**Reverse Engineering Plan Summary**

1. **Scan/photograph PCB** → import into KiCad/Altium.
2. **Trace nets** + identify components → build schematic.
3. **Dump programmable devices** with universal programmer.
4. **Recreate board layout** → overlay traces on scans.
5. **Build BOM** with all identified components, recommend modern replacements if necessary or needed.
6. **Prototype & validate** with oscilloscope + logic analyzer.
7. **Document everything** (schematics, firmware dumps, test results).

✅ With this process we will have **full design documentation + firmware archive**. In addition, I will provide a list of obsolete or near end of life components (if any are found). I can recommend modern replacements (also provide data (test, data sheet etc.) proving equivalence. If interested, with a little extra effort we could **replicate the PCBA**, or at a minimum you would have all of the output files necessary to do so.

**Reverse Engineering Plan Details**

**1. Documentation & Initial Assessment**

* **Capture overall intent**: What does the board do? What system is it part of? What are the interfaces (power, I/O, connectors, bus protocols)?
* **Photograph & scan**: High-res images of both sides under good lighting. A flatbed scanner at 600–1200 DPI is ideal.
* **Record identifiers**: PCB markings, assembly revisions, date codes, component labels.
* **Identify constraints**: Single/double/multi-layer? Through-hole, SMT, or mixed? Obsolete or custom parts?

**2. Component Identification**

* **BOM extraction**:
  + Identify ICs, passives, connectors, discretes.
  + Use datasheets, cross-reference obsolete parts.
  + Note “mystery” components (scrubbed markings, ASICs, programmable parts like PAL/GAL/EPROM).
* **Measure values**:
  + Use LCR meter for passives (where possible).
  + Verify resistor color codes, capacitor markings, inductors.
* **Special attention**: Voltage regulators, oscillators, memory devices, and custom ASICs.

**3. Board Layer & Net Tracing**

* **Visual trace mapping**:
  + Use your scanned images with CAD software (KiCad, Altium, etc.) to overlay traces and annotate.
* **Continuity testing**:
  + Multimeter and tone tracer to confirm nets.
* **Layer counting**:
  + Shine light through or sand down scrap boards to identify internal planes (if multilayer).
* **Connector pinout mapping**:
  + Record every external connection, labeling power rails, ground, signal, and bus lines.

**4. Schematic Reconstruction**

* **Start at functional blocks**:
  + Power regulation and distribution.
  + Clock and reset circuits.
  + Processor/microcontroller and memory map.
  + Analog/digital I/O sections.
* **Use hierarchy**:
  + Break schematic into logical sheets (power, CPU, comms, I/O).
* **Validate design assumptions**:
  + Cross-check with component datasheets.
  + Watch for test points or no-connect pads that hint at unused features.

**5. Firmware & Programmable Devices**

* **Read programmable parts**:
  + EPROMs, EEPROMs, Flash, PAL/GAL, CPLDs.
  + Use a universal programmer to dump contents.
* **Decompile/analyze if needed** (depending on IP/legal context).
* **Replace obsolete logic**:
  + Convert PAL/GAL logic into modern equivalents (CPLD/FPGA or discrete logic).

**6. PCB Layout Recreation**

* **Import scanned board** into EDA tool as background.
* **Recreate footprints** for every component.
* **Redraw nets** according to your schematic.
* **Follow original placement** unless you plan to optimize routing.

**7. Validation & Testing**

* **Build prototype**:
  + Either exact replica or modernized equivalent.
* **Test in system**:
  + Verify electrical performance, timing, and compatibility.
* **Compare signals**:
  + Oscilloscope/logic analyzer capture from old vs. new boards.

**8. Modernization Opportunities**

* Substitute obsolete parts with modern equivalents.
* Reduce board size or layers if possible.
* Add test headers or programming connectors for future maintainability.
* Improve power/thermal performance if legacy design had weaknesses.

