IronForge on Arty A7-100T

AI-guided FPGA Linux development on native Windows, with standalone flash boot, network boot and onboard Linux services.

IronForge brings FPGA hardware, boot software, Linux and applications into one Windows workspace. PowerShell commands and private MSYS helpers drive installed native toolchains. Linux runs on the board; the build workflow needs no WSL, Linux VM or container.

Codex or Claude can follow the tracked command workflow to acquire sources, configure components, build, validate receipts, package images and diagnose the target. The same commands are available interactively from PowerShell.

At a glance	Arty profile
FPGA	Digilent Arty A7-100T · `xc7a100tcsg324-1`
Processor / memory	MMU MicroBlaze · 100 MHz · 256 MiB DDR3
Host tools	Native Windows PowerShell + MSYS helpers; AMD Vivado/Vitis 2026.1
Common software	Linux 7.3.4 · U-Boot 2026.10 · BusyBox 1.38.0
Network	AMD EthernetLite · 100 Mbps link capability
Boot choices	Flash-to-RAM root + JFFS2 data; whole-root NFS; explicit JFFS2 root
Applications	Dashboard/API · SQLite · MQTT · iperf2 · C telemetry · games
Console	UART at 115200 baud, 8 data bits, no parity, 1 stop bit, no flow control

Standalone boot. The default loads a separate Linux kernel, device tree and compressed initramfs from SPI flash. Linux runs from writable RAM with 1 MiB of persistent JFFS2 storage at `/data`. Measured startup from the loader banner to a shell is about 47 seconds, excluding FPGA configuration.

One workspace, shared components

IronForge acquires immutable version/URL pins and verifies SHA-256 before using downloads. A component has one archive and one canonical extracted source tree, shared by every board. A Git-pinned component instead has one canonical checkout, a full commit pin and a tracked-input hash manifest. Compilers consume original sources directly.

Location	Maintained role
`apps/<component>/`	Shared application recipes, patches and owned assets; downloaded sources under its `sources/`
`system/<component>/`	Base programs, libraries, filesystem tools and the original common BRAM loader
`linux-7.3.4/`, `u-boot-2026.10/`	One shared tree for each common core release
`rtl/`	Complete downloaded RTL packages, including their drivers/support, plus authored HDL and benches
`platforms/<board>/`	All board policy: configuration, constraints, scripts, overlays, diagnostics and patches
`build/<platform>/`, `artifacts/`	Isolated compiled outputs and packaged images
`rootfs/`	One generated runtime filesystem assembled for the selected board

Each board builds from the same patched sources. Board configuration and the device tree select the peripherals and drivers. The compiler writes objects, generated headers and sysroots into the selected output directory.

Arty device-tree templates are maintained in `platforms/arty-a7-100t/dts/`. Generated DTS/DTB files are ignored beneath its `generated/` directory. Packages bind the templates and check generated DTB content; no top-level `dts/` is used.

From source to a demonstrable system

1. Acquire checked sources and prepare the canonical trees under the shared lock.
2. Select the platform and configure Linux, U-Boot, BusyBox and applications. Terminal menuconfig and optional VS Code actions expose the configuration workflow.
3. Compile hardware, diagnostic firmware, core software and applications with native Windows toolchains.
4. Validate real build receipts against current inputs, then assemble the single `rootfs/` and package target images.
5. Identify the exact board, load the selected artifacts and establish acceptance with fresh console and network records.

