic link to the one BusyBox executable
Generated JFFS2	Linux symbolic-link entries reconstructed from target metadata	The filesystem stores `/bin/busybox` as the link target, not another copy of the executable

On NTFS, `Get-Item` reports the full executable length for every hardlinked name. Adding those apparent lengths does not measure allocated storage: when hardlink creation succeeded, the data belongs to the shared file rather than being duplicated for each directory entry. The fallback path

writes complete independent copies if `os.link` raises an error, so a universal single-inode/storage claim would be incorrect. The current observed BusyBox executable and Windows applet names each have an apparent length of 1,294,944 bytes; that value changes when BusyBox is rebuilt.

Packaging stages a new BusyBox file and replaces the `bin/busybox` inode, then removes/recreates the generated applet names. It does not rewrite the old hardlinked file's bytes in place, which protects any unrelated hardlink still referring to the old file. The source binary remains `artifacts/busybox`; `rootfs/bin/busybox` and applet names are generated installations. Do not edit one hardlinked name expecting it to be independent: writing its data in place can change every name sharing that inode. Preserve intentional changes in the generating source/configuration instead.

From the repository root, these read-only PowerShell checks distinguish apparent length from hardlink identity:

```
Get-Item .\rootfs\bin\busybox, .\rootfs\bin\ls, .\rootfs\sbin\init |
  Select-Object FullName, Length
fsutil hardlink list .\rootfs\bin\busybox
fsutil file queryfileid .\rootfs\bin\busybox
fsutil file queryfileid .\rootfs\bin\ls
```

On a supporting NTFS volume, matching file IDs and the hardlink list establish sharing. If a particular applet is absent from that list or has a different file ID, check whether the copy fallback was used; matching lengths alone do not establish a hardlink. These commands inspect the Windows tree, not Linux image link records.

On the JFFS2 target, inspect the reconstructed Linux links:

```
ls -l /bin/busybox /bin/ls /bin/mkdir /sbin/init
```

The current target observation shows `/bin/ls`, `/bin/mkdir` and `/sbin/init` as 12-byte symbolic links to `/bin/busybox`, while the executable itself is 1,294,944 bytes. Twelve bytes is the length of the target string, not a second executable. On the NFS root those applet names are regular files instead, by design. The current BusyBox configuration does not include the `readlink` applet; if it is deliberately enabled in a later configuration, `readlink /bin/ls` is another way to inspect the Linux link target. Do not rely on that command in the current image.

Supported metadata and input boundaries

`/dev/console` and `/dev/null` are metadata-defined character devices rather than Windows special files. The newc writer emits target modes, ownership and device numbers directly, including generated applet `S_IFLNK` entries. Private provisioning directories/files receive restrictive defaults. Application installation has its own generated install record.

Traversal rejects native Windows symlink/reparse-point inputs, unsupported types, nonportable names and case ambiguities. Required generated paths/reserved devices are checked. Modified generated destinations produce an error rather than silent overwrite. The implemented symbolic-link reconstruction is specifically for generated BusyBox applets targeting `/bin/busybox`; generic user-authored relative/absolute, directory or dangling Linux links are not implemented by this interface. Do not infer generic symlink support from the generated applet case or try to bypass traversal by editing metadata. Editable rootfs files remain build inputs; these checks do not audit their application behavior.

JFFS2 construction

`flashfs.py` snapshots files with explicit mode, UID/GID, generated applet link targets, hashes and device records. For each applet, it checks the regular host file against generated metadata and reconstructs the symbolic link in disposable JFFS2 staging. The staging script sets `MSYS=winsymlinks:sys` and invokes MSYS `ln -s`; this uses MSYS symlink representation without native Windows symlink privileges. The MSYS-built `mkfs.jffs2` interprets those entries as symlinks and emits real Linux symbolic-link nodes in JFFS2. It does not merely copy Windows hardlink data under every applet name.

Staging/device tables feed upstream `mkfs.jffs2`. `tools/flashfs-build.sh` compiles that host writer from locked mtd-utils/zlib and the tracked MSYS include patch. It builds the writer needed on Windows, not Linux-only flash access utilities. Staging is generated and disposable; it does not add a second editable rootfs.

