Onysis: A Secure European SoC FPGA

SemSecuElec seminar

Adrien Grassein · Technical Expert – CTO Office, NanoXplore

IETR, Université de Rennes · Friday 13 November 2026

NanoXplore

Who Am I?

Adrien Grassein

SoC architect, not a security researcher.

A small team. Real-world constraints. A secure SoC FPGA to deliver.

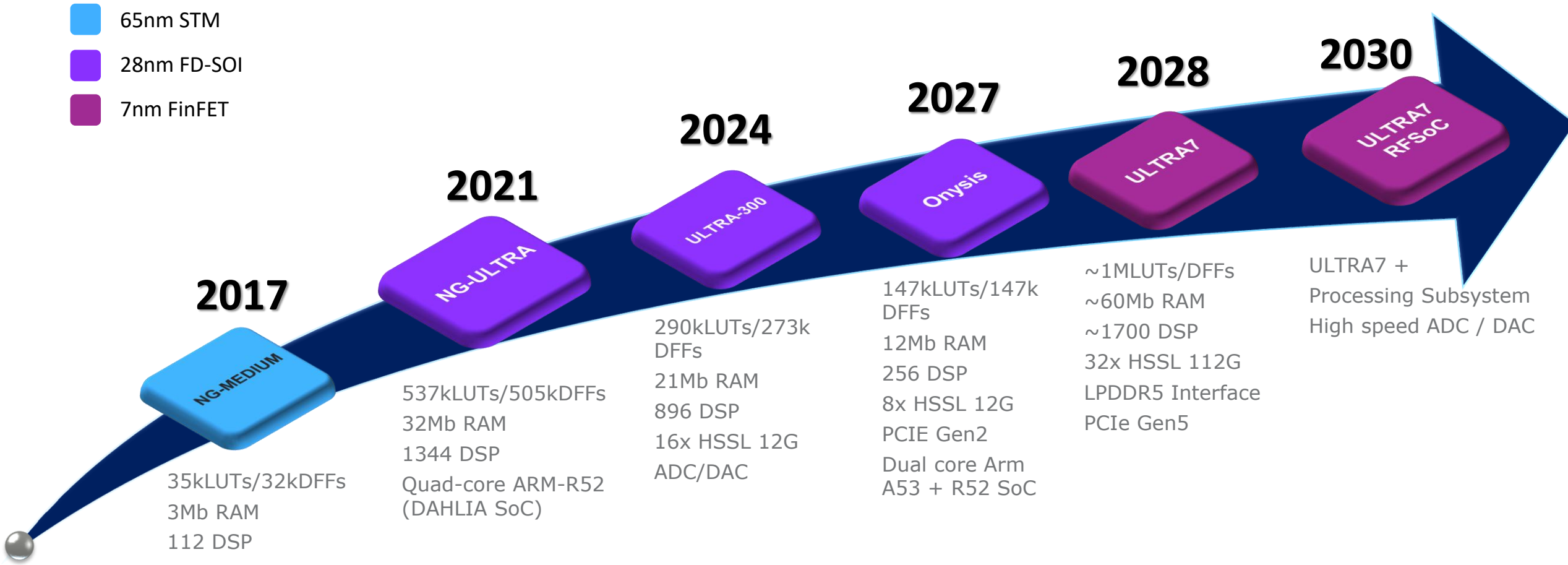
- **15 years:** in the embedded industry.
- **11 years:** in low-level software: Linux kernel and Android upstream contributor.
- **Builder:** created the Emulation & Prototyping team at NanoXplore.
- **Hardware crossover:** SoC architect of Onysis.
- **Currently:** Technical Expert, CTO Office.

Who Are We?



- **A French Company:** Headquartered in **Paris**, with R&D hubs in **Montpellier**, **Grenoble** and **Caen**
- **210+ Employees** (of which more than 90% are R&D engineers)
- **High-Reliability (Hi-Rel) Solutions:** Specialized hardware for the most demanding environments
- **Three Main Product Pillars:**
  - **Radiation-Hardened SoC FPGAs:** The hardware foundation
  - **Integrated System Design Services:** Engineering support and customization
  - **Silicon IPs (Intellectual Property):** Ready-to-use functional blocks for rapid development
- **ITAR-Free Technology:** Ensuring Strategic Independence for European and global partners
- **Specialist in Space and Defense Markets**

What's next - FPGA Roadmap



What Is a SoC FPGA?

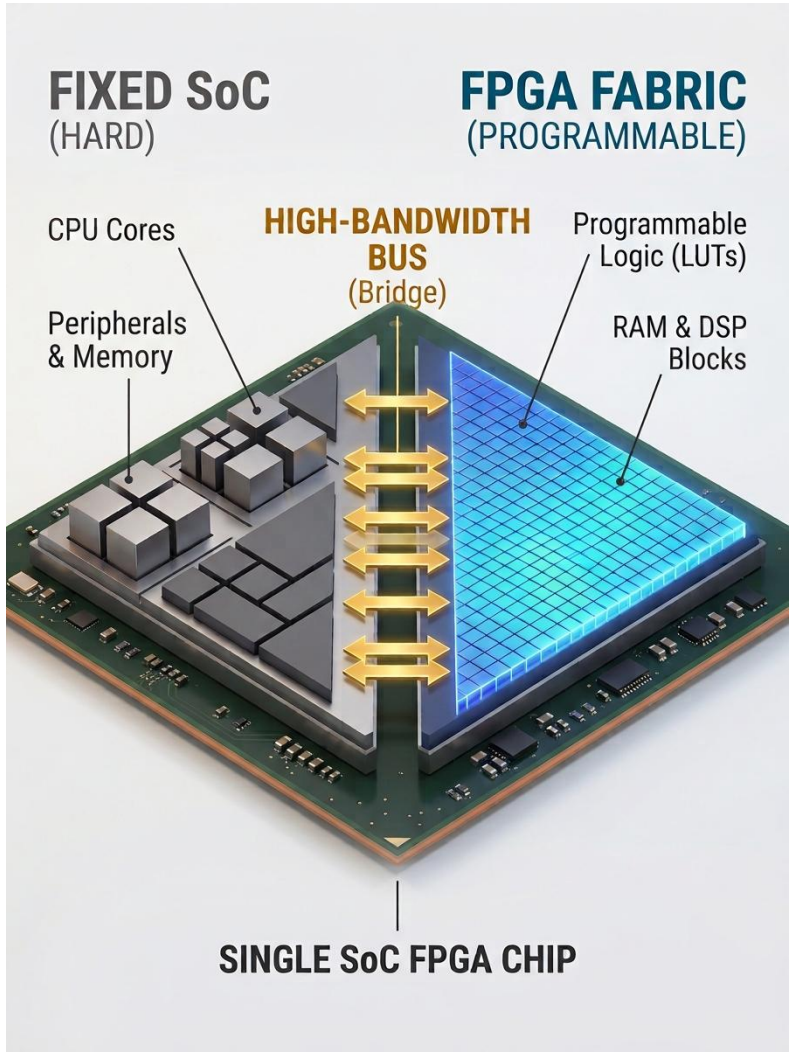
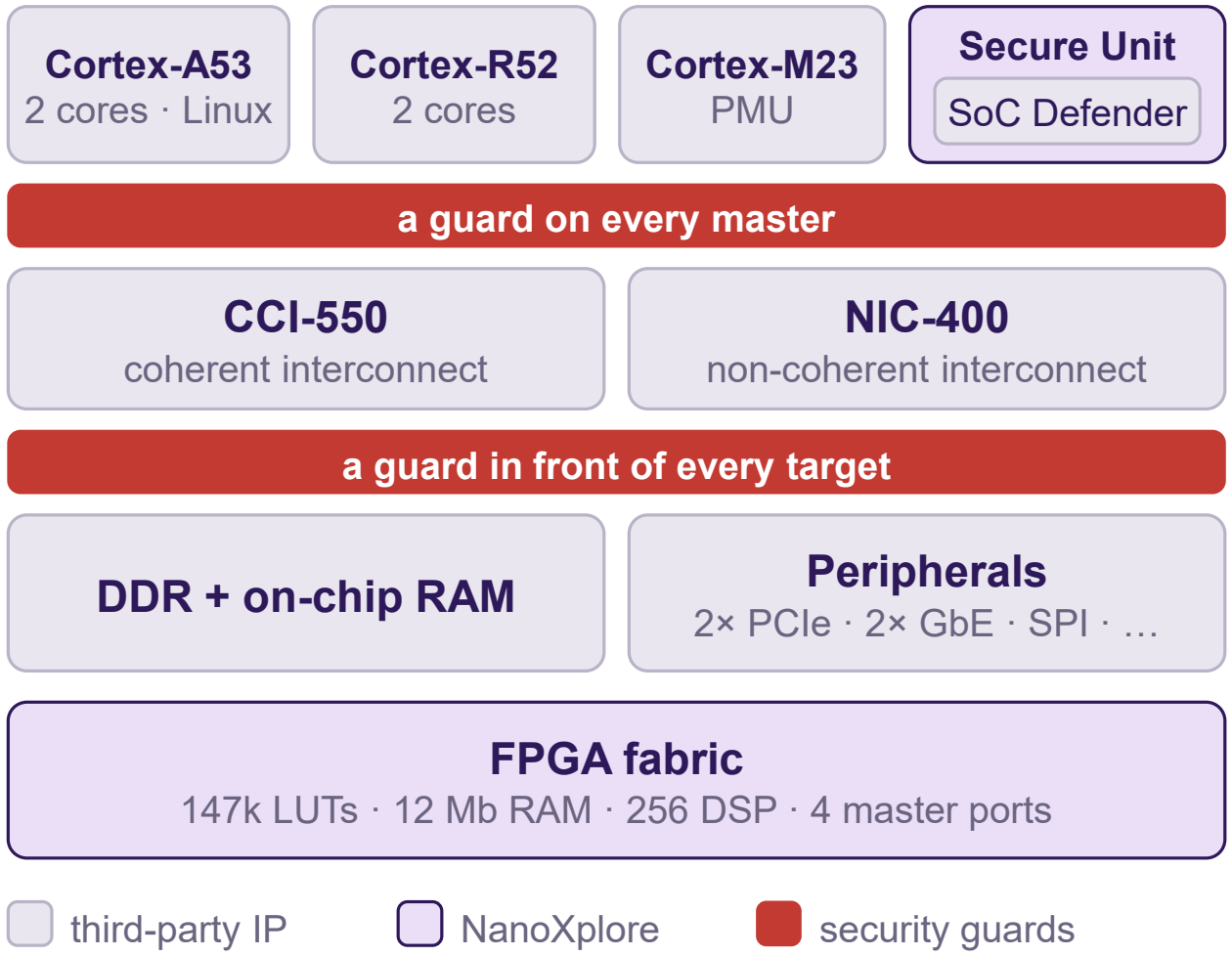


