

9 MISTAKES YOU SHOULD NEVER SHIP

FIRMWARE PRE-SHIP CHECKLIST

Nine firmware mistakes that cause dead units, copied products, and failed certifications. And the fixes to make before you ship.



DEBUG BUILD
SHIPS UNPROTECTED

FIX →



RELEASE BUILD
LOCKED AND SIGNED

PREPARED BY
JOHN TEEL

DOC NO.
FW-09

REV
A

SOURCE
[PREDICTABLEDESIGNS.COM](https://predictabledesigns.com)

Firmware Pre-Ship Checklist

Catch these nine firmware mistakes before your production build ships.

The firmware mistakes in this checklist usually go unnoticed in a prototype, and none of them cause a problem until the product is already in customers' hands. One of them wears out the flash memory in a microcontroller in days instead of years, another lets anyone copy your entire firmware, and a third is now against the law in the UK and California and will block your CE mark in Europe.

Each entry explains why the mistake breaks down in production and gives you the specific checks to make before you sign off on your production build. The list follows the same countdown as the video, starting with a mistake that costs you a few milliseconds and ending with one that can cost you a fine. Several of the fixes are only a few lines of code.

The Checklist at a Glance

#	THE MISTAKE	THE FIX	WHAT IT COSTS YOU
9	Debug logging left on	Compile verbose logs out of release	Leaked internal data
8	Blocking delays	Timers, a state machine, or an RTOS	Missed button presses
7	Constant flash writes	Write on change only, or use FRAM	Worn-out memory
6	Watchdog never enabled	Enable it, reset from the main loop	Hung units and returns
5	Brown-out detection off	Keep it on and fix the power problem	Corrupted flash
4	Open debug port	Turn on readout protection	Cloned firmware
3	Hardcoded secrets	Per-device keys in protected memory	Every unit compromised
2	No secure update path	Signed images plus a fallback copy	Unfixable bugs
1	Shared default password	Unique per unit or set on first boot	Fines and no CE mark

Every fix above is spelled out in its full entry, so keep reading before you sign off on the build.

#9

Debug Logging Left in the Shipping Build

SERIAL LOGGING

WHY IT FAILS IN A PRODUCT

- Each print call has to format a string and push it out the serial port one character at a time, and with the default serial setup on most microcontrollers that stalls the rest of your main code while it runs.
- A single long line of text can tie up the processor for a couple of milliseconds, which is long enough to miss a sensor sample or throw off a timing loop, and the chip cannot drop into a low-power sleep mode while it is busy printing.
- Development logs tend to include internal state, memory addresses, and sometimes the actual credentials the device is using, and anyone with a five dollar USB to serial adapter can clip onto that UART pin and read it.

BEFORE YOU SHIP

THE CHECK	WHY IT MATTERS
Put logging behind levels	The verbose output is compiled out of the release build, and only warnings and errors make it into the product.
Audit every remaining log line	Nothing that reaches the UART should expose internal state, memory addresses, or credentials.
Keep a sane production log level	Some logging is useful for debugging units that come back from the field, so do not strip it out entirely.

A few lines of code

This is one of the cheapest fixes on the list. Put your logging behind levels and let the release build compile the verbose output away.

#8

Blocking Delays in the Main Loop

TIMING AND POWER

WHY IT FAILS IN A PRODUCT

- Without an operating system, a blocking delay means your main loop sits and spins doing nothing for that entire time, so it cannot check a button, service a sensor, or respond to a wireless packet.
- A quick button press that happens entirely during a 500 ms delay is never detected, and you may never catch it in your own testing, because your presses may just happen to land outside of that delay.
- It is also brutal for battery life, because a processor spinning in a delay loop draws full run current the whole time instead of sleeping.

BEFORE YOU SHIP

THE CHECK	WHY IT MATTERS
Move to a non-blocking design	A hardware timer or a simple state machine decides when things happen, and the processor sleeps in between.
Check the clock before acting	Even a basic loop that checks the clock before doing anything is a big step up from a blocking delay.
Use a small RTOS if the product is complex	A real-time operating system handles the scheduling for you once there are enough tasks to juggle.
Keep any delay to microseconds	A few microseconds of delay to satisfy a peripheral's timing requirement is fine, and that is the only place it belongs.

The bug you will never see on your bench

A press that lands inside a delay is not a rare event once thousands of customers are pushing the button, even if it never once happened while you were testing.

#7

Hammering Flash or EEPROM with Constant Writes

NON-VOLATILE MEMORY

WHY IT FAILS IN A PRODUCT

- A typical microcontroller's internal flash is rated for around 10,000 write and erase cycles per sector, and a dedicated EEPROM chip is usually rated somewhere around a million.
- Firmware that saves a setting every second burns through a million cycles in under two weeks, and it usually looks innocent, like saving the volume level or a run-time counter every time through the loop.
- Once a cell wears out, reads start returning garbage and settings reset themselves, and there is no fixing it in the field because the memory is physically damaged.