`build/filesystem/rootfs.jffs2` is little-endian, zlib-compressed and checked against partition/erase capacity. It includes `/boot/uImage` and `/boot/system.dtb`, loaded by U-Boot before Linux mounts the filesystem. Packaging-only control/metadata files are omitted according to the writer. Staging is disposable; `rootfs/` remains the editable input.

Initramfs and runtime mounts

Packaging also produces `build/filesystem/rootfs.cpio.gz` with explicit newc metadata and gzip. Boot kernel/DTB files are omitted from that archive to avoid duplicating boot payloads. Linux leaves `CONFIG_INITRAMFS_SOURCE` empty; the archive is separate. Standalone initramfs acceptance remains open.

Init mounts kernel virtual filesystems and RAM-backed runtime areas. `/tmp/ironforge/demo.db` and `/tmp/ironforge/` logs are volatile. Root credentials are staged into private tmpfs because NFS permissions differ from a native root filesystem. Persistent provisioning inputs remain in the local Windows rootfs private area.

JFFS2 changes survive FPGA reconfiguration unless flash is reprogrammed. A full-image program replaces rootfs/environment with packaged bytes. NFS changes live on Windows; editing while Linux holds handles can cause stale-handle/caching problems. Halt Linux and restart the export after installing changes. Export volatile CSV/log data before stopping the board.

13. NFS Bring-up and Operation

Select the PC's reachable LAN IPv4 before building U-Boot if it should become compiled default `serverip`. Changing that default requires U-Boot rebuild/package; runtime `setenv serverip` overrides it immediately.

```
.\ironforge.cmd netboot prepare --address '<PC-LAN-IPv4>'
.\ironforge.cmd netboot serve
```

Keep the serving terminal open throughout NFS-root operation. Patched native WinNFSd exports only `rootfs/` as `/ironforge`, not the entire Windows filesystem. It refuses occupied TCP/UDP ports 111, 1058 and 2049. Use `netboot build-server` after server source/patch/tool changes.

Firewall

Preview ordinary-user rules; apply from Administrator PowerShell only when necessary:

```
.\tools\netboot-firewall.ps1 -Address '<PC-LAN-IPv4>'
.\tools\netboot-firewall.ps1 -Address '<PC-LAN-IPv4>' -Apply
```

Rules bind the built executable/selected local address and permit the required TCP/UDP ports from the local subnet. `-ClientAddress '<Arty-IPv4>'` narrows to one client. Installation does not apply firewall changes or create a service.

Board selection and diagnostic

```
.\ironforge.cmd board identify --cable '<Arty-cable-serial>'
.\ironforge.cmd board program
.\ironforge.cmd board memtest
.\ironforge.cmd board netboot
```

The dedicated hardware server filters the selected cable and rejects an occupied selected server port. It does not adopt an arbitrary existing server. Other devices, including Zybo, remain outside that selection. Open serial separately before `netboot`.

`program` loads the normal bitstream. `memtest` executes the destructive BRAM diagnostic and records matching hardware/cable success. `netboot` loads U-Boot into already programmed hardware; it does not program FPGA configuration. State can be reused only while the same hardware remains programmed. Repeat programming/diagnostic after power loss or another design.

The diagnostic checks synchronized DDR readiness, walking data bits and two address-derived patterns across all 256 MiB, then scans MDIO for the expected PHY family. It records the discovered address and matching input identities. This is bring-up coverage, not an exhaustive reliability test.

U-Boot and Linux

Interrupt the three-second countdown at 115200 8N1. At `IronForge>`:

```
setenv serverip <PC-LAN-IPv4>
run netboot
```

The first command is unnecessary if the existing environment is correct. Netboot uses DHCP, NFS-fetches both `/boot` files, clears `fdt_high` and selects NFSv3/TCP root. Inspect `printenv` if saved settings override current defaults. After Linux starts, inspect `uname -a`, `/proc/mounts`, `dmesg`, network address and HTTP status; perform a small intended root write/read check. Ping alone does not establish NFS writes or service startup.