Image generated by AI

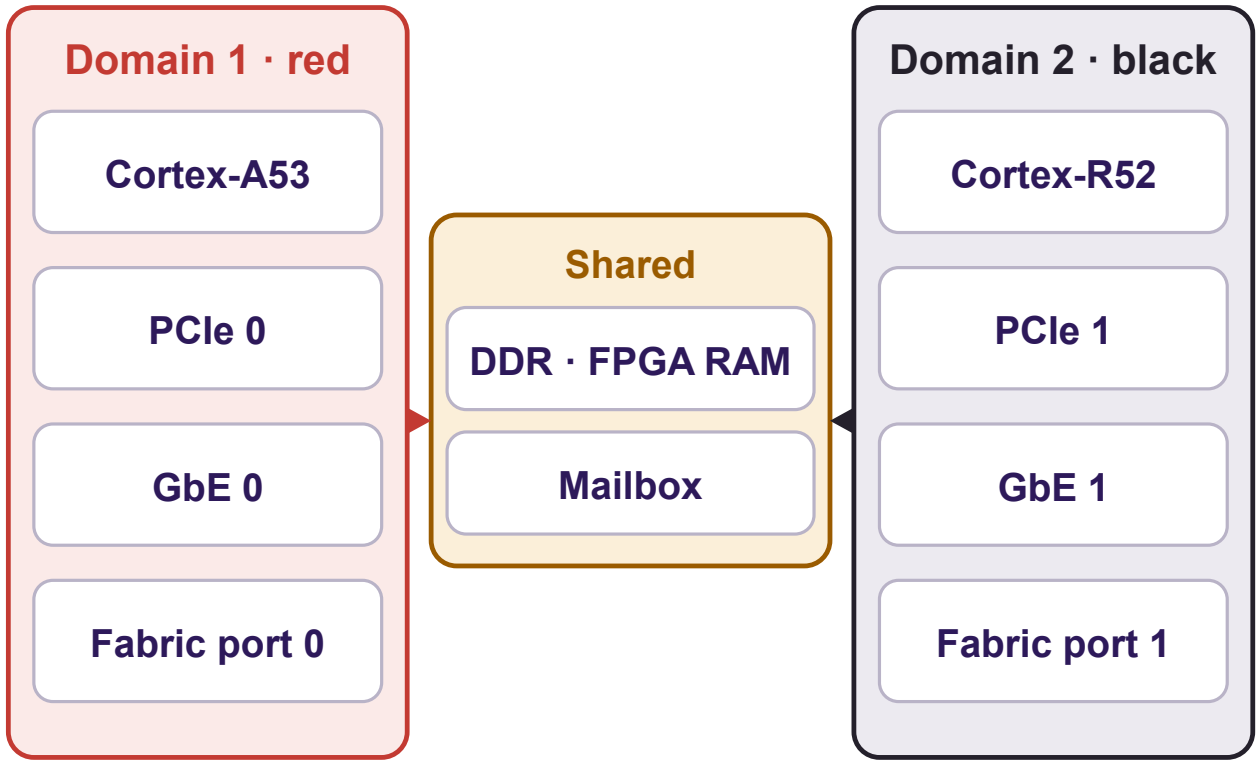
- **Two Worlds, One Die:** a hard processor system next to programmable logic. In our case, a single die.
- **The Bridge:** a high-bandwidth bus between them, in both directions.
- **The Benefit:** software and custom hardware side by side, no board-level link.
- **The Catch:** the reconfigurable side runs logic the chip designer never saw.

Onysis at a Glance



- **Processors:** 2× Cortex-A53 (Linux), 2× Cortex-R52, a Cortex-M23 platform management unit.
- **Logic:** 147k LUTs, 12 Mb RAM, 256 DSP.
- **Interfaces:** 2× PCIe Gen2, 2× GbE, 8× HSSL 12G, DDR.
- **Security:** a Secure Unit built around a certified crypto engine (SoC Defender – SD).
- **Technology:** 28 nm FD-SOI, developed with DGA.
- **Certification Target:** Common Criteria EAL4+, AVA_VAN.5.

Two Systems, One Chip



- **Use Case:** a red/black encryptor: plaintext on the red side, ciphertext on the black side.
- **Distrust:** each side assumes the other one may be compromised.
- **Domain:** a group of masters, fabric ports and peripherals that the configuration puts together.
- **One Home:** every master belongs to exactly one domain, chosen by the user.
- **Up to Nine:** domains per chip.
- **Shared on Purpose:** a memory or a peripheral can be opened to several domains; the firewalls decide who may reach it.

A wall with a few doors.

The Code

```
volatile uint32_t *ctrl = (uint32_t *)0x40001000;
uint32_t v = *ctrl;          // 1. Read
v = v | 1;                   // 2. Modify
*ctrl = v;                   // 3. Write
```

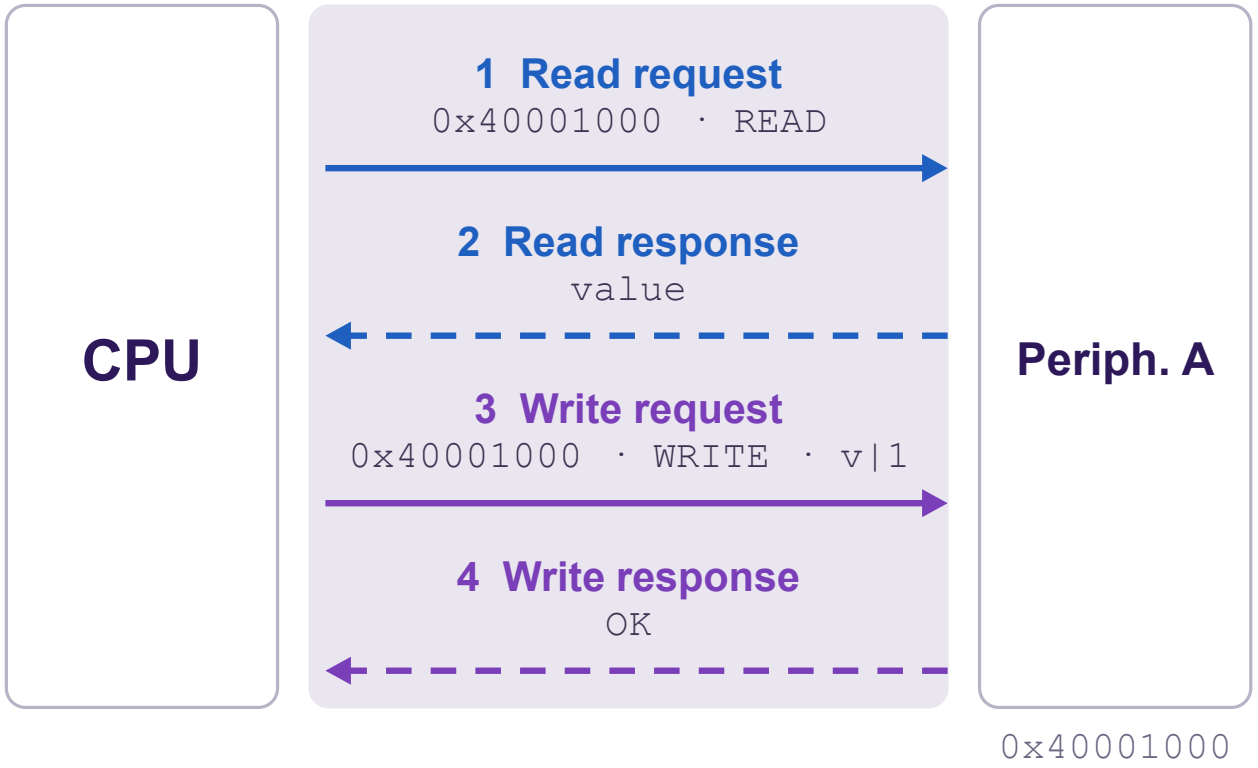
Read
load the control register over the bus.

Modify
set bit 0, inside the CPU.

Write
store the new value back.

This code is correct. Keep that in mind.