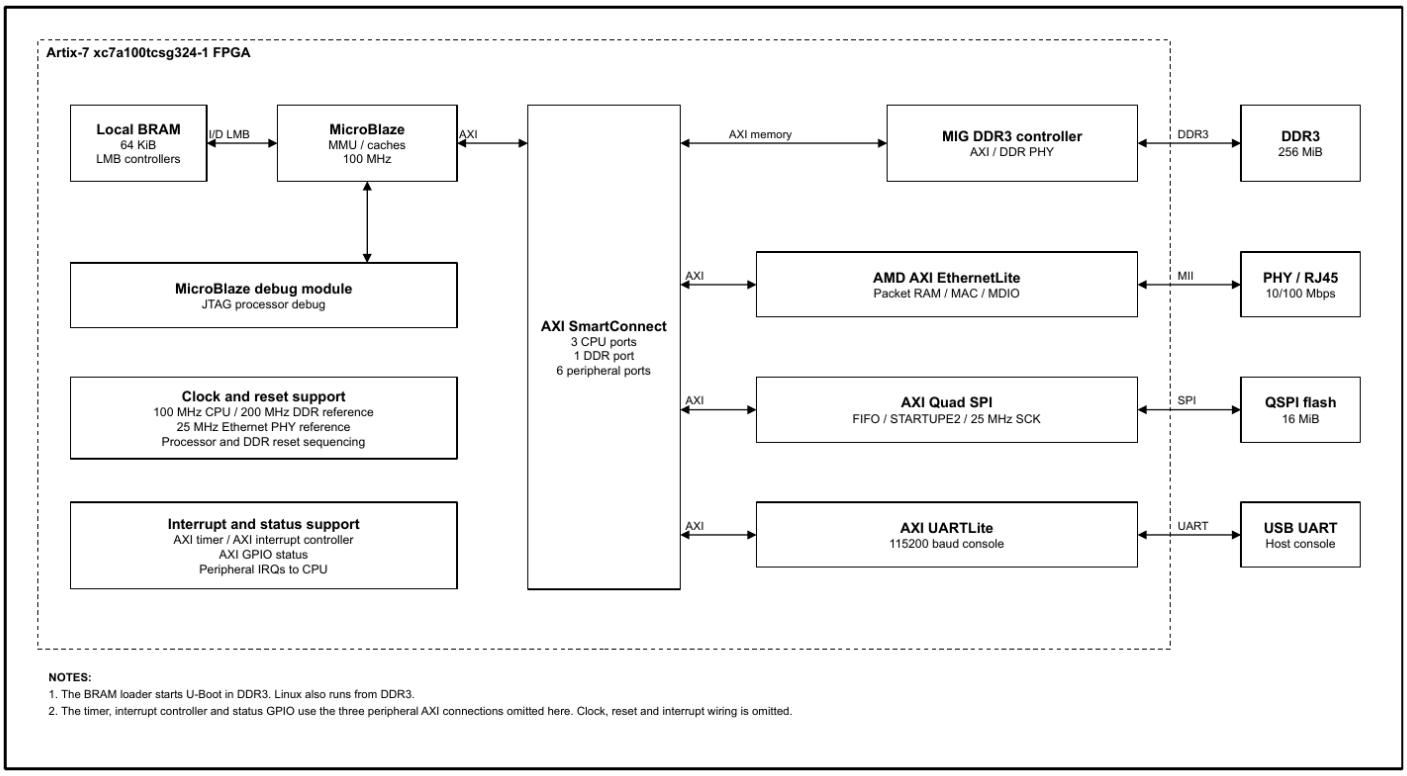
Shared native toolchains

Arty uses the native MicroBlaze toolchain. SAM9X60 Curiosity **EV40E67A** and STM32MP157F-DK2 share the native Windows ARM32 distribution. i.MX8M Mini EVK uses the ARM64 distribution. These profiles share the same Linux and U-Boot releases and application components.

The Windows NFS server exports the generated runtime filesystem. An NFS-booted board needs that export throughout its session.

Hardware and system architecture

Figure 1. FPGA hardware architecture (architecture, sheet 1).



The Arty FPGA design runs SPI at **25 MHz**, from the 100 MHz reference with divider four. MMU MicroBlaze executes Linux from DDR3 through the AXI system fabric. BRAM holds startup and diagnostic code. UART, timer, interrupt and SPI peripherals support Linux, and AMD EthernetLite provides Ethernet.

The original common **MIT-licensed BRAM SREC loader** in `system/loader/` reads the checked U-Boot S-record payload from flash. Its SPI reader, U-Boot and Linux use FIFO streaming with continuous chip select through command/address/data phases. Existing SPI-NOR page-program and erase handling remains in the driver stack. JFFS2 uses those drivers without a filesystem-specific transfer workaround.