14. SPI Image, Programming and Recovery

```
.\ironforge.cmd package
.\ironforge.cmd flash package
.\ironforge.cmd flash inspect
```

Flash packaging validates ordinary package inputs, then uses Vitis native HSI through `sdtgen.bat` to generate installed AMD `srec_spi_bootloader` and standalone BSP into a fresh directory. The exact-context patch modifies only that app copy. Native tools compile BSP/BRAM ELF. Checks include BRAM/stack bounds and U-Boot SREC checksum, record length, overlap, entry point, CRLF/S3 format and DDR destination limits.

The loader bounds reads to the U-Boot partition and writes to the allowed DDR window, and limits serial progress output. Vendor context changes stop patching. `updatemem` uses the MMI to embed the loader in a separate flash bitstream; normal `system.bit` stays available for JTAG.

Assembly places FPGA/loader, U-Boot SREC, two initialized environments and JFFS2 at profile offsets. `artifacts/ironforge-arty-a7-flash.bin` is exactly 16 MiB. `artifacts/flash.manifest.json` records layout, payload/input hashes, sizes and provisioning status. `flash inspect` checks bytes and current inputs again. The image is marked `public_release: false`: private SSH provisioning makes it a local demo image, not a generic public release.

Identify, preserve and program

Halt Linux (`sync; halt`). Load current normal hardware, diagnostic and U-Boot via the selected JTAG flow as needed, leave serial at `IronForge>`, then close the console for probe ownership:

```
.\ironforge.cmd flash probe --cfgmem-part '<exact-Vivado-part>'
.\ironforge.cmd flash backup
.\ironforge.cmd flash program
.\ironforge.cmd flash boot
```

Probe records actual JEDEC/geometry and exact matching Vivado memory part through the selected COM/cable. Backup reads all 16 MiB into timestamped `artifacts/flash-backups/`. Program creates/validates another full backup before erase, full-image program and verification. Backup/program temporarily replace FPGA configuration with AMD's indirect flash bridge; they cannot run against active Linux.

`flash boot` requests FPGA reconfiguration using startup mode pins; fit JP1 for Master SPI. It is not physical power removal. Linux has no restart controller: `reboot` halts. Use clean halt followed by reconfiguration or physical cycling.

Environment and recovery

Inspect `printenv boot netboot bootcmd serverip` and current partition variables when qualifying a new image. `run boot` uses SPI/JFFS2 `fsload` through the MTD adapter. `run netboot` retains DHCP/NFS fetch and NFS root. Both replace boot arguments. Saved SPI state may override newly compiled defaults.

Full-image programming replaces environment/rootfs. `saveenv` writes redundant sectors even during NFS work. If startup fails, interrupt autoboot and inspect/restore intended values from the current build, saving only after checking addresses/layout. JTAG, a known-good build set and complete backups provide recovery. Keep receipts/hashes with backups. There is no A/B kernel/rootfs rollback, atomic update claim or current five-partition recovery qualification.

15. Linux Applications and Runtime API

BusyBox init starts the onboard Mosquitto broker before native telemetry, plus HTTP and iperf2. SQLite records history; Dropbear supplies key-only SSH. The dashboard is white with a top-left IronKernel logo linked to `https://www.IronKernel.com`, desktop/mobile layouts and uptime-based graphs. Unsynchronized clock timestamps are labeled accordingly.

```
.\ironforge.cmd software apps
.\ironforge.cmd demo setup --address '<Arty-IPv4>' --interval 5
.\ironforge.cmd package
```

Setup prepares keys/configuration offline without contacting the board. Optional `--address` remembers the board for host clients. Telemetry always uses onboard `127.0.0.1:1883`; no Windows broker is needed. Stop Linux/NFS before setup/package changes the served rootfs.

HTTP and SQLite

Open `http://<Arty-IPv4>/`. BusyBox HTTP serves the page and native CGI:

Endpoint	Function
`/cgi-bin/status`	Current telemetry/status
`/cgi-bin/history`	SQLite history query
`/cgi-bin/csv`	CSV export

`/tmp/ironforge/demo.db` retains the latest 3600 readings, about five hours at five seconds. Other intervals change retained duration. `/tmp` is RAM-backed; export CSV before shutdown. SQLite is local, not a network database service. Recorded HTTP index checks compare root, `/index.html`, query-bearing root, HEAD and CGI failures; repeat after HTTP/BusyBox changes.