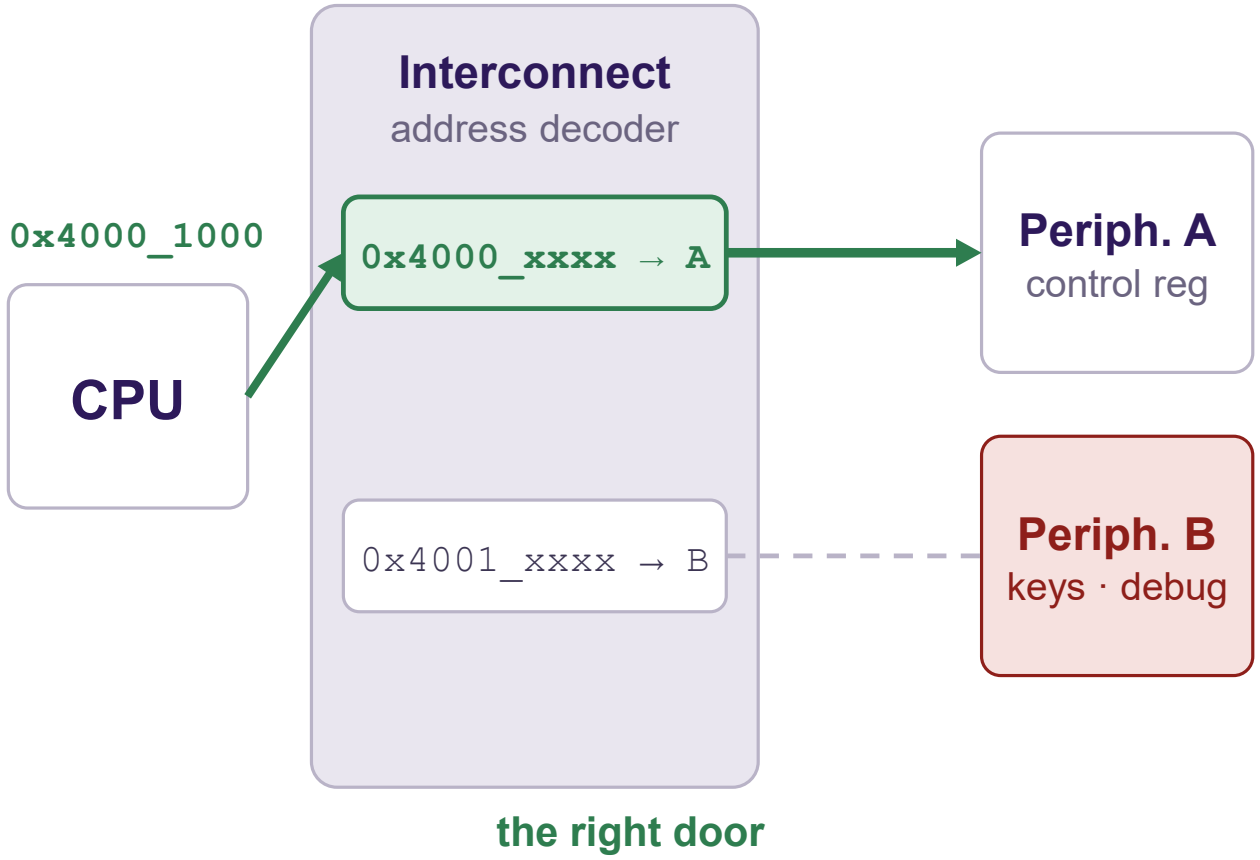
A Transaction Is a Message



Modify happens inside the CPU: nothing on the bus.

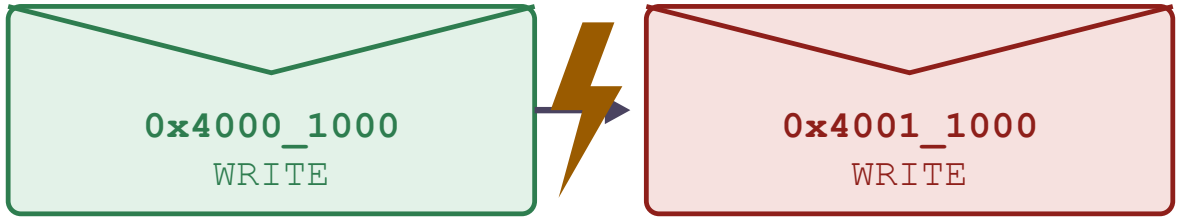
- **One Bus, Both Ways:** requests go out, responses come back.
- **What Travels:** where (address), what (read or write), the data, and the answer.
- **Which Bus:** AXI on our chip, because we use Arm cores. Others carry the same information.
- **The Point:** the hardware obeys the message, it never asks if the software meant it.

Two Peripherals, One Interconnect



- **Same Code:** it still writes to 0x40001000.
- **The Decoder:** the interconnect looks at the address, and only at the address.
- **Peripheral B:** the one you fear: a key register, or a debug enable.
- **One Bit Apart:** A and B differ only by bit 16.

One Bit Away



what the code sent *what the bus carried*

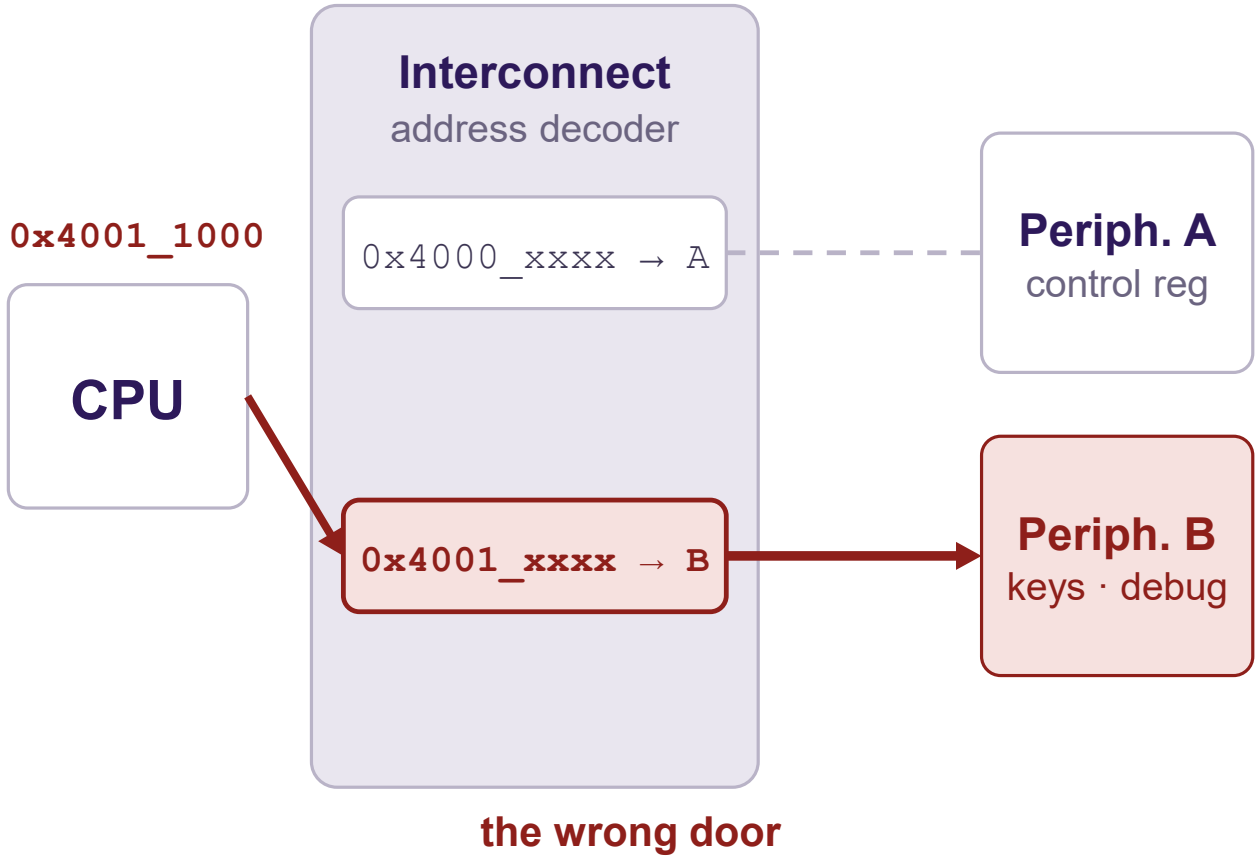
```
0x40001000 = 0100 0000 0000 0000 0001 …
0x40011000 = 0100 0000 0000 0001 0001 …
```

bit 16

- **The Cause:** a laser shot, a voltage glitch, or just a wild pointer.
- **The Effect:** the same code now writes to B.
- **The Detection:** none. The CPU, the bus and B all behaved correctly.
- **The Code:** still correct.

Every component did its job.

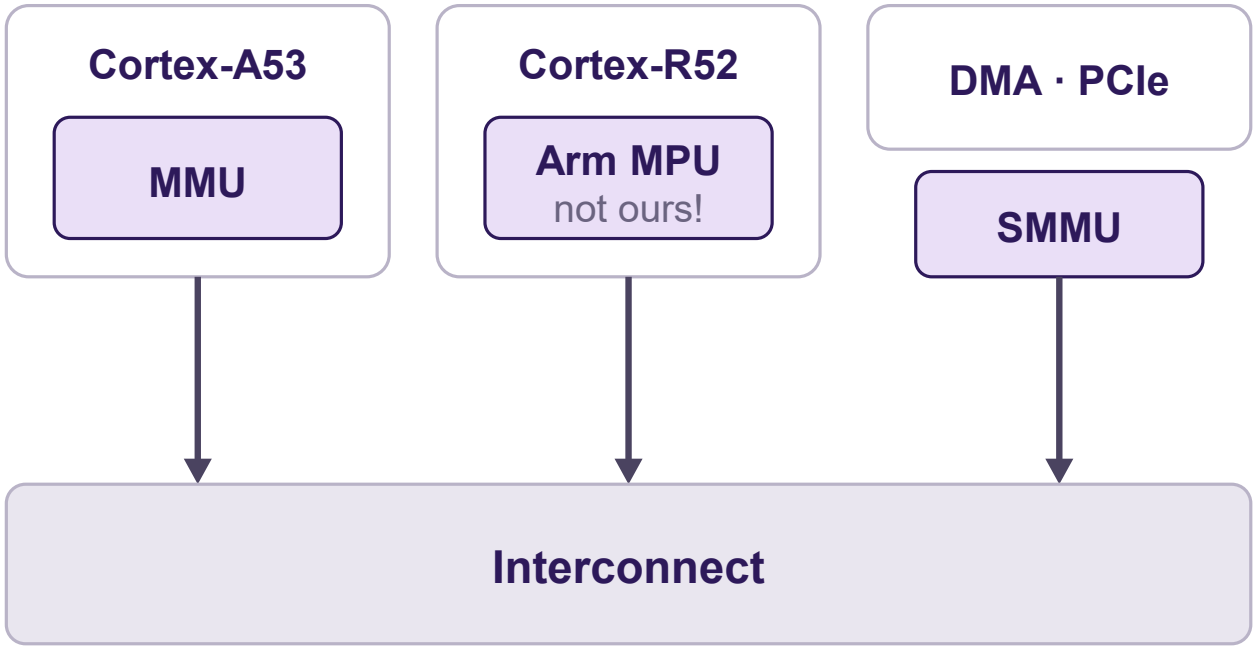
Same Code, Wrong Door



- **The Address:** 0x4001_1000, after the flip.
- **The Decoder:** does its job: bit 16 says B.
- **Peripheral B:** receives a write it was never meant to get.
- **No Alarm:** no error, no log, no interrupt.