BEFORE YOU SHIP

THE CHECK	WHY IT MATTERS
Write only when a value changes	Then wait a few seconds before committing, so a customer spinning a knob does not generate a hundred writes.
Spread continuous logging across the memory	Do not write to the same spot every time, so no single section wears out early. Most chip vendors offer a storage library that handles this for you.
Use FRAM for heavy write loads	FRAM is a type of non-volatile memory that can handle practically unlimited writes.

Do the math

One write per second is 86,400 writes a day. Against a million-cycle budget, that is less than twelve days of use before the part starts failing.

#6

Watchdog Timer Never Enabled

FAULT RECOVERY

WHY IT FAILS IN A PRODUCT

- A watchdog is a simple countdown that resets the chip unless your firmware keeps resetting the counter, and it is built into just about every microcontroller on the market. Most products still ship with it turned off.
- Firmware hangs for all kinds of reasons that never show up in your own testing, like an I2C sensor holding the data line low or a rare buffer overflow that only happens once a week.
- Without a watchdog, a hung microcontroller stays hung until someone cycles the power. The customer sees a product that just stops working, and most of them will return it rather than power cycle it.

BEFORE YOU SHIP

THE CHECK	WHY IT MATTERS
Enable the watchdog timer	If your code hangs, the counter stops getting reset, the timer runs out, and the chip reboots itself.
Reset it from the main loop	A timer interrupt keeps firing even when the main loop is completely stuck, so a watchdog reset from an interrupt never trips and gains you nothing.
Confirm every critical task first	Reset the watchdog only after you have verified that each critical task is still running.

The mistake that hides inside the fix

Resetting the watchdog from a timer interrupt is the most common way to enable it and still get nothing from it, because the interrupt keeps running while the rest of the firmware is frozen.

#5

Brown-Out Detection Turned Off

POWER INTEGRITY

WHY IT FAILS IN A PRODUCT

- Brown-out detection is a circuit inside the microcontroller that watches the supply voltage and holds the chip in reset whenever the voltage drops below a safe level. The same kind of protection shows up in lots of other chips as an undervoltage lockout.
- A microcontroller with insufficient supply voltage does not just stop. It skips instructions and corrupts registers while it thinks everything is fine, and that can happen from a slow supply ramp at power-on just as easily as from a voltage dip.
- If a battery sags under load or a motor kicks on while the firmware is halfway through saving your settings, what ends up in flash is corrupted, and depending on what was being written, the unit becomes non-functional.

BEFORE YOU SHIP

THE CHECK	WHY IT MATTERS
Keep brown-out detection enabled	If the detector keeps tripping, it is catching a real power problem that needs fixing. Turning it off only masks that problem.
Set the threshold for flash writes	The threshold belongs above the minimum voltage the chip needs for reliable flash writes, not just the minimum it needs to run.
Add a supervisor for a tighter threshold	An external voltage supervisor chip is a good option, but never remove the built-in detector with nothing in its place.

A warning from the real world

A Hardware Academy member ran into a microcontroller that kept brown-out resetting, and the common reaction is to make the resets stop by turning off the detection. The advice he got was to be grateful the detector was tripping, because it was catching a real power problem that needed fixing.

#4

Debug Port Left Open with No Readout Protection

FIRMWARE PROTECTION

WHY IT FAILS IN A PRODUCT

- The debug port, usually SWD or JTAG, is how you program and debug the chip during development. Ship a product with that port wide open and anyone with a debugger can pull your entire firmware off the chip in seconds, and that is your whole product in a file.
- For a lot of products the hardware is the easier part to copy, especially if you are not using any custom designed chips, so your firmware is often the only thing standing between you and a knockoff.
- Most product creators never turn protection on, usually because nobody wants to lose the ability to debug a returned unit.

BEFORE YOU SHIP

THE CHECK	WHY IT MATTERS
Enable readout protection as the last step	Set it at the end of your production programming, after the firmware is loaded and tested, so you keep full debug access until the unit is finished.
Use your chip family's built-in setting	STM32 has readout protection levels, Nordic parts have an access port protection setting, and the ESP32 needs a permanent fuse that disables the debug port plus flash encryption to keep the firmware unreadable.
Pick a level you can still erase	The highest readout level on most STM32 parts can never be undone even by you, so choose the level that blocks readout but still allows a full erase for warranty repairs.

One setting

Almost every chip family has this fix built in, and it is usually one setting in your production programming sequence.