MQTT

The onboard broker listens TCP 1883 and drops to `mosquitto` UID/GID 1883. Persistence is disabled. Limits are 16 connections, 100 queued messages/client and 16 KiB payloads. There is no TLS or authentication; use a trusted LAN.

Topic	Payload / behavior
`ironforge/arty/status`	Telemetry samples
`ironforge/arty/control`	`interval=N`, N from 1 through 60 seconds
`ironforge/arty/ack`	Control acknowledgement

```
.\ironforge.cmd demo status --address '<Arty-IPv4>'
.\ironforge.cmd demo mqtt --address '<Arty-IPv4>'
.\ironforge.cmd demo mqtt --address '<Arty-IPv4>' --interval 2
```

The helper subscribes for a sample or sends interval control. Logs are `/tmp/ironforge/mosquitto.log`. To establish onboard routing, test with any Windows broker stopped.

SSH and iperf2

SSH uses Windows OpenSSH, persistent developer/Dropbear keys and strict host-key checks. Setup reuses ignored `local/ssh/developer_ed25519`, records expected identity in `local/ssh/known_hosts`, and stages private input under ignored `rootfs/etc/ironforge-private/`. Boot places root credentials in private tmpfs. Inspect identity conflicts; do not bypass verification. SFTP is absent.

```
.\ironforge.cmd demo ssh --address '<Arty-IPv4>'
.\ironforge.cmd demo iperf --address '<Arty-IPv4>' --seconds 10
```

The board starts `iperf -s -D` on TCP 5001. Windows uses iperf2, not incompatible iperf3. Duration accepts 1..300 seconds. Setup compiles Windows Mosquitto clients/Dropbear key helper and stages the locked official iperf2 executable. Host peers run foreground; connection setup/cold executable reads can take seconds even when commands inside an established session are faster.

16. Build Evidence and Packaging Gates

Receipts identify input/output combinations and let subsequent operations reject stale artifacts. They check known files; they are not proof against every untracked source modification or hardware failure. Keep receipts with tested images/backups.

Stage	Checked/recorded evidence
Source setup	Locked archive identity, SHA-256, extraction record
Preparation	Relevant generated/configuration inputs and outputs
Core build	Source identity, patch hashes, final configuration, output hashes
U-Boot	Generated DTB identity and loader compatibility
Applications	Recipe/source/input identity, target/host output hashes
Hardware	Profile/design correspondence, bitstream/XSA checks
Package	Current receipts/configurations, platform, source lock, patches and installed outputs
Board	Selected cable, programmed bitstream and matching diagnostic pass
Loader/flash	Generated vendor inputs, BRAM/SREC bounds, capacities, payload/image hashes
Flash operation	Actual identity/geometry, complete backup and program/verify evidence

`provenance.py` validates expected core output sets/digests, final `.config`, patch hashes, lock/extraction identity and U-Boot DTB. Manifest validation compares platform, source lock, tracked patch set, build configurations and application receipts. Flash inspection checks complete image size/hash/layout and current inputs. A changed input should produce rebuild instructions rather than loading plausible stale bytes.

Source identity covers archive/extraction identity, not a full recursive hash of every upstream local edit. Preserve/record edits deliberately when reproducing a result. Controlled timestamps do not substitute for source/tool/configuration matching. MSYS package receipts describe the actual rolling-repository selection on that PC.

Resource locks coordinate IronForge operations but do not detect every external action. Another hardware server, manual FPGA programming or concurrent rootfs editing can invalidate assumptions. Repeat program/diagnostic after external programming or power loss. Package locks do not authorize stopping another user's processes or operating another cable.

Agent reports should state what command ran, whether it passed, meaningful skips/failures, where evidence is retained and what remains untested. A build-only pass cannot be reported as an actual boot, a reconfiguration pass as a cold power-cycle pass, or a baseline capture as current-layout evidence.