The interconnect routes. It doesn't judge.

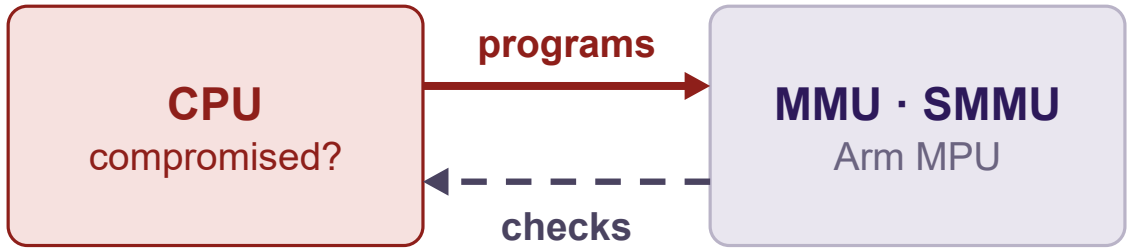
What Arm Already Gives You



On our Armv8 cores, all three are programmed by software on the CPU.

- **MMU:** inside the CPU: translates virtual addresses and checks permissions, page by page.
- **SMMU:** the same, in front of DMA masters such as PCIe or a DMA engine.
- **Arm MPU:** on the Cortex-R52: no translation, permissions per memory region.
- **Naming:** the Arm MPU is not our MPU. Naming is hard.
- **Who Sets Them:** privileged software, running on the CPU.
- **Scope:** our Armv8 cores. Armv9 adds RME (Realms), out of scope today.

Let the CPU Police Itself?

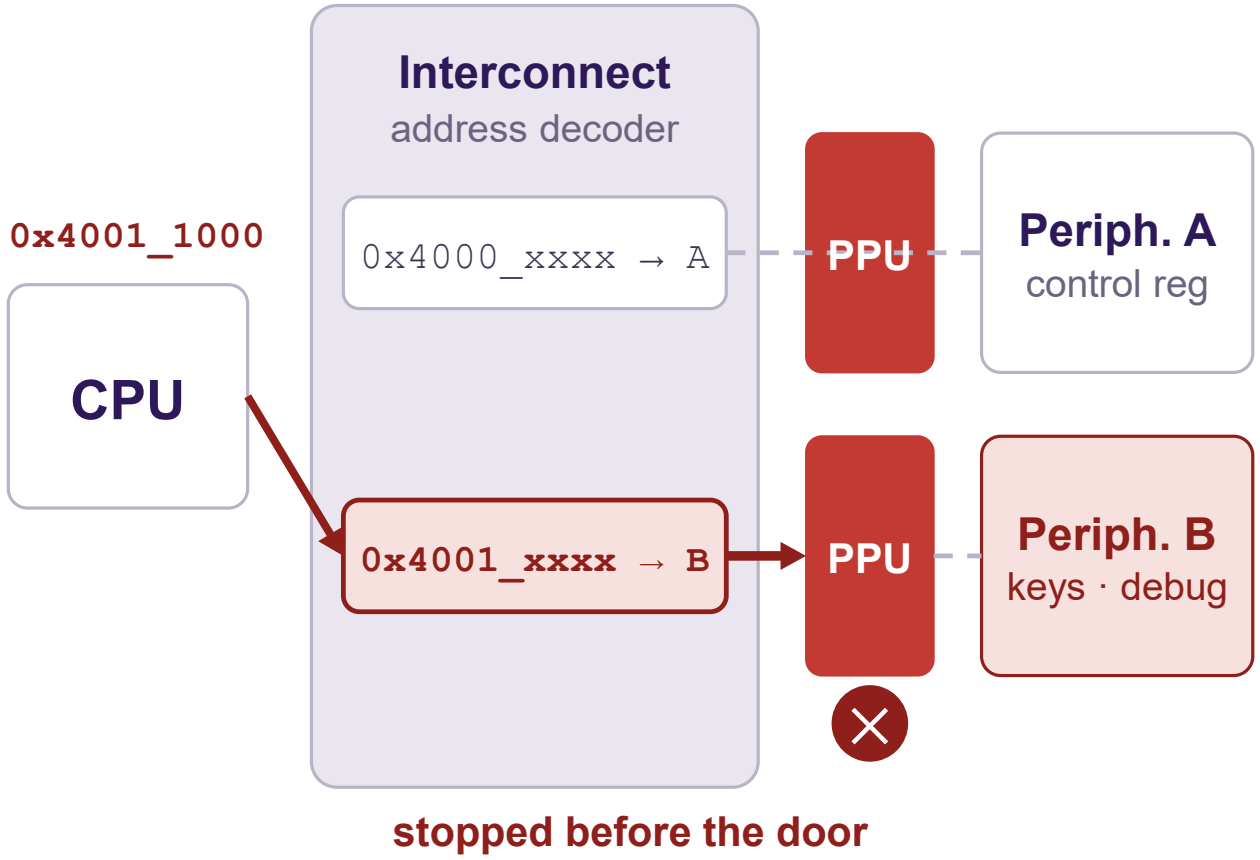


So who programs the guards?

Keep that question open.

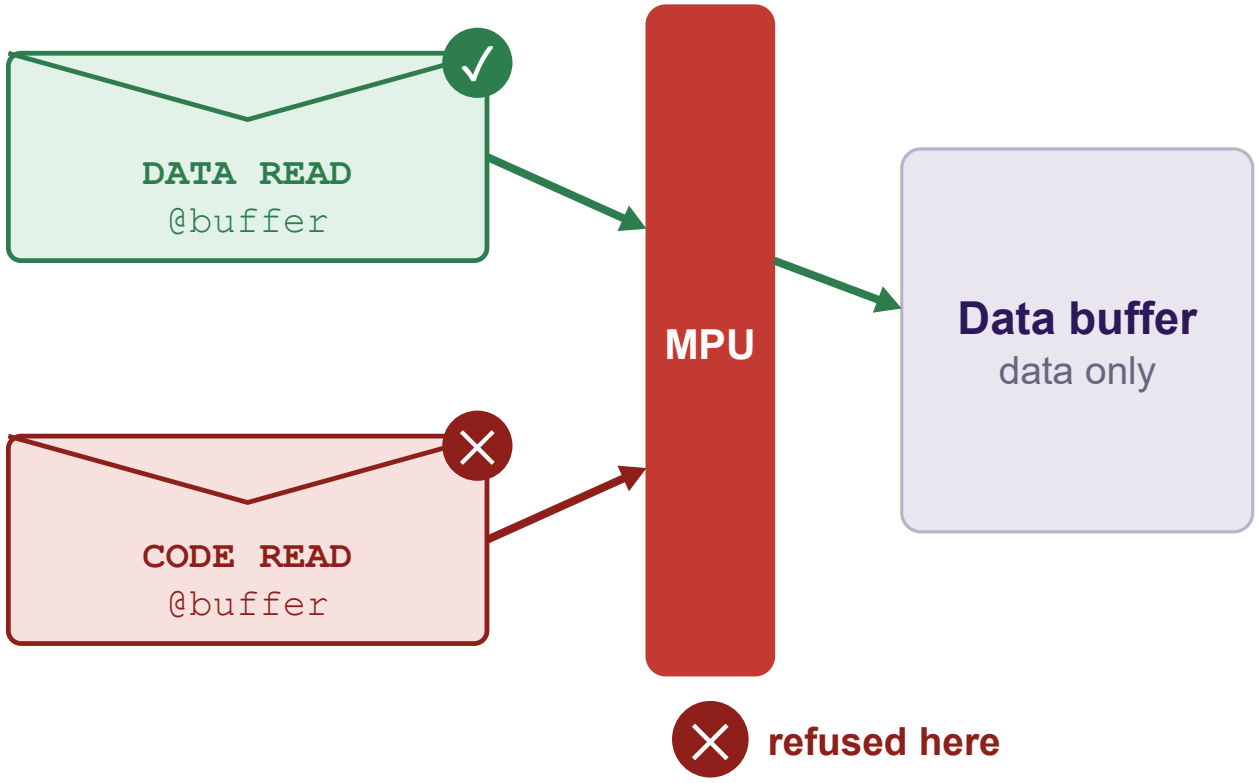
- **The Obvious Answer:** let the MMU, SMMU or MPU stop the wrong access.
- **The Catch:** they are programmed by the CPU we are trying to contain.
- **Still Useful:** great against bugs, but they can't be our root of trust when they're controlled by what we're trying to contain.