The routed design meets setup, hold, pulse-width and external flash I/O timing at 25 MHz. The MISO input register captures on the falling edge. The BRAM diagnostic checks two patterns across the entire **256 MiB DDR3** and identifies the Ethernet PHY at address 1.

Boot paths and persistent data

Path	Kernel / device tree	Linux root and dependency
Default flash-to-RAM	Separate legacy `uImage`, external DTB and compressed cpio in raw flash	Writable RAM root; 1 MiB JFFS2 `/data`; no Windows server
Windows NFS	Separate `uImage` + `system.dtb` via TFTP from `rootfs/boot`	Entire root on NFSv3/TCP; no initramfs; server stays running
Explicit `jffs2-root`	Kernel + DTB FIT in `/boot/boot.itb`	Writable flash root using the selected-mode image

In Master SPI mode, the FPGA configures from flash. Its block RAM loader starts U-Boot, which verifies payload lengths and CRC32 values before starting Linux. Only `/data` persists. Volatile JTAG bring-up uses the current package's explicit `artifacts/boot.commands`. There is one generated rootfs and one active build set.

Default 16 MiB SPI layout

The default layout separates boot payloads from the persistent data partition.

Partition	Offset	Reserved capacity	Linux access
FPGA	`0x000000`	`0x300000` / 3 MiB	Read only
U-Boot	`0x300000`	`0x1c0000` / 1.75 MiB	Read only
Environment / redundant	`0x4c0000` / `0x4d0000`	64 KiB each	Read/write
Kernel	`0x4e0000`	`0x300000` / 3 MiB	Read only
DTB	`0x7e0000`	`0x010000` / 64 KiB	Read only
Compressed initramfs	`0x7f0000`	`0x710000` / 7.0625 MiB	Read only
JFFS2 data	`0xf00000`	`0x100000` / 1 MiB	Read/write

Measured RAM-root boot

A warm flash boot reached a shell in **46.5 s** from the loader banner, excluding FPGA configuration. Cold power-on timing has not been measured. U-Boot spent 2.8 s reading the kernel, 28 ms reading the DTB and 7.5 s reading the compressed initramfs. It loads directly from those raw partitions. Linux `reboot` halts on Arty. To restart, sync and halt Linux, then reconfigure the FPGA.

Applications and useful demonstrations

The dashboard is a Windows-browser view of real services running on the FPGA Linux target. An original C telemetry service reads Linux counters, writes a RAM-backed SQLite database and publishes to the Mosquitto broker on the board. Windows needs only the appropriate client for each demonstration.

Component	Demonstration
HTTP dashboard	Desktop/mobile telemetry for CPU, memory, network activity and uptime
JSON API / CSV	`/cgi-bin/status`, `/cgi-bin/history`, `/cgi-bin/versions`; history export through `/cgi-bin/csv`
SQLite	Latest 3,600 samples; about five hours at five-second sampling; RAM history disappears at reboot
Mosquitto 2.0.23	Onboard broker, publishers/subscribers, telemetry and interval control with acknowledgements
iperf2	Target TCP server and Windows peer helpers for measured application throughput
Original C demo	System sampling, local storage and loopback MQTT publication
nInvaders	ASCII terminal game over the USB UART
Vector Defense / Rocks	Original MIT games; shared rules and renderer, with bounded headless PPM capture

The target clock is unsynchronized. Uptime supports ordering and graph axes; wall-clock labels must retain that limitation. Export CSV before reboot to keep history. The LAN MQTT demo uses port 1883 without TLS/authentication; its configuration is for a trusted local demonstration network.