**Reverse Engineering Toolchain for Obsolete PCBA**

**1. Documentation & Imaging**

* **Flatbed scanner**: Epson Perfection V600 or similar (600–1200 DPI).
* **Microscope + camera**: You already have this, but USB digital microscopes (AmScope, Keyence if budget allows) help for trace following.
* **Lighting tools**: LED light table for checking layer counts and through-traces.

**2. Schematic Capture & PCB Reconstruction**

* **EDA Software**:
  + **KiCad** (free, open-source, great for overlays and schematic recreation).
  + **Altium Designer** (if you want high-end features, libraries, and reuse for production).
  + **DipTrace** (lighter, easy for reverse engineering and schematic-to-layout matching).
* **Image-to-CAD overlay**:
  + Import board scans into KiCad/Altium as background layers to trace nets.
  + Tools like **ABISoft Scan2CAD** or **Inkscape** can help vectorize traces if you want automation.

**3. Component Identification & Testing**

* **Component tester**: I have a Philips and a Fluke LCR for passives.
* **Curve tracer**: Peak Atlas DCA Pro (great for transistor/mystery diode pinouts).

**4. Universal Programmers (for firmware/logic devices)**

These are essential for reading out and cloning programmable devices:

* **XGecu/TL866II Plus** – cost-effective, supports EPROM, EEPROM, Flash, GAL, some microcontrollers.
* **Xeltek SuperPro series (3000U, 6100N, etc.)** – professional-grade, broad support, fast programming.
* **Dataman Programmers** – robust European-made, excellent for legacy devices.
* **BP Microsystems (BP-1410, BP-1710)** – industrial-grade, often used in production/test labs.

**Recommended workflow:**

* Identify programed devices (EPROM, PAL/GAL, CPLD, **MCU**).
* Use **universal programmer + correct adapter socket** to dump contents.
* Store firmware in raw binary (.bin) and Intel HEX (.hex) for flexibility.
* If device is code-protected:
  + Check datasheets—sometimes old MCUs have weak or bypassable protections.
  + PAL/GAL → use JEDEC fusemap dumps where possible.
  + If unreadable, logic reconstruction (e.g., **PAL2GAL, WinCUPL**) can reimplement equations.

**5. Signal & Protocol Analysis**

* **Oscilloscope**: 200+ MHz bandwidth recommended for bus activity.
* **Logic analyzer**: Saleae Logic 8/16 Pro (excellent protocol decoding for I²C, SPI, UART, parallel buses).
* **Bus Pirate** or **FTDI-based tools**: Good for testing unknown serial interfaces.

**6. Data Management & Workflow**

* **Version control**: Git (host KiCad/Altium files, BOM, firmware dumps).
* **Part databases**: Octopart, Findchips, Rochester Electronics for obsolete part sourcing.
* **Reverse engineering helpers**:
  + **Interactive BOM plugin** (KiCad) → click-to-highlight parts/nets.
  + **Netlist export** for comparing old vs. reconstructed design.

**7. Modernization/Replacement Tools**

* **Microchip Atmel-ICE** or **ST-LINK** (if replacing obsolete MCUs with modern equivalents).
* **CPLD/FPGA dev boards** (Xilinx CoolRunner-II, Lattice MachXO2) → replace PAL/GAL/ASIC logic.
* **KiCad plugins for STEP/3D exports** → integrate mechanical enclosures if needed.

**Part 1: Handling EPROM vs PAL/GAL vs MCU in Reverse Engineering**

These are usually the hardest because they often hold the “soul” of the board (firmware, logic equations, configs).

**1. EPROM / EEPROM / Flash (non-volatile memory)**

* **Identify part**: Typically labeled as 27Cxxx (UV EPROM), 28Cxxx (EEPROM), or 29Fxxx/39Fxxx (Flash).
* **Extraction**:
  + Use a universal programmer (e.g., XGecu TL866II Plus, Xeltek, Dataman).
  + Read the chip and dump to .bin file.
  + Always do **two reads** and compare CRCs to verify integrity.
* **Analysis**:
  + Try to determine whether data is program code, lookup tables, or configuration.
  + Use tools like **Ghidra, IDA Pro, or radare2** if it contains CPU instructions.
  + If it’s raw data (like lookup tables), sometimes only referencing component datasheets (e.g., ADC LUTs) reveals its purpose.