Filter on Where



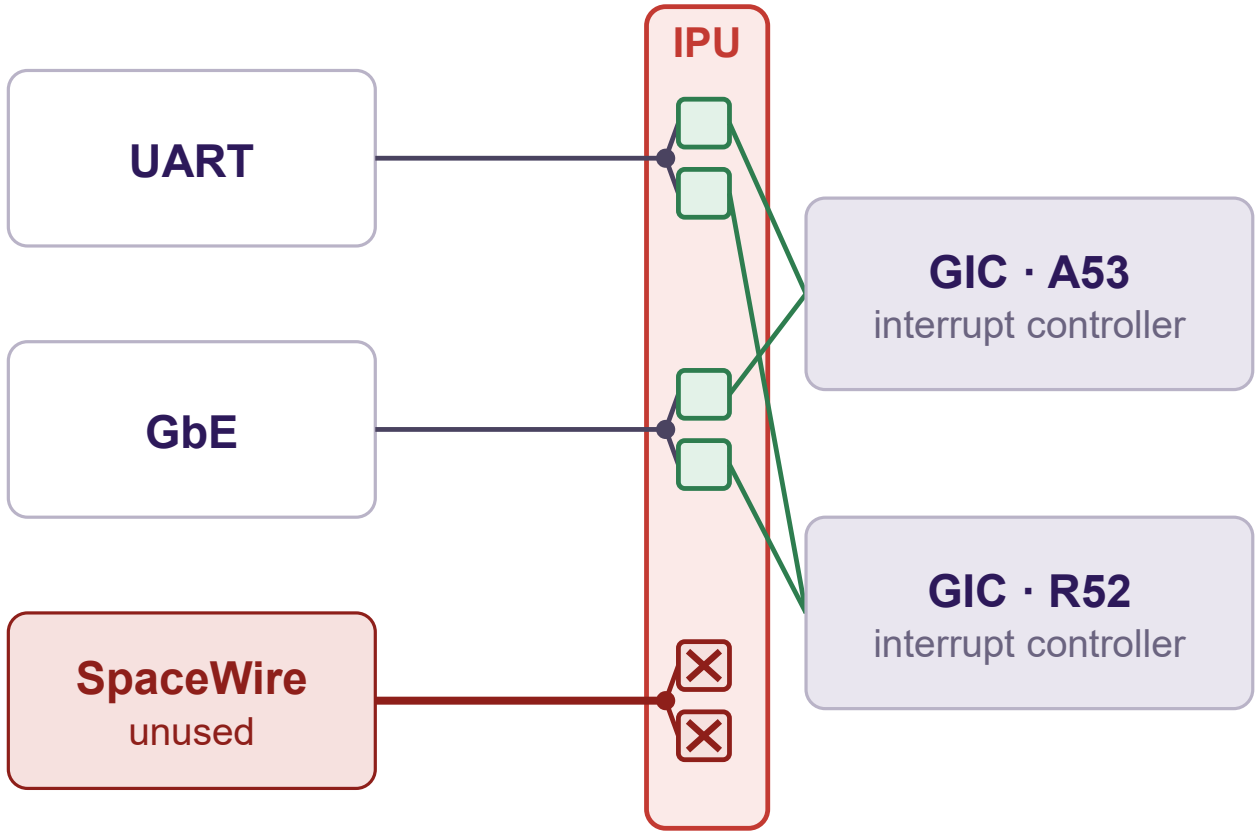
- **A Guard per Door:** a PPU in front of each peripheral, an MPU in front of each memory.
- **Rule for A:** read and write allowed.
- **Rule for B:** refused for this code.
- **Default Deny:** anything not in the rules is refused.
- **Who Is Asking:** for now, the rules don't know. We'll come back to it.

Filter on What



- **The Fault:** a bit flip makes the CPU jump into a data buffer.
- **The Address:** is fine: the buffer is a legitimate region.
- **The Bus Knows:** every read says if it is a code read (instruction fetch) or a data read.
- **The MPU:** this region holds data: it lets data reads through and refuses code reads.

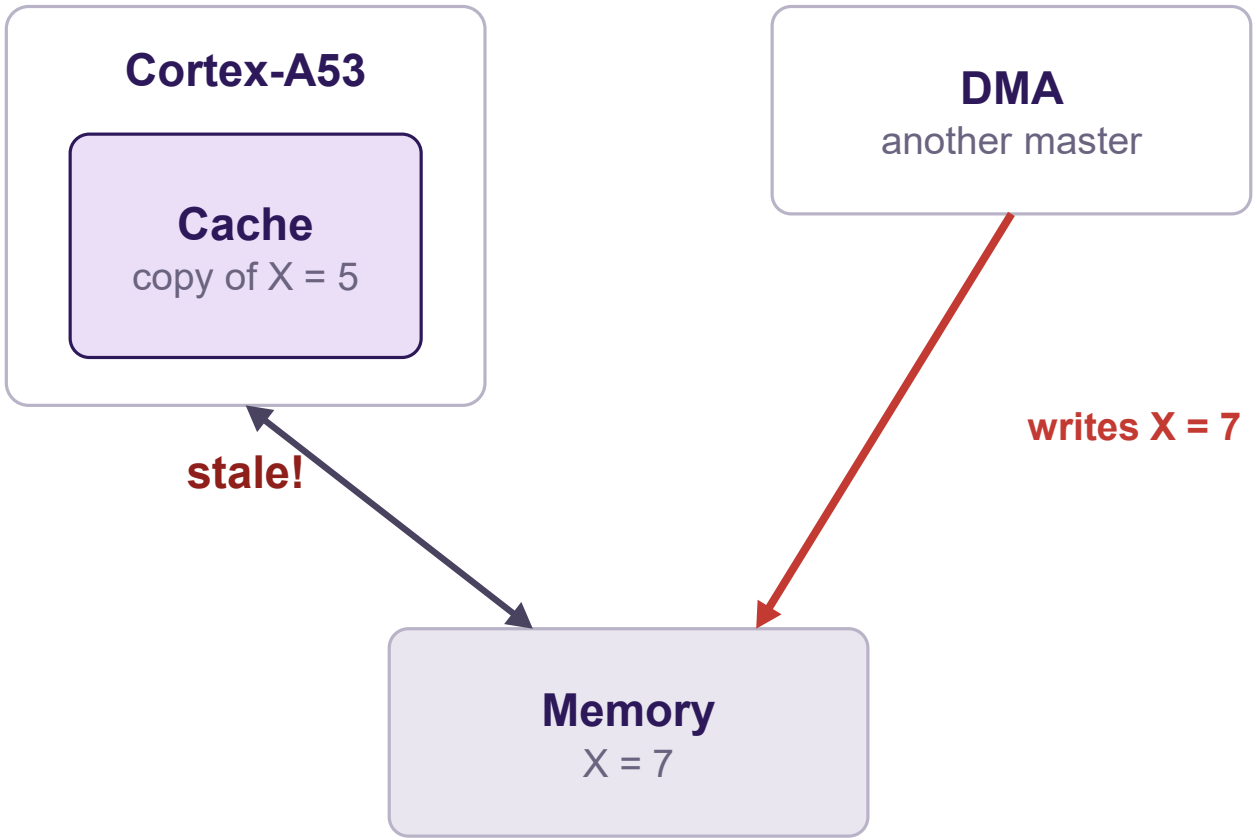
Wires (Interrupts) Too



IPU: one gate per line, per GIC

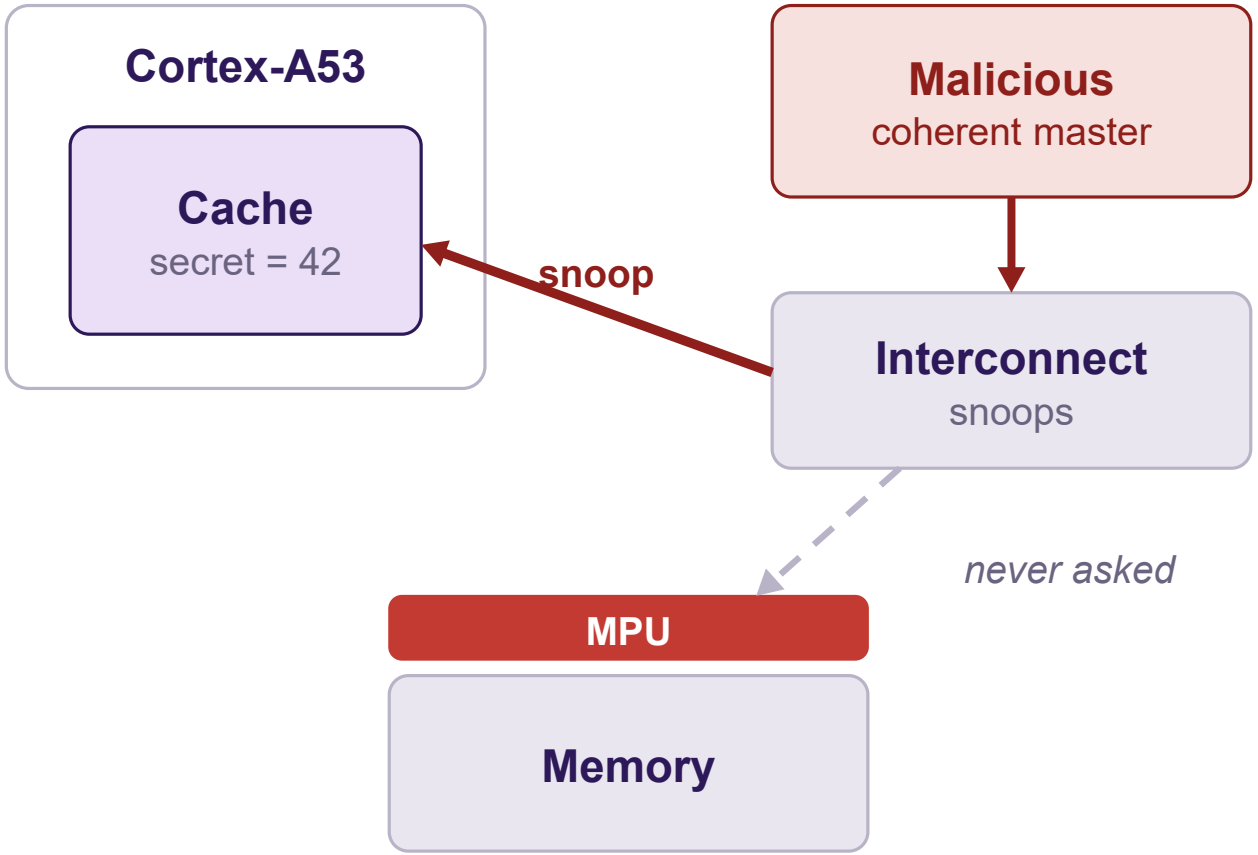
- **Legit:** each peripheral raises its interrupt on one wire, split inside the IPU.
- **The Storm:** an unused peripheral fires nonstop and drowns the CPU.
- **Not a Transaction:** a wire has no address and no identity: firewalls never see it.
- **IPU:** an IRQ gate on every line, for every interrupt controller (GIC), set by the policy.
- **MSI:** a message-signalled interrupt is a write, so it goes through the normal firewalls.

