Compression, Air, Fuel, Spark

Steve Spano
linkedin.com/in/sspano01

Or: The Four Things a Board Needs Before Any of the Clever Parts Matter

On resets, power, clocks, and code, why nearly every bring-up problem I have ever chased lived in one of them, and why you have to understand the basics before you can rule them out.

The engine that understood bring-up

A modern car is a preposterous machine. Under the hood there are miles of wire, dozens of sensors and switches, a few hundred pounds of reciprocating metal, and a network of controllers arguing with each other over CAN. It looks impossible to reason about, and for most people it is. Strip all of that away and an internal combustion engine needs exactly four things to run. It needs **compression**, **air**, **fuel**, and **spark**. Every mechanic who has ever diagnosed a car that won't start has walked those four in some order, because if any one of them is missing nothing else about the car matters. The heated seats are not the problem.

A circuit board looks the same way on the bench. There are a dozen layers, power planes and polygons, serial buses, PCI Express, a few lanes of high-speed Ethernet, and more power rails than anyone wants to count. It is easy to convince yourself that the failure is hiding in the exotic parts, and it almost never is. Every board I have ever brought up needed four things before the first LED lit or the first packet crossed a link, and those four are **reset**, **power**, **clock**, and **code**. I keep the engine in my head because it holds me to the order.

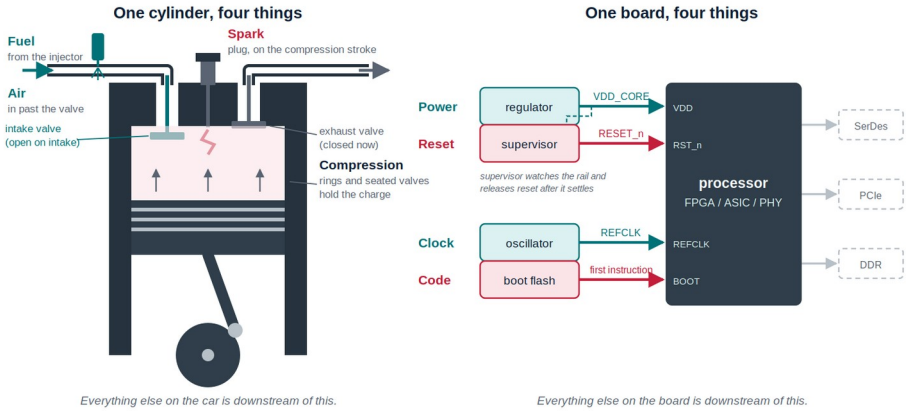


Figure 1 — One cylinder, one board. Air in past the intake valve, fuel from the injector, spark on the compression stroke, and rings and seated valves holding the charge, beside the regulator, supervisor, oscillator, and boot flash that feed a processor. The SerDes, PCIe, and DDR sit downstream of all of it on the board, the same way the heated seats sit downstream of the cylinder.

Know how the basics work, not just that they exist

Naming the four is the easy part. The part that saves the week is understanding how each one works well enough to know what **working** looks like, and how to prove it with an instrument. A mechanic who knows that compression is a seal made by rings and valves knows to reach for a compression gauge and knows what number to expect. A mechanic who only knows the word will replace the plugs. On a board, knowing that a supervisor has a threshold and a fixed timeout tells you what to look for on the reset line and how long the low pulse should be. Knowing that a PLL needs a reference inside its capture range before it will lock tells you that a clock which is present

but 300 ppm off is not present in any way that matters. Knowing that the stack grows down toward the vector table tells you what a board that dies after ten microseconds has probably done to itself.

No matter how complex the system is, it needs a small number of basic things to operate, and everything impressive about it is downstream of those. Start there. Understand how the basics work, understand how to confirm that they are working, and only then go chasing the corner case, because the corner case is very often a poor reset, and the car is very often just out of gas.