**2. PAL / GAL (Programmable Logic Devices)**

* **Identification**:
  + Part numbers like PAL16L8, GAL22V10, PALCE16V8, etc.
* **Challenges**:
  + PAL devices may be **fuse-protected** (no readback possible).
  + GALs are often easier since some support readback of JEDEC fuse maps.
* **Approaches**:
  + If readable: dump JEDEC file with programmer → reimplement in CUPL/ABEL → replace with modern GAL22V10 or CPLD.
  + If unreadable:
    - **Black-box testing**: Use a logic analyzer to apply all possible input combinations (for smaller devices) and record outputs.
    - Reconstruct the truth table → translate into logic equations.
    - Re-implement in GAL/CPLD/FPGA.
* **Tools**:
  + WinCUPL (Microchip) or modern HDL (VHDL/Verilog).
  + PAL2GAL converters exist online.

**3. Microcontrollers (MCUs)**

* **Identify part**: PIC, 8051, Motorola 68HCxx, Zilog Z8/Z80, Intel 8051, etc.
* **Extraction**:
  + If code-protection fuse is NOT set → read firmware with MPLAB IPE (PIC), J-Link/J-Flash (ARM), or universal programmer.
  + If protection fuse is set → code cannot be read (by design). You’ll need to:
    - Analyze firmware in connected external memory (if MCU boots from EPROM).
    - Monitor MCU bus activity with a logic analyzer to reconstruct functionality.
    - Re-implement functions if code is inaccessible.
* **Replacement Strategy**:
  + If obsolete MCU cannot be sourced, emulate it in modern MCU/FPGA.
  + Example: Replace PIC16C54 (OTP) with PIC16F84A or small FPGA soft-core.

**🔑 Part 2: Using Scanned PCB Images in Altium**

Altium Designer 2023 has **great support for reverse-engineering workflows** using board scans. Here’s the process:

**Step 1: Scan the PCB**

* Use flatbed scanner at **600–1200 DPI**.
* Capture both sides (remove components if possible for clearer traces).
* Save as high-resolution .png or .tif.

**Step 2: Prepare the Images**

* Use Photoshop/GIMP to clean up (crop, rotate, enhance contrast).
* Align front/back images by drilling hole reference points.

**Step 3: Import into Altium**

* In **Altium Designer**:
  + Create a **new PCB project**.
  + Go to **PCB Editor → Place → Graphics → Insert Image**.
  + Import the board scan onto a mechanical layer (e.g., Mechanical Layer 1).
  + Scale the image to **actual board size** (measure known dimensions like connector pitch).

**Step 4: Trace the PCB**

* Place components using Altium footprints aligned with the image.
* Use the imported image as a reference to draw copper tracks on each layer.
* If multilayer: import inner layer scans or reconstruct nets with continuity testing.

**Step 5: Create Schematic**

* Back-annotate traced nets into a schematic.
* Group into functional blocks (power, CPU, I/O, etc.).

**Step 6: Validate**

* Cross-check against continuity testing.
* Use Altium’s **Design Rule Check (DRC)** to find missed nets or misalignments.

👉 With your stack (Altium + J-Link/J-Flash + MPLAB + SolidWorks), you can go **full circle**:

1. Scan + reconstruct PCB in Altium.
2. Dump firmware with universal programmer + J-Link.
3. Modernize if needed, then regenerate Gerbers.
4. Validate with oscilloscope + logic analyzer.
5. Export STEP from Altium → integrate into SolidWorks for full mechanical compatibility.