Caches: Fast, but a Copy



- **The Cache:** a copy of memory, right next to the CPU.
- **The DMA:** another master writes memory directly.
- **The Problem:** the copy is now stale.
- **The Fix:** coherency: the interconnect lets a master look into the CPU's cache, and act on it.

The Cache Is a Side Door



- **The Snoop:** a coherent master reads what sleeps in the A53 cache.
- **The Bypass:** the data never comes from memory, so it never meets the memory firewall.
- **Both Ways:** its requests also act on the victim's cache.
- **On Onysis:** only one snoopable cache: the A53's.

Convenience and exposure are the same feature.

Don’t Firewall the Coherency Protocol

Removing a master from the coherent domain

1. Stop the master from allocating shareable data into its cache, for example by disabling the data cache.
2. Clean and invalidate all shareable data from its caches.
3. Disable snoops and DVM messages to the master: hardware control, or the Snoop Control Register.
4. If you used the register: execute a barrier instruction.
5. Poll the Status Register until the change is effective.

…and five more steps to add it back.

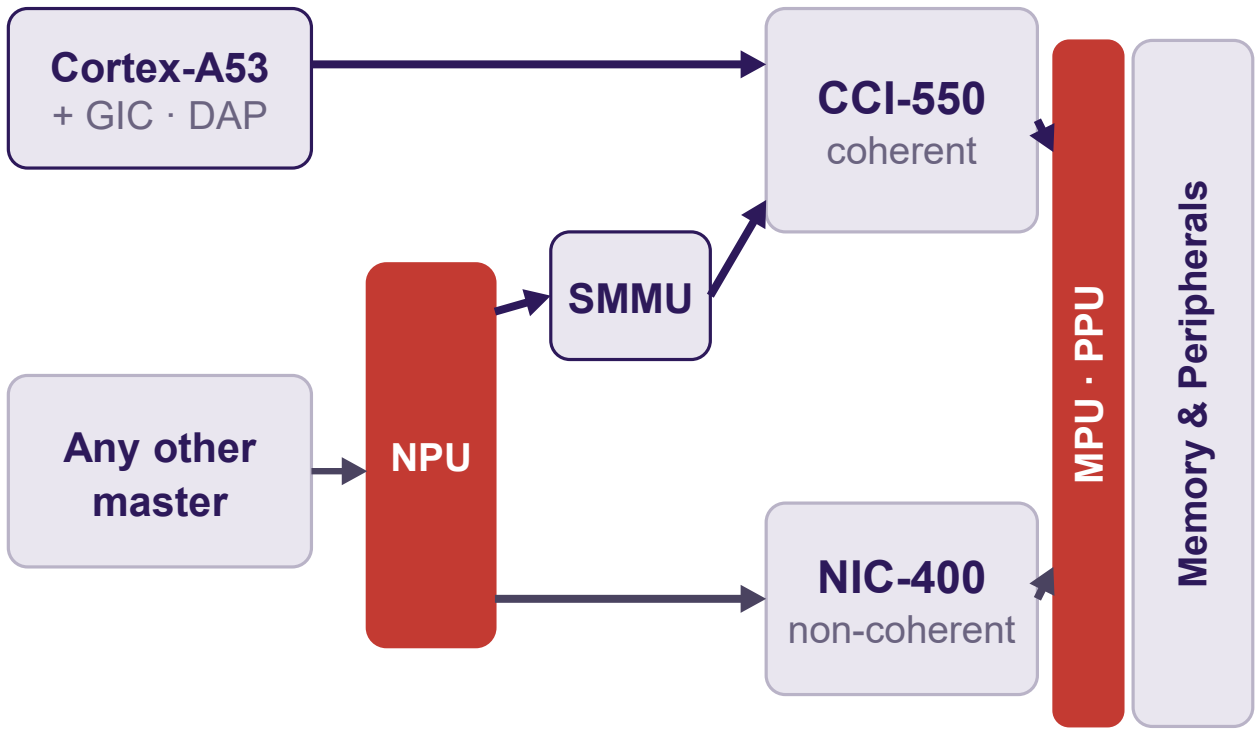
Source: Arm® CoreLink™ CCI-550 TRM, r1p0

…or just add a second interconnect.

- **Liveness:** a snoop that gets no answer can stall the whole system.
- **The Arm Way:** a documented sequence to leave a coherency domain.
- **Our Reaction:** trust software to run five steps, in order, every time? We laughed.
- **Our Way:** firewall the right to be coherent, not the protocol.

Silicon is cheaper than verification.

Two Roads to Memory

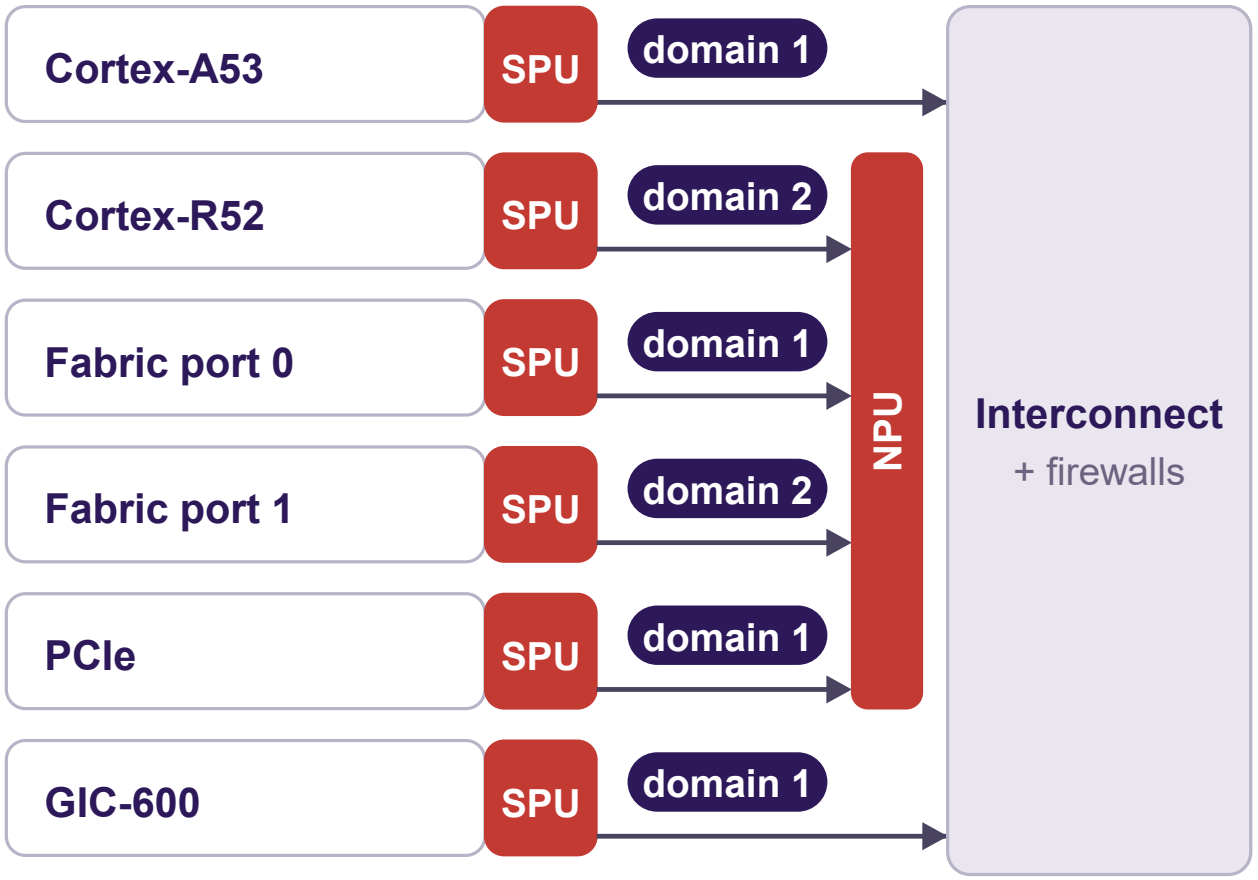


one buffer on both roads? you own the consistency

- **Why Coherent:** the A53 itself: once its MMU and caches are on, it sends DVM messages only a coherent interconnect can answer. On a non-coherent one: deadlock.
- **The SMMU Gate:** any master can become coherent, but only through the SMMU.
- **What It Buys:** once Linux enables it, a coherent master can only snoop the pages it is mapped to, and a hypervisor fits in easily.
- **Three Exceptions:** the A53, its interrupt controller, and the debugger.
- **Domain Attribute:** coherency is chosen per domain, at boot, by the policy.

Coherence is a trust decision, not just a performance one.