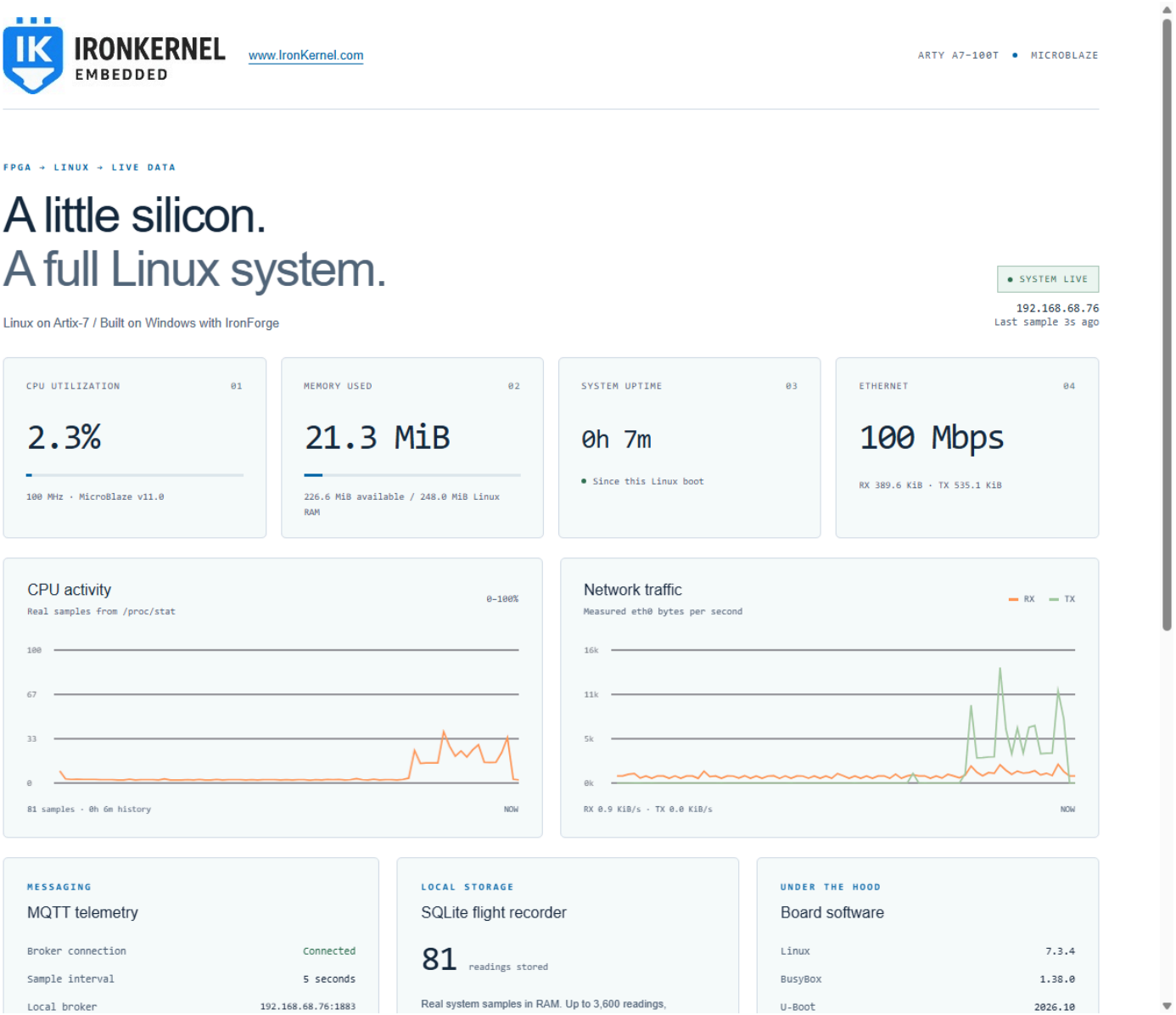
Inspect and control from Windows

Open the board’s DHCP address in a browser. The command helpers expose the same services.

```
.\ironforge.ps1 --platform arty-a7-100t demo status --address <board-address>
.\ironforge.ps1 --platform arty-a7-100t demo mqtt --address <board-address>
.\ironforge.ps1 --platform arty-a7-100t demo iperf --address <board-address> --seconds 3
```

The dashboard, API, CSV export and MQTT work from the RAM-root and NFS systems. A three-second TCP iperf run measured **5.35 Mbit/s** overall.