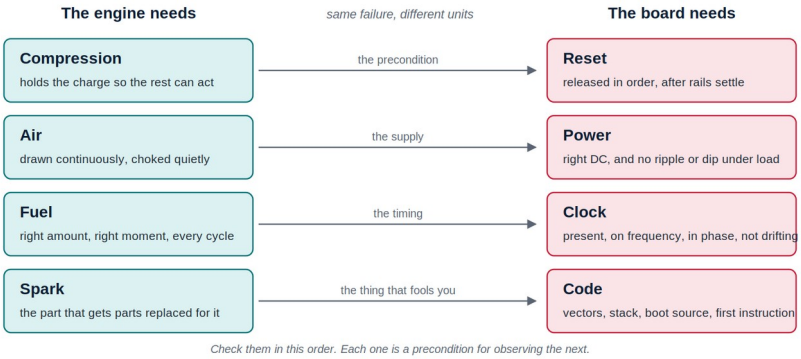


Figure 2 — The mapping. Compression, air, fuel, and spark on the engine side, reset, power, clock, and code on the board side. Each one is a precondition for observing the next, which is why the order matters.

The board that cranks and won't catch

The first article from the contract manufacturer is on the bench, the supply is current-limited, the switch flips, and nothing happens. There is no LED, no console banner, no link light, and the current draw is plausible enough to tell you nothing. A murkier problem is the board that does boot, prints a banner, and then is simply gone. Murkier still is the Ethernet link that trains, drops, trains, and drops, in a rhythm that looks like it means something. Every one of those is a car that cranks and won't catch, and the temptation is to go straight to the PCIe root complex or the SerDes eye, because those are the parts that feel like they could be wrong. Ninety-some percent of the time the fault is upstream of all of it, in one of the four fundamentals, and the fastest route to the exotic bug is to clear the four first.

Compression is reset

An engine with no compression can have perfect air, fuel, and spark and it will never run, because the cylinder cannot hold the charge long enough for the other three to do anything. Reset is the same precondition. Every processor, FPGA, PHY, and switch ASIC on the board sits in reset until something lets it out, and the **order** and **timing** of that release is the first thing to verify and the last thing most people look at. The reset tree on a real board is a circuit in its own right. A voltage supervisor watches the core rail, holds reset until the rail is above its threshold, and then waits a fixed delay before releasing. Downstream, a PHY wants its reset released only after its reference clock is stable, and a switch ASIC wants its reset held until the host has finished bringing up a bus.

If reset releases before the rails settle, the part comes out of reset in an undefined state, and the failure is intermittent in the worst way, because it depends on how the supply happened to ramp that morning. If reset never releases, because a strap resistor is stuffed to the wrong value, a debugger is quietly holding the line, or the supervisor threshold sits a hair above where the rail lands, the board is a brick that draws exactly the right current. Put a scope on every reset line and watch each one release, in order, with its rails already above threshold. Do the compression check first.

Air is power

Air is the thing the engine draws on continuously and the thing nobody thinks about until a filter clogs. Power is the same. A board that powers up at all has a good chance of having rails that read the right DC voltage on a meter, and that is where most people stop checking. The problems that bite live in the AC. **Ripple** on a rail feeding a PLL or a SerDes becomes jitter on the clock and closes the eye. A **dip** during a current transient, when the FPGA finishes configuration and every I/O bank wakes up at once, drags the core rail through the supervisor threshold for a microsecond and the part resets itself, which from the outside looks like a boot loop with no cause. Sequencing belongs here too. Rails that come up in the wrong order forward-bias ESD structures and latch parts into states the datasheet never describes.