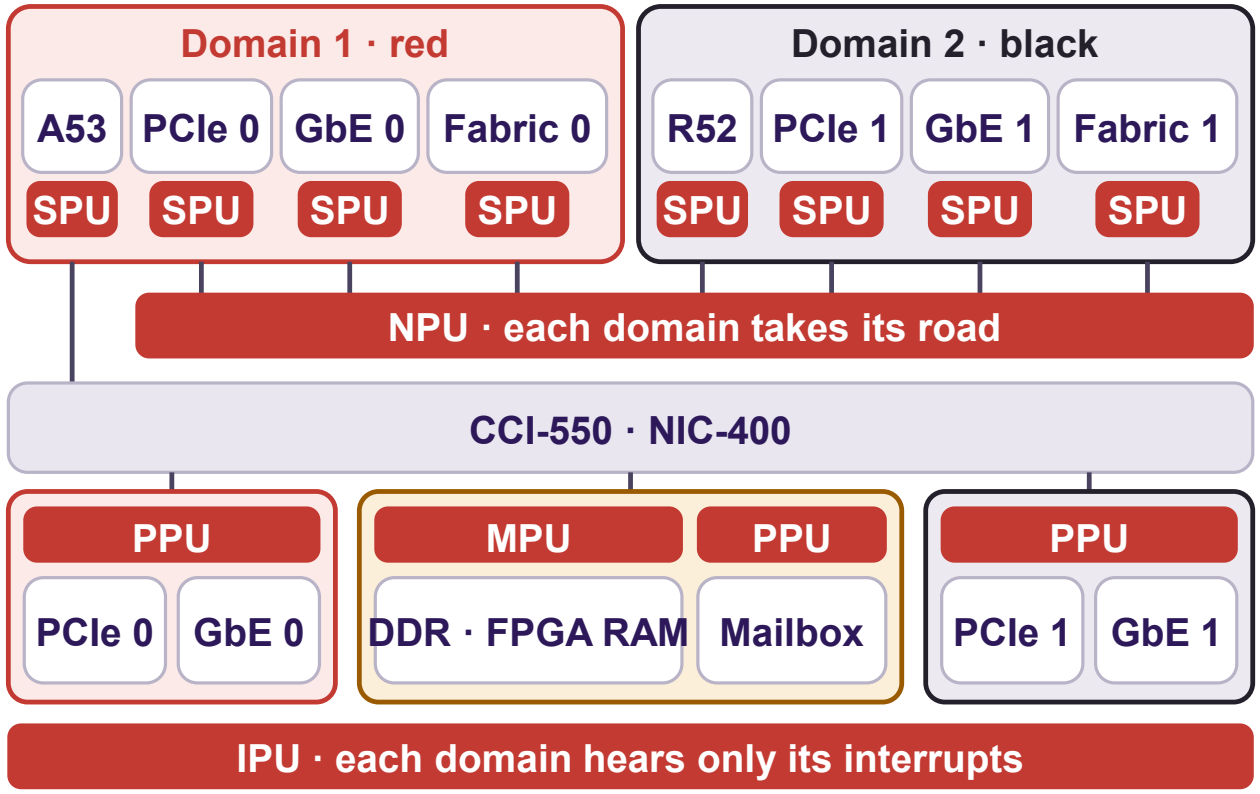
Who’s Asking? Give Every Master a Badge



- **The Problem:** we don’t modify third-party IP, and a master picks its own bus ID: not unique, and easy to forge.
- **SPU:** System Protection Unit: one block glued to every master, the same block everywhere.
- **The Badge:** it writes the domain into every transaction, overwriting what the IP sent.
- **Granularity:** one master port, one domain: our hardware containers. Nine to choose from.
- **Fail-safe:** an unconfigured SPU raises an error: the chip doesn’t boot.
- **TrustZone:** the SPU also sets the security state: pass, force Secure, force Non-secure.

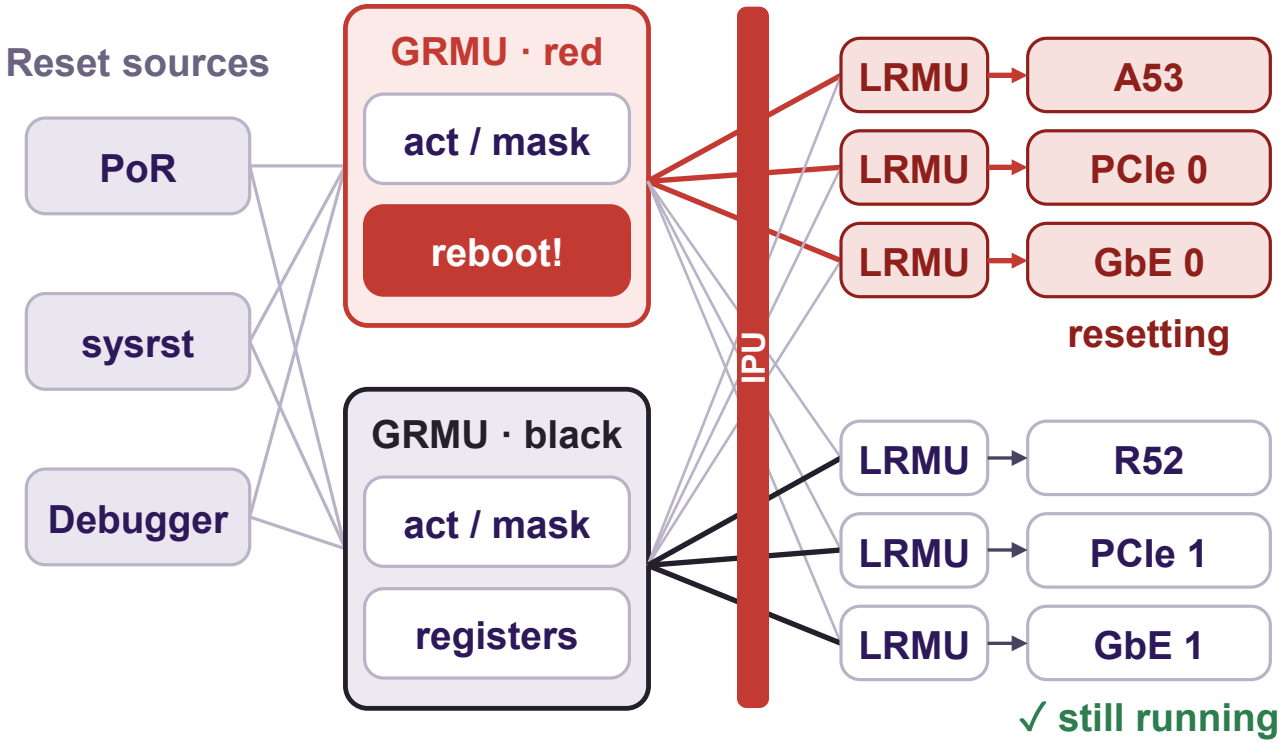
No IP gets to write its own badge. Not even PCIe.

Domains, Guarded



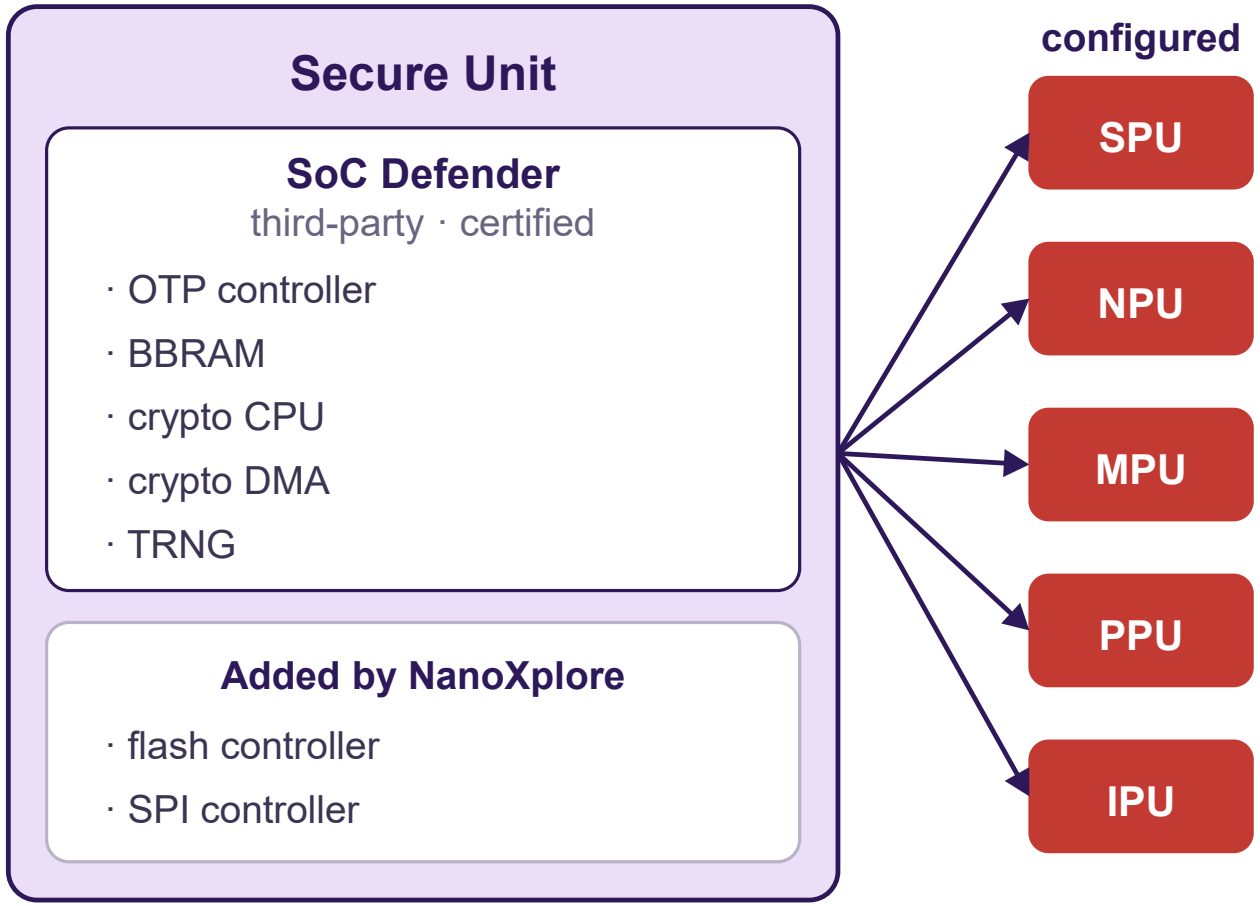
- **SPU:** stamps every transaction with its domain.
- **NPU:** sends each domain down its road, coherent or not.
- **MPU & PPU:** let each domain reach only its memories and peripherals.
- **IPU:** lets each domain hear only its own interrupts.
- **Shared:** only what the policy opened: here DDR, FPGA RAM and a mailbox, its interrupt reaching both sides.