17. Source Modules and Host Scripts

Engine modules

File under `src/ironforge/`	Responsibility
`__main__.py`	Parse CLI, settings, dispatch and resource locks
`tools.py`	AMD/MSYS discovery and path conversion
`sources.py`, `host_tools.py`	Locked download/extraction and private bootstrap
`msys.py`	Controlled host/cross environment and host-program selection
`process.py`, `locking.py`	Child processes/logging and resource coordination
`preparation.py`	Reusable configuration/generated preparation records
`software.py`	Core patch/configure/build and receipts
`menuconfig.py`	Package manifest and interactive Kconfig
`platform.py`	Hardware/profile/ELF and interrupt-type validation
`images.py`	DTS/DTB, rootfs/newc and boot packaging
`provenance.py`	Current receipt/manifest checks
`apps.py`, `app_recipes/*.py`	Application builds, install validation and recipes
`demo.py`	Offline credentials/config and Windows clients
`nfs_build.py`, `netboot.py`	Patched server build, boot preparation/export
`board.py`	Filtered hardware-server lifecycle and board evidence
`flash_layout.py`	Geometry/partition checks and derived values
`loader.py`	Fresh AMD app/BSP, exact patch, BRAM/SREC validation
`flashfs.py`	Explicit metadata, host JFFS2 tool and image sizing
`flash.py`	Environment/image assembly and inspection
`flash_probe.py`, `flash_board.py`	Serial identity and indirect bridge operations

Entrypoints and scripts

Path	Responsibility
`start.md`	Agent/developer entry, scope and evidence guidance
`install.cmd`, `install.ps1`	Native installer, previews, preparation/build steps and status
`ironforge.cmd`, `ironforge.ps1`	CLI launch and AMD Python selection
`hardware/arty-a7-100t/build.tcl`	Generate/implement hardware and timing/DRC/export gates
`hardware/arty-a7-100t/board.tcl`, `load.tcl`	Selected debug/program/load actions
`hardware/arty-a7-100t/board.xdc`	Board pins and external constraints
`hardware/arty-a7-100t/reset_guard.v`	Reset/calibration synchronization and PHY delay
`hardware/arty-a7-100t/diagnostic/`	BRAM startup/linker, DDR/PHY program and reset simulation
`tools/host-setup.sh`	MSYS2 package preparation/receipt
`tools/msys-menu-env.sh`	Menuconfig terminal environment
`tools/build-loader.tcl`	Native HSI app/BSP generation
`tools/flashfs-build.sh`	Compile upstream host JFFS2 writer/zlib
`tools/flash-images.tcl`, `flash-board.tcl`	Vivado configuration-memory image/device actions
`tools/flash-probe.ps1`	U-Boot serial flash-identification interaction
`tools/netboot-firewall.ps1`	Preview/apply scoped NFS firewall rules
`tools/nfs-smoke.py`	NFS protocol smoke checks
`tools/serial.ps1`	Native UART monitor
`tools/build-ethctl.py`, `target/ethctl.c`	Target Ethernet helper build/source
`tools/target/ironforge-demo.c`	Native telemetry, SQLite/MQTT and CGI
`tools/version.tcl`	Vivado doctor version probe
`src/extension/package-vsix.py`	Local extension packaging
`documentation/build_pdf.ps1`, `pdf/build.mjs`	Documentation rendering, separate from target packaging

18. Rebuild and Update Workflows

Application/dashboard update

Compile apps while the old system runs if useful. Before installing into its served rootfs, run on Linux:

```
sync
halt
```

Stop Windows `netboot serve` with Ctrl+C, then:

```
.\ironforge.cmd software apps
.\ironforge.cmd demo setup
.\ironforge.cmd package
.\ironforge.cmd netboot prepare --address '<PC-LAN-IPv4>'
.\ironforge.cmd netboot serve
```

Reload/boot using the selected qualified board flow. Restarting the export supplies fresh handles. For SPI deployment also package/inspect, back up and program the new flash image; editing Windows files does not alter board-local flash.

Core settings or source changes

Use configure/menuconfig/build/package for the changed component. Preserve the final `.config` or promote intended changes into its fragment. BusyBox changes can alter applet installation/metadata. U-Boot defaults require a new loaded/flashed U-Boot, and saved environment may still override them.

