DOCUMENT
IDB-FWA-020

FIRMWARE · OTA · MTM · SECURITY · REFERENCE DOCUMENT · PUBLIC

Firmware architecture and manufacturing test

Reference for embedded firmware architecture, OTA update strategy, security baseline, and the manufacturing test mode (MTM) that production lines use to verify every unit.

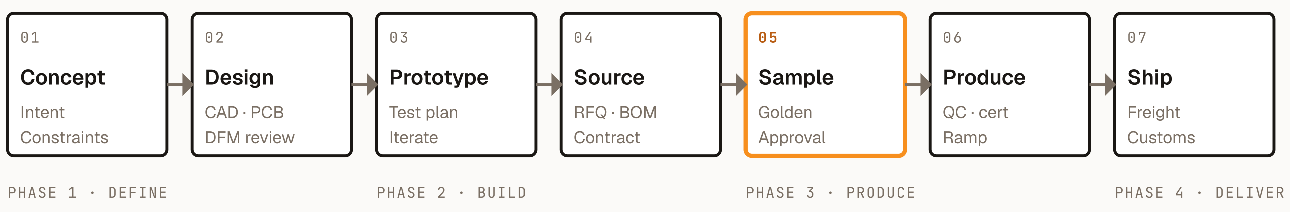
REVISION	ISSUED	OWNER	COMPANION
1.0	May 2026	Ideambox engineering	PDF reference

ABSTRACT

Firmware is the part of the product that updates after launch. Architectural decisions made early determine whether the team can patch a critical bug in week 1 of production, push an OTA security fix in year 3, or ship a manufacturing variant for a new market without re-spinning hardware.

Section 1 covers architecture patterns (HAL, RTOS, bare-metal). Section 2 covers Manufacturing Test Mode (MTM) — the firmware mode that the factory uses to verify every unit. Section 3 covers OTA update strategy. Section 4 covers security baseline. Section 5 covers version control and release process.

HARDWARE PRODUCT DEVELOPMENT — 7-STAGE PIPELINE



FIRMWARE DECISIONS AFFECT EVERY LATER PHASE — PRODUCTION TEST, CERTIFICATION, FIELD UPDATES, AND SECURITY RESPONSE.

CONTENTS

1. Firmware architecture
2. Manufacturing Test Mode (MTM)
3. OTA updates
4. Security baseline
5. Version control + release

1. Firmware architecture

Pick architecture that matches product complexity. Over-engineering wastes ROM; under-engineering creates maintenance debt.

1.1 Architecture patterns

PATTERN	USE	COMPLEXITY
Bare-metal (super-loop)	Simple sensors, no real-time constraints	Lowest
State machine	UI-driven products, defined modes	Low
Cooperative scheduler (event loop)	Connected products with periodic tasks	Medium
Preemptive RTOS (FreeRTOS, Zephyr, ThreadX)	Multi-tasking, real-time deadlines	Medium-high
Embedded Linux	High-resource SoCs (Raspberry Pi, Yocto)	High
Hypervisor + dual-OS	Safety-critical or multi-domain (auto, medical)	Highest

1.2 Hardware Abstraction Layer (HAL)

A HAL separates application code from hardware specifics. Critical for:

- **Supporting multiple MCU variants**
  Same code, different chip.
- **Porting to new platforms**
  Swap drivers, keep application.
- **Mock hardware for unit testing**
  Run application code on PC.
- **Survivability across silicon obsolescence**
  Hardware changes don't ripple through application.

1.3 Memory budget

MCU CLASS	FLASH	RAM	USE
Low-end (Cortex-M0, M0+)	32–128 KB	4–16 KB	Simple sensors, BLE peripherals
Mid-range (Cortex-M3, M4)	128–512 KB	32–128 KB	Most consumer IoT, BLE central
High-end (Cortex-M4F, M7, M33)	512 KB – 2 MB	128–512 KB	Wi-Fi devices, displays, audio
Application processor (Cortex-A)	External	External	Linux-class products, smartphones

Reserve ~25–35 % of flash for OTA updates (need to store new image while running old).

1.4 Boot architecture

- **Single-image boot**
  One application image; OTA replaces in-place. Risk: power-loss during update bricks device.
- **Dual-image boot (A/B)**
  Two slots, switch on successful update. **Recommended** — survives power loss.
- **Bootloader-based**
  Small bootloader manages app load. Standard for production.