#3

Secrets Hardcoded in the Firmware Image

CREDENTIALS

WHY IT FAILS IN A PRODUCT

- Hardcoding private API keys, cloud secrets, or Wi-Fi credentials into the firmware is the fastest way to get a prototype talking to the cloud, and that is exactly why it ends up in production.
- A firmware image is just a file, and every plain text string inside it is sitting there in the open. Anyone who pulls the image off the chip, or downloads it from your update server, can run one command that lists every readable string in the file.
- Since the same key is baked into every unit, one compromised device compromises every device you have sold, and that is when somebody starts pushing fake data into your cloud, running up your API bill, or pulling other customers' data.

BEFORE YOU SHIP

THE CHECK	WHY IT MATTERS
Provision Wi-Fi through the customer	Network details come in through a provisioning step, where the customer enters them through your app or a temporary setup mode, so no Wi-Fi password ever lives in the firmware image.
Make cloud credentials unique per device	Program them in at the factory or generate them on first boot, and store them in a secure element or a protected region of memory rather than in the main image.
Strip your own office credentials	The Wi-Fi credentials that end up hardcoded are usually your own, left over from prototyping, and anyone who reads your firmware can get onto that network.

Check the image itself

Run that one command on your own release image before somebody else does. If a key or a password shows up in the list of strings, it ships with every unit.

#2

No Secure, Recoverable Firmware Update Path

FIELD UPDATES

WHY IT FAILS IN A PRODUCT

- Firmware bugs are about the only bugs you can fix after the product ships without getting units back from customers, but only if you built a way to do it. A surprising number of products ship with no update path at all, so the first serious bug becomes a permanent feature of every unit sold.
- The other version of this mistake is almost worse, an update path with no security on it, where the device accepts any firmware image somebody pushes to it. Anyone who figures out your update protocol can load their own code onto your customers' devices.
- An update that is not recoverable turns a failed download or a power loss halfway through into a non-functional unit.

BEFORE YOU SHIP

THE CHECK	WHY IT MATTERS
Sign every firmware image	The device has to check that signature before it runs any of the new code.
Keep two copies of the firmware in flash	The device runs the old copy until the new one is verified and then switches over, so a failed update never leaves the unit dead.
Budget flash for two copies	You need room for both images, either in the microcontroller's internal flash or in an external flash chip, so keep it in mind when you choose a microcontroller.
Set up the bootloader before the first unit ships	Most modern chip families come with a bootloader framework that handles this, but it has to be in place before the first unit leaves the factory.

#1

One Default Password Shared Across Every Unit

COMPLIANCE

WHY IT FAILS IN A PRODUCT

- Shipping every unit with the same default login, like admin and password, was standard practice for years, and manufacturers sometimes even printed it in the manual so customers could get into their settings.
- Attackers know those default logins too, and there are automated tools that scan the internet for devices still using them.
- This is the only mistake on the list that is now against the law in parts of the world.

BEFORE YOU SHIP

THE CHECK	WHY IT MATTERS
Generate a unique password per unit	Create it at the factory and print it on the label, or force the customer to set their own the first time they power the product on.
Check the UK rules	Since April 2024 the UK has banned universal default passwords on connected consumer products, with fines up to ten million pounds or 4% of global revenue.
Check the California rules	California has required unique or user-set passwords on connected devices that can be logged into remotely since 2020.
Check the European rules	The cybersecurity rules for internet-connected wireless products that took effect in August 2025 treat a shared default password as a failure of the standard, which in practice means no CE mark through the normal self-certification route.

Compliance warning

A shared default password is no longer just a security weakness. In the UK, California, and Europe it is now a legal or certification failure on top of one.

The Five Pre-Ship Filters

Every production build gets judged on the same five filters once it is inside a customer's product, and the firmware has to clear all of them. Run your release build through this list before you sign off.

- 1 Timing**
 Nothing in the main loop blocks, and no print call stalls the code that keeps the product responsive.
- 2 Memory**
 Every write to flash or EEPROM fits inside the part's rated write budget for the life of the product.
- 3 Recovery**
 The watchdog and brown-out detection are both enabled and configured so a hang or a voltage dip ends in a clean reset instead of a dead unit.
- 4 Protection**
 The debug port is locked, no secrets live in the image, and every update is signed and recoverable.
- 5 Compliance**
 Every unit ships with a unique password or forces the customer to set one, which is what the UK, California, and Europe now require.

Get a second set of eyes before the first unit ships.

Checks like your watchdog setup, your update path, and your readout protection are an everyday conversation inside the Hardware Academy, where you can get feedback on your specific product from me and lots of other experts.

PredictableDesigns.com/academy