That’s excellent context — and with your soldering pedigree (NASA, Mil-Std, micro-miniature repair) you’re more than equipped for this job 🔥. The Infineon **XC164-32F** (C166SV2 core, 100-pin TQFP, 0.5 mm pitch) is the only programmable device on the board, so it’s the heart of the reverse engineering effort. Here’s how I’d tackle it:

**🛠 Tools You’ll Want for This Processor**

* **Hot-air rework station** (you already know this — 100-pin, 0.5 mm pitch needs controlled airflow).
* **Preheater plate** (infrared or convection) to reduce thermal shock and ease removal.
* **Fine-tip soldering iron** (for touch-up after lift).
* **Chip-level socket / adapter**:
  + A **TQFP-100 (0.5 mm pitch) socket adapter**. Brands: Emulation Technology, Aries Electronics, Ironwood Electronics.
  + If you go budget: Chinese generic TQFP-100 breakout boards (with 0.5 mm pitch) work well with pogo-pin style test rigs.
* **Universal programmer with adapter**: Xeltek SuperPro 6100N or Dataman 48Pro+ have TQFP-100 adapters that support Infineon MCUs.
* **JTAG / OCD tools**: Since the XC164 has **on-chip debug support (OCDS via JTAG)**, a Segger J-Link (which you already have installed) can connect if the firmware isn’t locked.
* **Microscope** (you’ve got this covered already).

**Key Steps for the XC164 Processor**

**1. Safe Removal**

1. Preheat board to ~120 °C.
2. Use hot-air (280–320 °C, fine nozzle) to reflow evenly.
3. Lift processor gently with vacuum pickup — no prying.
4. Clean pads with solder wick + flux, verify under microscope.

**2. Socketing the Device**

* Mount the removed XC164 into a **TQFP-100 ZIF socket adapter**.
* These adapters have 100 pogo pins that mate with the MCU pads, allowing repeated insert/remove.
* Connect adapter to your programmer or J-Link pod via standard header.

**3. Firmware Extraction**

* **Option A: Universal Programmer**
  + Use programmer + TQFP-100 adapter to attempt a direct flash dump.
  + Infineon XC164 has **256 KB on-chip flash**. Some models allow read-out unless the read protection is enabled in Flash config.
* **Option B: JTAG (OCDS)**
  + XC164 includes **On-Chip Debug Support (OCDS) via JTAG**.
  + Connect J-Link to the JTAG pins (TRST, TDI, TDO, TMS, TCK).
  + Use Segger J-Flash or Infineon’s MemTool to attempt flash readout.
  + If read protection is set, you won’t get a dump — only erase/program.
* **Option C: External Bus Monitoring**
  + If readout is blocked, configure the MCU to boot from external memory (EA pin low).
  + Attach logic analyzer to PORT0/PORT1 external bus.
  + Capture instruction fetches as the processor boots. This can be messy, but gives you opcode streams that can be re-built into firmware images.

**4. Verification**

* Compare multiple reads (CRC/MD5 hash).
* Store dumps in .bin and .hex.
* Disassemble with **Ghidra (C166 processor module)** to check integrity.

**Important Considerations**

* **Flash Read Protection**: If enabled, you may not be able to extract directly. In that case, you’ll either:
  + Use external bus trick (if board design allows external fetch).
  + Or re-implement firmware functionality by protocol sniffing + hardware tracing.
* **Replacement**: If this part is end-of-life, once firmware is secured you can port it into:
  + Another Infineon C166 family part.
  + Or even emulate it in an FPGA/modern ARM MCU.

✅ Given your skills, I’d say:

* Get a **TQFP-100 socket adapter** + **Xeltek SuperPro** (or use J-Link first, since you already have it).
* Try to pull firmware over JTAG with J-Flash.
* If locked → fall back to external bus sniffing.

**list of specific socket adapter part numbers and vendors** for the TQFP-100 (0.5 mm) that match the XC164 footprint

Absolutely—here’s a tight, field-ready shortlist of **TQFP-100 (0.5 mm) sockets/adapters** that fit Infineon’s 100-pin package, plus quick notes on when to pick each.