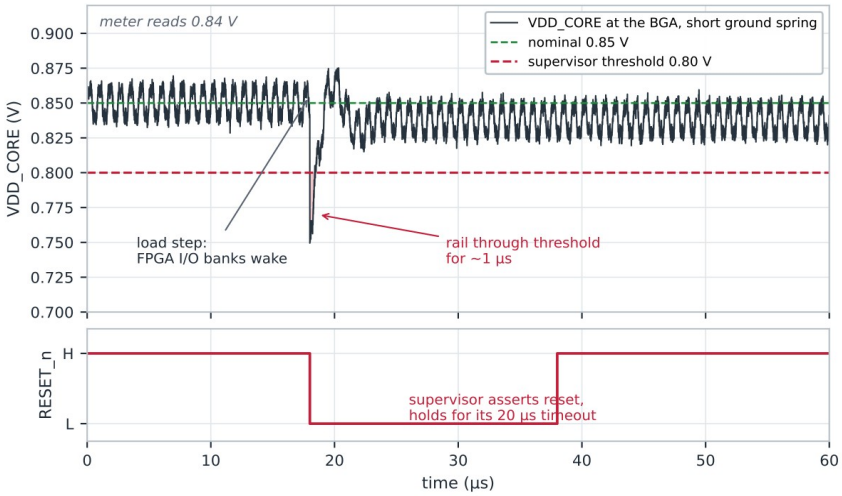


Figure 3 — What the meter cannot see (illustrative). A core rail with normal switching ripple takes a load step, dips through the supervisor threshold for about a microsecond, and recovers. The supervisor asserts reset for its full timeout anyway. The meter reads 0.84 V the whole time.

The meter tells you the average, and the average is fine. Look at each rail on the scope with a short ground spring at the BGA, during the events that draw current, and put the reset line on the same screen so the cause and the effect sit next to each other. Figure 3 shows the shape you are looking for.

Fuel is clock

Fuel has to arrive in the right amount at the right moment on every cycle, and the engine is fussy about both. A clock is fuel. It has to be present, at the right **frequency**, in the right **phase** relative to the things that depend on it, and it has to stay there over temperature and time. Each of those is its own failure. A crystal with the wrong load capacitors runs a few hundred ppm off, and the UART works fine, but the Ethernet link, whose specification allows ±100 ppm, trains and drops as the elastic buffers overflow. A SerDes reference clock that is present at the oscillator but never makes it through a mis-strapped clock buffer leaves the PLL unlocked and the link dead, and the device reports nothing useful because it never got far enough to report. Drift over temperature moves a link that passed at room temperature into a failure at the cold end of the chamber.

Measure the clock at the pin of the device that consumes it, not at the oscillator. Check the frequency against the number in the datasheet to the ppm, and read the PLL lock bit before trusting anything downstream of it. A link that trains and drops is a fuel problem until proven otherwise.

Spark is code

Spark is the part of the engine that is most obviously electrical and the part that fools mechanics into replacing things. A car that cranks but won't catch gets new plugs, new coils, and a new crank sensor before anyone checks whether the timing is off by a tooth. Code is spark. The board comes out of reset with good rails and a clean clock, the processor fetches its first instruction, and ten microseconds later it is in outer space. The stack overflowed into the vector table, an interrupt vector points at the wrong address, a boot strap selected the wrong boot source, or the linker script put the vector table somewhere the processor was never going to look. From the bench it looks exactly like a hardware fault, because the board runs for a moment and dies, and a week goes into probing rails and reflowing BGAs before someone attaches a debugger and finds the program counter in unmapped memory.

The cure is the same discipline in reverse. Once reset, power, and clock are proven, assume the **code** and make it prove otherwise. Halt on reset, single-step the first few hundred instructions, and look at where the program counter went and what the stack pointer was doing when it got there.

Check the four in order

The mechanic's order is not arbitrary and neither is the board's. Reset, then power, then clock, then code, because each one is a precondition for observing the next. You cannot judge a clock on a part that is still in reset, you cannot judge power ripple without the load the code creates, and you cannot debug code on a part whose clock is a few hundred ppm off. There are exotic timing problems and there are strange edge cases, and I have chased my share of them. But nearly every bring-up problem I have ever solved ended with one of the four being wrong, and the only difference between a good week and a bad one was how long it took me to go back and check them.