1.5 Recommended bootloader features

- **Application image validation**
  CRC + signature verification before boot.
- **Failover**
  If new image fails, boot from previous slot.
- **Recovery mode**
  Pin-toggle or magic sequence enters bootloader for emergency reflash.
- **Verbose error reporting**
  Boot reason, last-flash status, watchdog counter accessible via UART.

2. Manufacturing Test Mode (MTM)

The firmware mode that the production line uses to verify every unit before it ships. The single most-important production-readiness firmware feature.

2.1 What MTM does

- **Hardware self-test**
  Exercises every accessible peripheral.
- **Calibration**
  Stores per-unit cal values (sensor offsets, RF tuning) in non-volatile memory.
- **Identity programming**
  Writes unique serial number, MAC address, regional config.
- **Quality measurement**
  Captures and logs pass/fail data per station.
- **Lockout after production**
  MTM disabled in shipping firmware; access requires JTAG or special command.

2.2 MTM test sequence (typical)

STAGE	TEST	METHOD	PASS CRITERIA
1	Boot + ID	Read chip ID, verify silicon	Match expected
2	Power rails	ADC measurements	3.3 V ±5 %, 1.8 V ±5 %
3	LED test	Sequentially light each LED	Operator visual
4	Button test	User presses each button	All registered within 30 s
5	I/O continuity	Output high/low, read input	Match expected
6	Comms peripherals	Loopback or external test	UART, SPI, I2C connect
7	Wireless RF	Transmit beacon, RSSI measure	RSSI in expected range
8	Sensor read	Take sample under known conditions	Within tolerance
9	Battery check	If powered by battery	Voltage in range
10	Final pass	All tests passed	Write production timestamp
11	Calibration	Per-unit cal	Sensor offset stored
12	Identity write	Serial, MAC, region	One-time programming (OTP)
13	MTM lockout	Disable test mode	Burn fuse or set flag

2.3 MTM access methods

- **Special command via UART**
  Production line connects to UART, sends magic sequence.
- **Boot pin combination**
  Hold specific buttons during boot.
- **JTAG-only**
  Access via debugger; common for security-conscious products.
- **Magic byte in NVM**
  Set at first programming; cleared after MTM lockout.

2.4 MTM access in shipping firmware

- **Lockout after MTM complete**
  Production timestamp + lockout flag prevents re-entry.
- **Recovery requires JTAG**
  Service technicians use JTAG to unlock for repair.
- **Audit trail**
  Each MTM session logs to NVM (which station, when, results).

2.5 MTM design patterns

- **Modular tests**
  Each test is independent; can be re-run individually.
- **Pass/fail reporting**
  Each test outputs structured data (JSON, CSV) to UART for the production fixture to log.
- **Per-station scripts**
  The production fixture script knows which tests to run at which station.
- **Failure capture**
  On failure, record waveforms, ADC values, register state for forensics.

3. OTA updates

Over-The-Air updates are mandatory for connected products. Architectural decisions made early determine OTA success rate.

3.1 OTA update architecture

`` 1. Server ─────[notification]─────→ Device 2. Server ─────[image download]─────→ Device (chunks, resumable) 3. Device ─────[signature verify]───── 4. Device ─────[CRC verify]───── 5. Device ─────[stage in slot B]───── 6. Device ─────[reboot to slot B]───── 7. Device ─────[mark slot B as primary]───── 8. Server ─────[confirmation]───── ``

3.2 OTA design rules

– **Dual-slot storage**
Always. Single-slot OTA has too high failure rate.

– **Signature verification**
Cryptographic signature on every image. Reject unsigned or tampered images.

– **Resumable download**
Network drops are common; resume from last chunk.

– **Atomic switch**
New image becomes active only after full download + verify.

– **Rollback support**
If new image fails post-boot, automatically rollback to previous.

– **Confirmation**
Device reports success/failure back to OTA server.

3.3 OTA security

– **Image signing**
RSA-2048 or ECDSA P-256 signature on image hash.

– **Public key in device**
Burn during MTM; can't be changed by attacker.

– **Encrypted images (optional)**
For products with proprietary firmware.

– **Anti-rollback**
Track version; refuse to downgrade to known-vulnerable versions.

– **Server-side authentication**
Device proves identity before receiving image (mTLS or per-device tokens).