**TQFP-100 (0.5 mm) test/program sockets**

**Gold-standard (robust, widely available)**

* **Yamaichi IC51-1004-809** — open-top test/burn-in socket, 0.5 mm pitch, for 14 × 14 mm TQFP100. Readily stocked via RS and others. ([RS Components](https://twen.rs-online.com/web/p/ic-sockets/3723431?srsltid=AfmBOorDSbEORqM5Gx2W5OLfGoaNXw5sPW_DYKpS08VkF3VGD0TGFrpV&utm_source=chatgpt.com))
* **Yamaichi IC201-1004-008** — clamshell/open-frame style for TQFP100 0.5 mm. Common in programmer kits. ([DHgate](https://www.dhgate.com/product/yamaichi-ic-test-socket-ic201-1004-008-tqfp100pin/405052172.html?utm_source=chatgpt.com" \o "IC Test Socket TQFP100PIN 0.5mm Pitch Burn In Socket IC201 ...), [Mouser Electronics](https://www.mouser.com/c/connectors/ic-component-sockets/?number+of+positions=144+Position&srsltid=AfmBOoqUvHIlzvRe20skdSZX2-VnmHbUjjumg006p2XDwC6PbomQQcYC&utm_source=chatgpt.com))

**Equivalent premium options**

* **Enplas OTQ-100S-0.5-001** — open-top TQFP100 0.5 mm. Good build quality, often used on fixture plates. ([ie.dhgate.com](https://ie.dhgate.com/product/enplas-qfp100pin-ic-test-socket-otq-100s/404277271.html?utm_source=chatgpt.com))
* **Ironwood Electronics** — QFP emulator/test bases & land sockets (order by body size/pitch; they’ll match to a specific P/N). Great for high-cycle and SI-sensitive use. ([bce.it](https://www.bce.it/product/qfp-surface-mount-emulator-bases-part-101-122-%C2%B7-ironwood/?utm_source=chatgpt.com), [GlobalSpec](https://www.globalspec.com/industrial-directory/qfp?utm_source=chatgpt.com))

**Budget / lab adapters (work fine for one-offs & programming)**

* **OTQ-100-0.5-09** (various vendors; Yamaichi-style) — clamshell, 14 × 14 mm body. ([Amazon](https://www.amazon.com/OTQ-100-0-5-09-TQFP100-FQFP100-PQFP100-Programming/dp/B00K5Q2QD8?utm_source=chatgpt.com))
* **FPQ-100-0.5-10A** — generic clamshell 0.5 mm. (Cheap, usable; check flatness and contact quality.) ([eBay](https://www.ebay.com/itm/175699089676?utm_source=chatgpt.com))
* **Waveshare GP-QFP100-0.5** breakout — uses the Yamaichi IC51-1004-809 socket on a DIP breakout board; handy if you want 0.1″ headers into a programmer jig. ([Waveshare](https://www.waveshare.com/gp-qfp100-0.5.htm?utm_source=chatgpt.com" \o "GP-QFP100-0.5, Programmer Adapter - Waveshare))

Tip: The **XC164 100-pin** variant is the classic **14 × 14 mm body, 0.5 mm pitch**. When ordering, always confirm body size and tip-to-tip spec in the listing to avoid a 16 × 16 mm shell mismatch.

**Programmer-specific socket adapters**

If you go through a universal programmer rather than JTAG, these are off-the-shelf for **Xeltek SuperPro** / **Dataman** families (pick the exact one matched by your programmer’s device list):