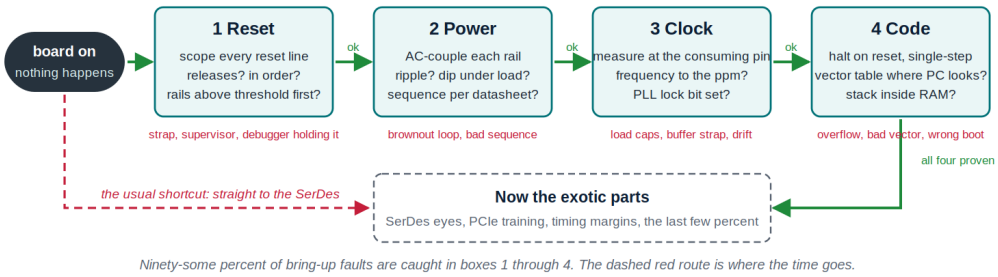


Figure 4 — The order, and the shortcut. Reset, power, clock, and code are checked in sequence, each with the instrument that can see it. The dashed route straight to the SerDes is where the time goes.

Summary

A board that will not come up, or comes up and dies, or brings a link up and drops it, almost always has one of four things wrong. Reset is not releasing, or releases at the wrong time or in the wrong order. Power reads correctly on a meter but ripples or dips under load and trips the supervisor. A clock is missing at the pin that needs it, or is off in frequency, phase, or drift. Or the code took its first few instructions and left, and the hardware is being blamed for a software fault. Work the four in that order, because each is a precondition for seeing the next, and hold the exotic theories until all four are proven. The point is not the checklist but understanding how each of the four works and what it looks like when it is right, so you can prove it rather than assume it. The corner case you are about to spend a week on is usually a poor reset, or a car that is just out of gas.

Field notes (the part you'll actually screenshot)

- Every board needs reset, power, clock, and code before anything downstream can be observed. Check them in that order. Hold the exotic theories until all four are proven.
- Know how each of the four works, not just that it exists. If you cannot say what a correct reset pulse, a clean rail, or a locked PLL looks like on an instrument, you cannot rule it out.
- Scope every reset line at release. Rails above threshold before reset lets go, and release order per the datasheets. A brick that draws the right current is usually a reset that never released.
- A meter reads the average. Scope each rail with a short ground spring at the BGA during the current transients, with the reset line on the same screen.
- Measure clocks at the consuming pin, to the ppm, and read the PLL lock bit before trusting the links they feed. A link that trains and drops is a clock problem until proven otherwise.
- A board that runs for microseconds and dies is code until a debugger says otherwise. Halt on reset and look at where the program counter went.

How this was written

This started as a conversation. The engineering — the four fundamentals, the bench scars behind them, and the calls on what matters — comes from twenty-five years of bringing up boards and chasing links. I guided all of the content, and my friend Claude, Anthropic's AI assistant, did most of the typing and drew the figures. The engine analogy is mine, and I stand by it. Any errors are mine too.

References & further reading

1. James D. Halderman, Automotive Technology: Principles, Diagnosis, and Service, 6th ed., Pearson, 2020 — the no-start diagnosis, in order.
2. Jack Ganssle, The Art of Designing Embedded Systems, 2nd ed., Newnes, 2008 — resets, brownout, and impersonating faults.
3. Robert A. Pease, Troubleshooting Analog Circuits, Butterworth-Heinemann, 1991 — the philosophy of troubleshooting.
4. Howard W. Johnson and Martin Graham, High-Speed Digital Design: A Handbook of Black Magic, Prentice Hall, 1993 — clocks, jitter, and supply noise in disguise.
5. IEEE Std 802.3, Ethernet — the ±100 ppm clock tolerance a mis-loaded crystal quietly violates.