3.4 OTA frequency considerations

– **Mandatory updates**
Security patches; force install.

– **Recommended updates**
New features; prompt user.

– **Critical timing**
Don't update during user interaction (e.g., timer running, music playing).

– **Battery threshold**
Don't update below 30 % battery.

– **Network conditions**
Pause on metered connections; resume on Wi-Fi.

3.5 OTA scale considerations

- **Staged rollout**
  1 % → 5 % → 50 % → 100 % over 2 weeks. Catches failed updates before full deployment.
- **Telemetry feedback**
  Monitor success rate, post-update crashes, battery drain changes.
- **Server load**
  A million devices auto-checking weekly = significant traffic. Use CDN, randomised check times.

4. Security baseline

Embedded firmware security is increasingly mandatory (EU Cyber Resilience Act 2024, UK PSTI Act).

4.1 Security requirements (EU CRA, UK PSTI Act 2024)

REQUIREMENT	WHAT IT MEANS
Unique credentials	No shared default passwords across units
Software update mechanism	Receive security patches
Vulnerability disclosure policy	Way to report bugs
Security update timeline	Patch within reasonable period
Secure default config	Default settings minimise risk

4.2 Hardware security features

- **Secure Element (SE) or TPM**
  Stores cryptographic keys in tamper-resistant hardware.
- **Secure boot**
  Cryptographic chain of trust from boot ROM to application.
- **Read-out protection**
  Prevents reading firmware via JTAG.
- **Anti-rollback fuses**
  Permanent prevention of downgrade.
- **Trusted Execution Environment (TEE)**
  ARM TrustZone or similar; isolated execution.

4.3 Firmware security baseline

- **No hardcoded credentials**
  Per-device unique secrets, never shared across units.
- **Encrypted storage**
  Sensitive data (Wi-Fi credentials, tokens) encrypted at rest.
- **Encrypted communication**
  TLS 1.3 for cloud; encrypted BLE pairing.
- **Crypto library**
  Validated (FIPS 140-3, NIST-approved): mbedTLS, BoringSSL, wolfSSL.
- **Random number generation**
  Hardware RNG; never reuse RNG state.
- **Memory safety**
  Avoid C buffer overflows; consider Rust for new firmware.

4.4 Common firmware vulnerabilities (OWASP IoT Top 10)

1. Weak, guessable, hardcoded passwords 2. Insecure network services 3. Insecure ecosystem interfaces (cloud, mobile, web) 4. Lack of secure update mechanism 5. Use of insecure or outdated components 6. Insufficient privacy protection 7. Insecure data transfer and storage 8. Lack of device management 9. Insecure default settings 10. Lack of physical hardening

5. Version control + release

Firmware is software. Apply software-engineering discipline.

5.1 Repository structure

- **One repo per product family**
  Branches per variant if hardware varies.
- **CI/CD pipeline**
  Build, unit test, static analysis on every commit.
- **Build reproducibility**
  Same source → same binary (deterministic build flags).
- **Code review required**
  No direct commits to main branch.

5.2 Release tagging

- **Semantic versioning**
  `MAJOR.MINOR.PATCH` (e.g., `1.2.3`).
- **Tag on production release**
  Tag in git matches version flashed to production.
- **Release notes**
  Per release, document changes, fixes, known issues.
- **Reproducible build artifact**
  Hash matches firmware in production.

5.3 Release process

1. **Feature complete** — All planned features merged to develop branch. 2. **Code freeze** — No new features; bug fixes only. 3. **Regression testing** — Run full test suite on hardware. 4. **Release candidate (RC)** — Tag, push to internal testers. 5. **Field beta** — Limited group of customers. 6. **Production release** — Tag, push to production server, MTM uses this image.

5.4 Build flags

Different firmware images for different production tiers:

BUILD	PURPOSE	DIFFERENCES
Production	Shipping firmware	MTM disabled, debug stripped, optimised
Debug	Engineering	MTM enabled, debug symbols, less optimisation
Manufacturing	Factory only	MTM enabled, identity-write enabled
Engineering	Internal testing	Verbose logging, all debug

> FINAL NOTE. firmware is the part of the product that keeps evolving after launch. The architectural decisions made in week 1 — boot strategy, OTA design, MTM, security baseline — determine whether the team can ship a security patch in year 5, manufacture a new SKU variant, or recover from a production-line bug.