For changed locks/patches, preserve upstream edits, rerun setup, rebuild affected/dependent components and repackage. An older applied patch may require clean extraction. Rebuild associated host tools as required. Never copy an old receipt to satisfy a new input check.

Hardware changes

Regenerate/implement hardware, rebuild diagnostic, regenerate matching DTB/dependent software/package inputs, then loader/flash image. Program the normal bitstream and rerun DDR/PHY before loading boot software. Recheck interrupts, flash reads and services: a timing-clean netlist does not establish driver compatibility.

Host/network changes

Select local AMD/MSYS paths, cable, COM and server address on each PC. Do not copy another PC's settings/keys wholesale. Multiple boards need unique locally administered unicast MAC addresses; generator checks address form, not LAN-wide uniqueness. Use detected PHY selection. Rebuild U-Boot if its compiled server default should change.

Preservation and restore

Retain repository revision, locks/patches, final configs, nonsecret settings, receipts/logs and complete flash backups. Protect private credentials separately. Export CSV and volatile logs before shutdown. A checkout alone omits ignored build inputs/state and local provisioning.

Full-image restore replaces environment/rootfs; preserve current contents first. JTAG is the recovery entry if SPI boot or saved commands fail. Future update tooling must handle kernel/DTB inside writable rootfs and power failure during erase/program. The present workflow does not claim atomic updates, automatic rollback or flash endurance qualification.

19. Verification and Recorded Measurements

Tests under `tests/` cover configuration, layout, Windows preparation, application recipes, image formats/bounds, filesystem metadata, stale-receipt rejection, boot environment and mocked flash/board flows. Native harness/simulation checks need corresponding host tools and may skip when absent. Installer fixtures use a temporary fake checkout without downloads, builds or hardware:

```
powershell -NoProfile -ExecutionPolicy Bypass -File tests/test-install.ps1
```

Run Python unittest discovery with the environment's Python for current suite evidence. Report the actual command, total, skips and failures. A prior suite count does not establish current boot-layout acceptance. Current full-build and hardware test results must be attached to the tested revision.

Current host build and filesystem evidence

On 11-16-2026, the native U-Boot build, all five application builds, ordinary package and complete flash package passed on the development PC. The current Python suite ran 176 tests with five skipped and no failures. A separate five-test filesystem run decoded the real generated JFFS2 and passed CRC, Linux mode and `/boot` byte/hash checks. Matching current-image board evidence is retained in `logs/unified-first-spi-boot.log` and its capture JSON, `logs/unified-spi-runtime.log`, `logs/unified-white-dashboard-check.log`, and `logs/demo-ssh-verification.json` with its command log.

Current generated item	Bytes	Meaning
Loader-initialized FPGA payload	2,247,868	Fits FPGA partition
U-Boot SREC	1,641,988	Fits bounded SREC partition
Each environment sector	65,536	Two separate redundant sectors
DTB	20,552	File stored inside JFFS2
JFFS2 image	9,540,076	Includes kernel, DTB and userspace
Rootfs partition	11,665,408	11.125 MiB capacity
Free whole erase blocks	2,097,152	32 complete 64 KiB blocks beyond rounded image use
Complete flash image	16,777,216	Entire device including unused fill

The recorded generated image SHA-256 is `7bac2de1677d7cc823fd5de061d262ea86aa189ffb18f6425095a13227f245cc`. Rebuilding or reprovisioning can change that value; validate the current flash manifest before programming. This is a private local image and no public binary release is implied.

The complete 16 MiB image was programmed and verified; the receipt is `artifacts/flash-backups/20261116-152654-544342/receipt.json`. With Windows NFS stopped, FPGA reconfiguration reached the AMD BRAM loader and U-Boot. JFFS2 `fsload` read the 2,527,086-byte uImage and 20,552-byte DTB; Linux mounted `31:4` writable, reached BusyBox and negotiated 100 Mbps full duplex. Current HTTP/MQTT, strict-key SSH and desktop/mobile browser checks passed, including the white background and linked IronKernel logo. This test did not remove board power.