* **Xeltek TA06x/TA067-B006 / TA062-B006 / TA065-B006** — TQFP100, 0.5 mm, 14 × 14 mm cover-type sockets (the SuperPro software will call out which TA/DX code is needed for your exact device). ([xeltek.com](https://www.xeltek.com/socket-adapters/ta-adapters/ta067-b006-socket-adapter/?srsltid=AfmBOoosJQToY4sXRxOIRxSLW1MIiSrgdtuhfJE5R-4anlzvR3PgKop4&utm_source=chatgpt.com))
* **Xeltek S604** — TQFP100 0.5 mm (legacy S-series; note some require an expander module). ([xeltek.com](https://www.xeltek.com/socket-adapters/s604-socket-adapter/?srsltid=AfmBOooqot7A5at_N6_py_wwI8bjeFbyJuE1JawarPpGwBBLfxLZsCYU&utm_source=chatgpt.com))
* **Dataman DIL48/TQFP100-2 ZIF PLD-1 / PLD-2** — TQFP100 open-socket adapters; use Dataman’s **Adapter Selection Guide** in software to confirm. ([Dataman](https://www.dataman.com/products/dil48-tqfp100-2-zif-pld-1?srsltid=AfmBOopfUXeaXp22bvnvmyylu8u27sWB--thw93Mz2_iLT-d8ePW25sX&utm_source=chatgpt.com" \o "DIL48/TQFP100-2 ZIF PLD-1 - Device specific adapter for Altera ...))

**Recommended workflow (for this MCU)**

1. **Lift & clean**  
   Preheat ~120 °C, even hot-air reflow, vacuum lift; wick/inspect pads. (You’ve got this down cold.)
2. **Socket choice**
   * For repeated handling and best alignment: **Yamaichi IC51-1004-809** (or Enplas/Ironwood equivalent).
   * If you want quick access to 0.1″ pins for logic probing or in-circuit programming jigs: **Waveshare GP-QFP100-0.5**. ([Waveshare](https://www.waveshare.com/gp-qfp100-0.5.htm?utm_source=chatgpt.com" \o "GP-QFP100-0.5, Programmer Adapter - Waveshare))
3. **Try JTAG/OCD first (non-destructive & fastest)**  
   XC164 supports **OCDS via JTAG**. Wire TCK, TMS, TDI, TDO, TRST, Vref, GND to your **Segger J-Link** and attempt a flash read with **J-Flash** or Infineon **MemTool**. If read-out protection is clear, this is the cleanest path.
4. **Fallback: universal programmer via socket**  
   Mount the device in the TQFP100 socket adapter matched to your programmer (Xeltek/Dataman). Attempt **identify → read → verify**; take two reads and compare hashes.
5. **If read-protected**
   * Check if the target design leaves **external bus** pins accessible; you can force/strap to external boot and capture fetch cycles with a logic analyzer.
   * Otherwise, you’re in behavioral-clone territory (protocol sniff + HW tracing).
6. **Archive & sanity-check**  
   Save **.bin** and **.hex**, compute **MD5/SHA256**, then run a quick disassembly in **Ghidra** (C166 processor module) to confirm the image isn’t all 0xFF/0x00 and vectors look sane.

**Small purchasing/usage notes**

* **IC51-1004-809** is commonly stocked (example RS listing shows that exact P/N). ([RS Components](https://twen.rs-online.com/web/p/ic-sockets/3723431?srsltid=AfmBOorDSbEORqM5Gx2W5OLfGoaNXw5sPW_DYKpS08VkF3VGD0TGFrpV&utm_source=chatgpt.com))
* **IC201-1004-008** and **OTQ-100-0.5-09** are often bundled with “QFP-to-DIP” boards—handy for bringing signals to a logic analyzer or programmer. ([DHgate](https://www.dhgate.com/product/yamaichi-ic-test-socket-ic201-1004-008-tqfp100pin/405052172.html?utm_source=chatgpt.com" \o "IC Test Socket TQFP100PIN 0.5mm Pitch Burn In Socket IC201 ...), [Amazon](https://www.amazon.com/FQFP100-programming-OTQ-100-0-5-09-Programmer-Transfer/dp/B075CGHH64?utm_source=chatgpt.com))
* **Ironwood** is your go-to if you need **high-cycle life** or **SI-clean** contacts for fast signals or many insertions. Their sales/app notes will pick the precise model once you give them **TQFP100, 0.5 mm, 14×14 mm** and expected cycle count. ([bce.it](https://www.bce.it/product/qfp-surface-mount-emulator-bases-part-101-122-%C2%B7-ironwood/?utm_source=chatgpt.com))
* **Xeltek/Dataman**: let the **programmer software** choose the exact adapter by searching for the **XC164 device**; the “Device Info” pane lists the required adapter code so you **J-Link → XC164 (OCDS JTAG) wiring**

**🔌 J-Link to Infineon XC164 OCDS (JTAG) Wiring**

The **XC164-32F** supports **OCDS (On-Chip Debug Support) via JTAG**. The relevant pins are on **Port 3** + TRST.