Live dashboard on the Arty



The Arty serves this page itself: its startup script mounts the runtime filesystems and starts BusyBox `httpd` on port 80 with `/www` as the document root. HTML, CSS, JavaScript and the logo come from the board. Browser JavaScript requests the status and history CGI endpoints about every five seconds; small shell wrappers invoke the static native C `ironforge-demo` executable to return JSON or CSV. Its background daemon samples Linux counters, keeps the latest 3,600 readings in SQLite under `/tmp/ironforge` and publishes telemetry to the local Mosquitto broker. No Windows web server is involved.

Browser capture of the dashboard served by the physical Arty. Uptime provides the time axis while the board clock is unsynchronized.

Build, boot and play

Download the complete 16 MiB flash image or the ZIP with programming instructions. Fit JP1 for SPI boot and open the USB serial port at 115200 baud.

From PowerShell at the repository root, after checked acquisition/configuration:

```
.\ironforge.ps1 --platform arty-a7-100t --boot-variant ram-root software all --jobs 2
.\ironforge.ps1 --platform arty-a7-100t package
.\ironforge.ps1 --platform arty-a7-100t diagnostic
```

Select the exact cable and console in local settings. `board identify`, `board program`, `board memtest` and `board boot` provide the controlled volatile bring-up sequence.

After RAM loading, interrupt U-Boot and set the RAM-root boot arguments:

```
setenv bootargs console=ttyUL0,115200 earlycon rdinit=/init loglevel=7 ip=off ironforge.boot=ram-root
```

Then paste the **current** `artifacts/boot.commands`; it supplies the exact packaged initramfs length and relocation settings. Wait for `IRONFORGE_BUSYBOX_READY` and the target's DHCP address. To use writable flash root, select `--boot-variant jffs2-root`, rebuild U-Boot and Linux, and repackage. Variant switching reuses the same sources and output directories.

ASCII game over UART

Connect an ANSI-capable terminal to the USB serial port at 115200 baud.

```
ironforge-game
```

Press **Space** to start or fire, the **arrow keys** to move, lowercase **p** to pause and lowercase **q** to return to the shell. The launcher starts nInvaders in an 80 × 30 Linux-terminal PTY. nInvaders 0.1.1 and its launcher retain their GPL-2.0-or-later notices. ncurses is shared under `system/ncurses/`.

Original vector games

Vector Defense protects six cities with aimed interceptors; Vector Rocks uses rotation, thrust and shots against splitting rocks. Both are original MIT-licensed code and art, with no SDL or GPU dependency. They share controls and the same component implementation across all platforms.

On Arty, both games run headlessly and output 640 × 480 PPM frames. A run on the NFS-root system produced identical captures to the Windows renderer using matching seeds and inputs. Use nInvaders for interactive play on the Arty terminal.

Deployment and operating requirements

U-Boot keeps a valid Ethernet address across restarts. On first boot without one, it generates a locally administered unicast MAC and saves it in SPI flash. Linux uses that same address for DHCP.

In the November 20 board test, Linux mounted a writable NFSv3/TCP root after fetching only the kernel and device tree. On the 100 Mb/s full-duplex link, six-second TCP tests measured 5.93 Mb/s from the PC to Linux and 6.20 Mb/s back through EthernetLite. The standalone RTLNet Ethernet image measured 94.9 Mb/s TCP in both directions and returned all 400 UDP echoes.

Item	Requirement
Board	Digilent Arty **A7-100T** with USB JTAG/UART
Windows build host	AMD 2026.1 device support, native MicroBlaze tools and the applicable vendor licenses
Local console	ANSI terminal, 115200 baud, 8N1, no flow control
Dashboard and remote demos	Ethernet connection to a reachable LAN with DHCP
Network boot	Windows TFTP for the kernel and DTB, plus an active NFSv3/TCP export for the running root filesystem
Standalone flash boot	FPGA image and boot payloads in SPI flash. Linux runs from RAM, with persistent files stored under `/data`.

Choose standalone flash boot for operation without a network server. Choose NFS root to iterate on applications and configuration from Windows without a flash update. Use the serial console for setup and diagnostics in either mode.

Arty ASCII gameplay

The screenshot shows nInvaders running on the Arty's serial terminal.