Recorded baseline observations

These observations belong to earlier development builds, including the seven-partition SPI baseline. They are individual checks, not guarantees for the current image. No new run date is assigned from the document date; retain the matching logs/captures when issuing a test report.

Check	Recorded result	Scope
NFS boot	U-Boot/Linux/BusyBox, shell, root write/read	Existing Windows host
SPI root/services	Writable JFFS2, HTTP/MQTT/SSH, persistence over FPGA reconfiguration	Earlier seven-partition layout, NFS stopped
Ethernet	100 Mbps full duplex	100 MHz MicroBlaze/EthernetLite
iperf2	7.03 Mb/s over ten seconds	Baseline transfer
HTTP API	Approximately 0.2..1.0 seconds	Recorded checks
Commands inside one SSH session	`uname -r` 16 ms; `ls /` 99 ms; uptime 65 ms	Excludes connection startup
First observed no-fork ash applets	`ls` 170 ms; `uname` 32 ms; `cat` 66 ms	Prior observations 3.837 s, 5.455 s, 3.936 s
Explicit BusyBox `ls`	About 3.9 s cold, 113 ms warm	Cold execution cost remains
256 KiB SPI read	0.81 s polling vs 6.93 s interrupt completions	Individual startup reads, not sustained benchmark
Concurrent flash/network and idle	Four pings 1..2 ms; HTTP 68 ms; status 245 ms	Three-minute idle with NFS stopped
MQTT	Samples and five-to-two-second control and back	Onboard broker, Windows broker stopped/removed

Check	Recorded result	Scope
HTTP index	Matching root/index/query; HEAD/CGI errors	Patched BusyBox baseline

Current acceptance work

Current programming/verification, JFFS2 kernel/DTB loads, writable root, HTTP/MQTT, strict-key SSH and desktop/mobile dashboard checks have matching evidence above. Retain the exact image hash, operation receipt and serial/runtime records together. The NFS regression also passed: `run netboot` from SPI-loaded U-Boot fetched the same kernel/DTB and reached BusyBox on a writable NFS root (`logs/unified-netboot-serial.log` and capture JSON). HTTP/onboard MQTT status passed (`logs/unified-nfs-http-status.json`); runtime mount, hardlink and export write/read/delete checks are in `logs/flash-console.log`. Linux was cleanly halted before the owned Windows NFS server was stopped. The final SPI boot, with NFS stopped, verified that saved environment marker `ironforge_unified_proof=jffs2_20261116` and `/etc/ironforge-unified-proof` survived FPGA reconfiguration (`logs/unified-final-spi-persistence.log`, `logs/unified-final-verification.json`). U-Boot again loaded both `/boot` files, including a DTB rewritten with identical bytes by Linux; its SHA-256 `fc36bea1c35caa689455cba29f8fb9b7fba0da42b02a3a2bc26ea133685c619d` matched the packaged artifact. The five MTD partitions, writable JFFS2 root and BusyBox-ready marker were confirmed again. Remaining application checks need their own current logs. No new iperf2 result is claimed. Actual power removal/restoration is a separate required test.

The current NFS fetch/root regression is recorded above. Exercise flash reads concurrently with network bursts for lost interrupts, then idle/services/logs/free space. Five reset/boot cycles, 10 Mbps and standalone initramfs acceptance remain open unless separately recorded. A fixture-test pass does not close hardware acceptance.

Figure 19-1. Current five-partition SPI serial boot, selected exact lines with omissions marked; FPGA reconfiguration with Windows NFS stopped.