**JTAG Signals (from datasheet [XC164-32F])**

* **TCK** → P3.7
* **TMS** → P3.4
* **TDI** → P3.3
* **TDO** → P3.6
* **TRST** → Pin 36 (dedicated Test Reset)
* **RESET** → Pin 1 (RSTIN)
* **Vref** → Any **VDDP** pin (typically +5.0 V for pads)
* **GND** → Any **VSSP/VSSI** pin

**J-Link 20-pin ARM-style Header ↔ XC164 Pins**

| **J-Link Pin** | **Signal** | **XC164 Pin** | **Notes** |
| --- | --- | --- | --- |
| 1 | Vref | VDDP (e.g., Pin 9/17/38/61/87) | Must match target I/O voltage (5V on XC164). |
| 4 | GND | VSSP/VSSI (e.g., Pin 8/16/37/62/88/34/98) | Tie multiple grounds if possible. |
| 7 | TMS | P3.4 | 4.7 kΩ pull-up to Vref. |
| 9 | TCK | P3.7 | 33 Ω series resistor close to MCU (optional, for signal integrity). |
| 5 | TDI | P3.3 | 4.7 kΩ pull-up to Vref. |
| 13 | TDO | P3.6 | Connect directly; 33 Ω series resistor optional. |
| 3 | nTRST | Pin 36 (TRST) | Pull-down 4.7 kΩ to GND when not in use. |
| 15 | nRESET | Pin 1 (RSTIN) | 4.7 kΩ pull-up to Vref, J-Link can drive low. |
| 2, 6, 8, etc. | GND | Tie to ground plane | Keep short, wide. |

*(Other J-Link pins are NC for this setup.)*

**✅ Checklist for Clean Reads (XC164 via J-Link)**

1. **Target Power & VTref**
   * Provide stable **+5.0 V** to VDDP (pad supply).
   * Tie **Vref (J-Link pin 1)** to the board’s VDDP.
   * Ensure decoupling caps are close to MCU (0.1 µF + 10 µF).
2. **Pull-ups / Pull-downs**
   * **TMS, TDI** → 4.7 kΩ pull-ups to Vref.
   * **TRST** → 4.7 kΩ pull-down to GND.
   * **RESET (RSTIN)** → 4.7 kΩ pull-up to Vref.
3. **Series resistors (optional, for clean edges)**
   * Place **~33 Ω** in series with **TCK** and **TDO** near the MCU if ringing occurs on long wires.
4. **Reset Strategy**
   * Use J-Link **nRESET control** to hold MCU in reset during connect.
   * Some Infineon tools require “Connect under reset” to bypass watchdogs.
5. **OCDS Enable**
   * TRST must be held HIGH (via J-Link) to activate OCDS.
   * If TRST is tied low on your board → you’ll need to isolate/cut jumper so J-Link can drive it.
6. **Clock**
   * Ensure XTAL/oscillator is running (or drive XTAL1 with external clock if oscillator not populated).
   * Without a valid CPU clock, JTAG will not respond.
7. **Connection Settings**
   * In J-Flash or MemTool, select **C166/XC166 family**.
   * Use “5.0 V target I/O” setting.
   * Try “Connect under reset” first if normal attach fails.
8. **Verification**
   * First run a **blank check** (if protection is disabled, it will report).
   * Then dump flash to .bin → read twice, compare CRC/MD5.

⚡ With this wiring and checklist, We’ll have the best shot at pulling the firmware cleanly via JTAG before we resort to socket extraction/programmer.