The Windows terminal renders the game's ANSI output from the board's UART. Space starts play and fires, arrow keys move the ship, and lowercase `q` returns to the shell.

Third-party components retain their upstream licenses. The common loader and original vector games are MIT-licensed.

Flash boot loader and payloads

```
Arty A7 | SPI loader and raw flash reads
Real COM5 capture | 115200 8N1 | rendered on Windows

IronForge SPI SREC
Starting U-Boot


U-Boot 2026.10 (Nov 17 2026 - 04:55:47 +0000)

Model: IronForge Arty A7-100T
DRAM:  256 MiB
Core:  8 devices, 8 uclasses, devicetree: embed
Loading Environment from SPIFlash...
SF: Detected s25fl128s with page size 256 Bytes, erase size 64 KiB, total 16 MiB
Loading Environment from SPIFlash... OK
In:    serial
Out:   serial
Err:   serial
Model: IronForge Arty A7-100T
Net:   EMACLITE: 40c00000, phyaddr 1, 1/1
eth0: ethernet@40c00000

Hit any key to stop autoboot: 3
Hit any key to stop autoboot: 2
Hit any key to stop autoboot: 1
Hit any key to stop autoboot: 0
IRONFORGE_RAW_BOOT_BEGIN
SF: Detected s25fl128s with page size 256 Bytes, erase size 64 KiB, total 16 MiB
IRONFORGE_RAW_KERNEL_BEGIN bytes=26841a crc32=7b757ae9
device 0 offset 0x4e0000, size 0x26841a
SF: 2524186 bytes @ 0x4e0000 Read: OK

time: 2.807 seconds
IRONFORGE_RAW_KERNEL_END
IRONFORGE_RAW_DTB_BEGIN bytes=5136 crc32=aeb608d7
device 0 offset 0x7e0000, size 0x5136
SF: 20790 bytes @ 0x7e0000 Read: OK

time: 0.028 seconds
IRONFORGE_RAW_DTB_END
IRONFORGE_RAW_INITRAMFS_BEGIN bytes=672a23 crc32=2054e6df
device 0 offset 0x7f0000, size 0x672a23
SF: 6760995 bytes @ 0x7f0000 Read: OK

time: 7.510 seconds
IRONFORGE_RAW_INITRAMFS_END
IRONFORGE_RAW_BOOTM
```

UART transcript from the SPI loader banner through `IRONFORGE_RAW_BOOTM`. U-Boot reads the kernel, device tree and compressed initramfs from separate raw partitions and verifies each payload's length and CRC32 before starting Linux.

Flash boot from RAM root to shell

```
Arty A7 | Linux initramfs to BusyBox shell
Real COM5 capture | 115200 8N1 | rendered on Windows

Freeing initrd memory: 6600K
Freeing unused kernel image (initmem) memory: 172K
This architecture does not have kernel memory protection.
Run /init as init process
  with arguments:
    /init
  with environment:
    HOME=/
    TERM=linux

init started: BusyBox v1.38.0 (2026-11-17 04:00:12 UTC)

starting pid 25, tty '': '/etc/rcS'
IRONFORGE_DATA_MOUNTED
IRONFORGE_DATA_READY bytes=1048576 filesystem=jffs2
IRONFORGE_USERSPACE_BEGIN
xilinx_emaclite 40c00000.ethernet eth0: Link is Down
udhcpc: started, v1.38.0
udhcpc: broadcasting discover
IRONFORGE_BUSYBOX_READY

starting pid 68, tty '/dev/ttyUL0': '/bin/sh'


BusyBox v1.38.0 (2026-11-17 04:00:12 UTC) built-in shell (ash)

#
```

UART transcript from Linux freeing the initrd buffer through the first BusyBox shell prompt. Linux uses a writable RAM root, mounts the 1 MiB JFFS2 partition at `/data` and starts Ethernet and the onboard services.

Contact IronKernel

Bring up an Artix-7 MicroBlaze design with IronKernel. We can review your FPGA design, integrate IP, and adapt the U-Boot and Linux flash boot path, device tree or peripheral drivers.

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

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