COM5 | Arty A7-100T | 115200 baud | recorded SPI boot

```
SREC SPI Bootloader
[... intervening recorded output omitted ...]
Loading SREC image from flash @ address: 300000
[... intervening recorded output omitted ...]
U-Boot 2026.10 (Nov 16 2026 - 15:21:43 +0000)
[... intervening recorded output omitted ...]
Loading Environment from SPIFlash... OK
[... intervening recorded output omitted ...]
### JFFS2 loading '/boot/uImage' to 0x82000000
[... intervening recorded output omitted ...]
### JFFS2 load complete: 2527086 bytes loaded to 0x82000000
### JFFS2 loading '/boot/system.dtb' to 0x84000000
[... intervening recorded output omitted ...]
### JFFS2 load complete: 20552 bytes loaded to 0x84000000
[... intervening recorded output omitted ...]
Creating 5 MTD partitions on "spi0.0":
[... intervening recorded output omitted ...]
VFS: Mounted root (jffs2 filesystem) on device 31:4.
[... intervening recorded output omitted ...]
xilinx_emaclite 40c00000.ethernet eth0: Link is Up - 100Mbps/Full - flow
control off
[... intervening recorded output omitted ...]
IRONFORGE_BUSYBOX_READY
[... intervening recorded output omitted ...]
BusyBox v1.38.0 (2026-11-16 02:29:43 UTC) built-in shell (ash)
[... intervening recorded output omitted ...]
#
```

Selected exact recorded serial lines rendered for print; omissions marked [...].

20. Debugging, Limitations and Licensing

Symptom	Inspect / action
AMD not found	Explicit release root, classic toolchain files, doctor logs
MSYS bootstrap incomplete	Installer/bootstrap error and explicit recovery instruction
Archive rejected	Lock/hash and download integrity; never bypass SHA-256
Patch conflict	Preserve edits, extract locked clean source, reapply stack
Configuration differs on another PC	Preserve/promote final `.config`, compare upstream defaults/tools
Stale package	Rebuild named component/dependencies, inspect receipts
Wrong/no JTAG target	Cable serial, dedicated port, exclude unrelated boards
DDR/PHY diagnostic failure	Stop loading; inspect calibration, clocks/reset, power, MDIO
Silent/busy serial	COM, 115200 8N1, DTR/RTS off, exclusive owner
NFS won't start	Occupied 111/1058/2049 ports and exact executable
DHCP/NFS stalls	LAN, DHCP, serverip, firewall, export terminal, boot paths
Stale NFS handles	Halt, stop export, install, restart export and boot
DTB relocation warning	Clear `fdt_high`, use current boot environment
SPI `fsload` failure	Partition selection, JFFS2 `/boot`, MTD patch/configuration
Defaults ignored	Inspect saved SPI environment
Slow first HTTP/SSH command	Compare cold/warm storage/execution; collect timed evidence
No MQTT samples	Onboard broker, loopback settings, broker log
SSH identity conflict	Confirm intended board/key, retain strict checks

Symptom	Inspect / action
Lost history	RAM-backed `/tmp`; export CSV before shutdown
`reboot` halts	Expected; clean halt then FPGA reconfigure or power cycle

Retain hardware logs and reports, component build/patch logs, core/application receipts, package/flash manifests, operation receipts and serial output. `local/environment.json` identifies resolved tools. Collect `/tmp` runtime evidence before restarting. Change platform/generator rather than hand-editing generated DTS into disagreement. Board-state records are invalid after external programming or power loss.

SPI is intentionally 6.25 MHz. EthernetLite/100 MHz CPU limit throughput below link rate. JFFS2 scanning/cold applications take time. Finite flash includes kernel, DTB, userspace and writable data; avoid uncontrolled persistent logs. No endurance or power-loss-safe update qualification is claimed.

HTTP/MQTT are unauthenticated. Key-only SSH does not make other services private. There is no TLS web server, authenticated MQTT, verified boot or generic credential-free release image. Private provisioning is included in local flash images/backups; protect those artifacts as credentials. Physical cold startup and complete fresh-PC installation/build remain open.

IronForge's project license has not yet been selected. Third-party projects retain upstream licenses/notices and applicable copyleft obligations. AMD tools, vendor IP and generated loader/BSP remain subject to AMD terms/licenses. Consult each actual downloaded license and selected AMD installation before redistribution. Availability of an archive is not redistribution permission. This specification does not grant rights to third-party code.

IronForge Specification

Contact IronKernel

Adapt the IronForge hardware and software workflow to your board. IronKernel can review your design, integrate AMD FPGA IP, bring up U-Boot and Linux, and develop custom device drivers or bare-metal firmware.

To discuss integration, reference designs or custom development, send your target board and project goal.

https://ironkernel.com/ · support@ironkernel.com