Rebooting One Container



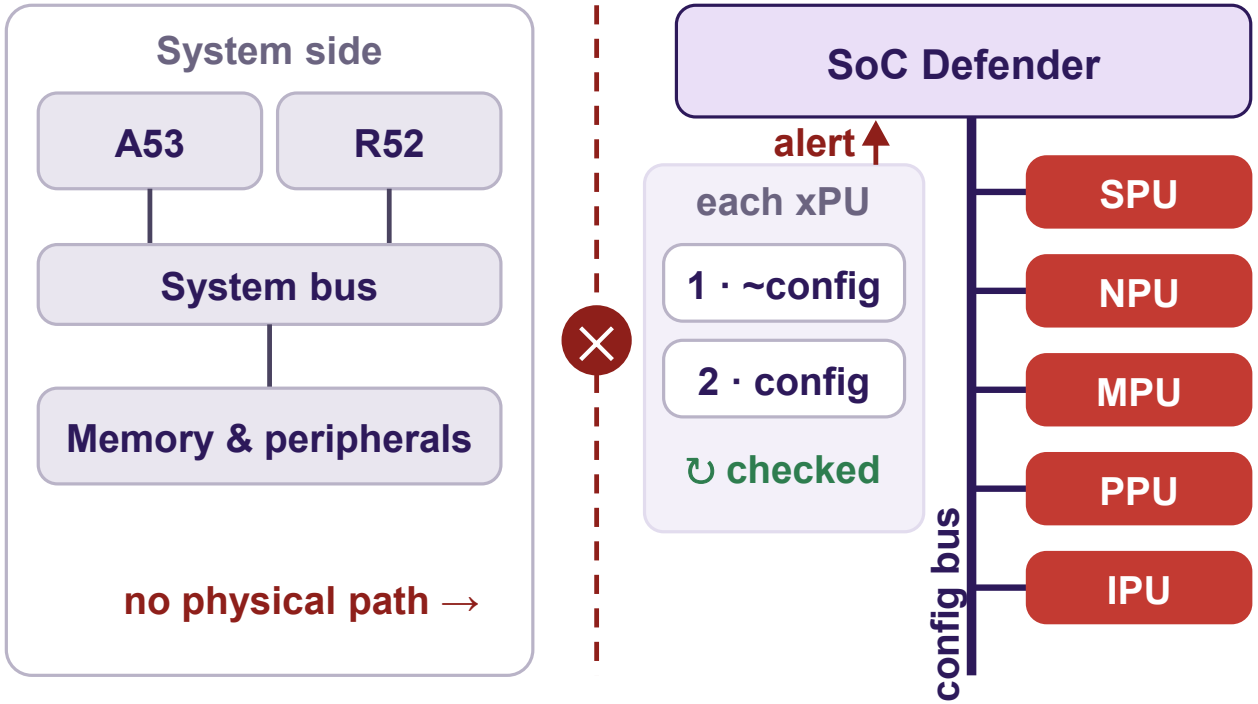
- **The Goal:** Linux on the red side reboots; the black side keeps running and never notices.
- **One Reset Manager per Domain:** nine GRMUs (Global Reset Management Units), one per domain.
- **Act or Mask:** each GRMU chooses which reset sources it obeys: power-on, system reset, debugger, its own registers.
- **Local Sequences:** one LRMU (Local Reset Management Unit) per block, running that block's reset sequence.
- **Any to Any:** every GRMU is wired to every LRMU; IPU gates decide which GRMU may drive which LRMU. Reuse, again.

Who Programs the Guards?



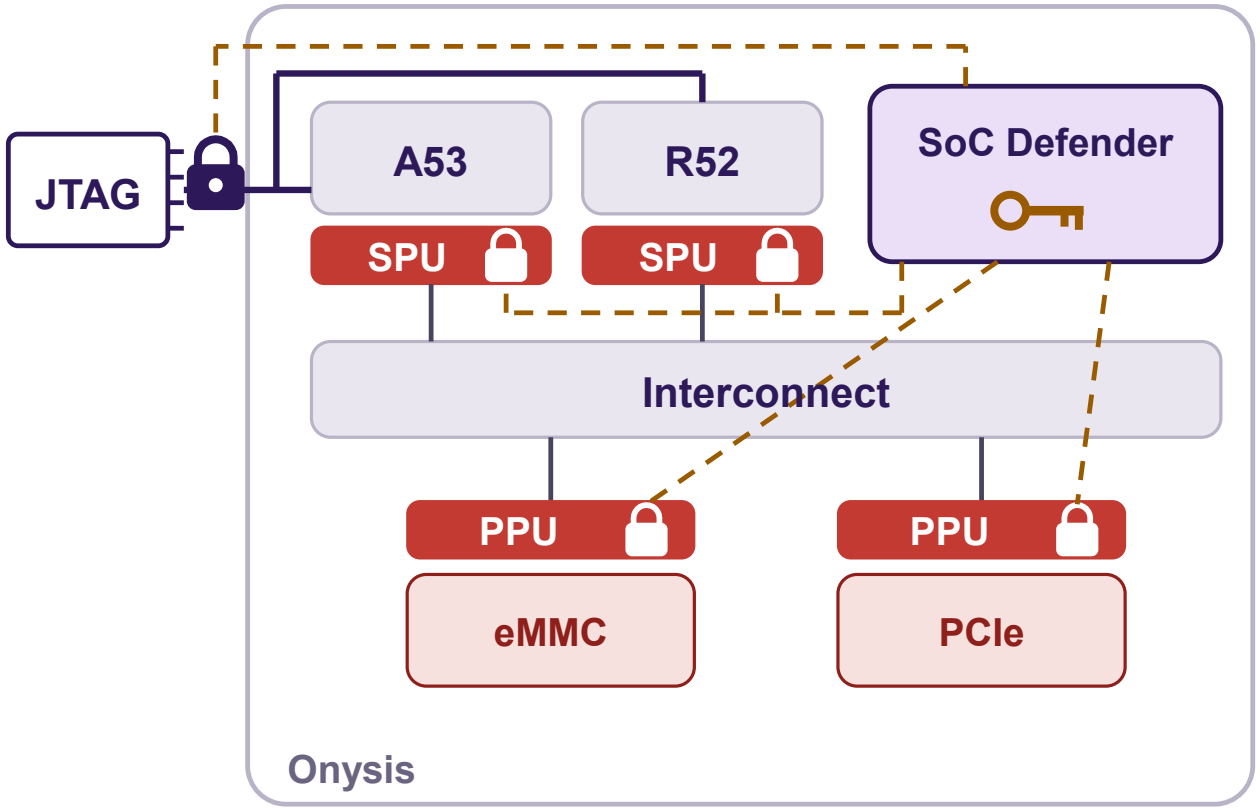
- **Not the CPU:** we just saw why.
- **SoC Defender:** a third-party security engine, certified Common Criteria, inside our Secure Unit.
- **Sole Configurator:** the only master allowed to configure the firewalls.
- **Boot-time Only:** configured at boot, then locked.
- **Self-protected:** behind its own firewall, with laser, temperature and voltage sensors.

Who Guards the Guards?



- **Own Bus:** xPUs are configured over a dedicated bus that shares nothing with the system bus. A compromised CPU has no physical path to them.
- **Written Twice:** every register is written as ~config, then config. A single fault can't produce a valid configuration.
- **Always Checked:** each xPU checks its configuration continuously; any mismatch raises a security alert, with the same graded reactions.

Closing the Back Doors



- **The Strongest Door:** one debug port sits directly on the coherent interconnect, with no SMMU: it sees exactly what the CPU sees.
- **Debug Under Control:** per CPU, per debug port, per CoreSight signal. Locked for good in OTP, or opened and closed by the security policy.
- **Scan:** in the user lifecycle, production scan and boundary scan are hardwired off.
- **Unused Masters & Peripherals:** their SPU or a hardware barrier cuts them off, set from OTP or by the policy. eMMC, PCIe, Ethernet, …
- **The One Exception:** peripherals start on; unused ones must be cut explicitly.

What you don’t use must be closed by configuration.

Five Guards, One Journey

Unit	Stands for	Sits	Checks
SPU	System Protection Unit	on every master	stamps the domain and the security state
NPU	Network Protection Unit	behind every SPU but the A53's and the GIC's	picks the coherent or the non-coherent road
MPU	Memory Protection Unit	in front of memories	who, where, read/write, fetch/data, security state
PPU	Peripheral Protection Unit	in front of peripherals	the same, for peripherals
IPU	Interrupt Protection Unit	on every interrupt wire	one gate per line, per controller

On violation: continue, reset, try the next boot image, jump to the BIOS interface (more later), or erase secrets and lock the product. Each one with or without a journal entry.

Naming things is the second hardest problem in computer science.

What Firewalls Don’t Do

Smaller surface, not empty.

- **Who, Not Why:** they check who accesses what, not when or why.
- **Confused Deputy:** a legitimate master, once hijacked, keeps its rights.
- **Integration:** expose a critical block to PCIe, and nothing stops it.
- **Timing Channels:** through shared resources: not something I’m claiming to address today.

NanoXplore

So how do you configure all of this?

Domains, roads, guards, gates, debug: someone has to write it down, once.

One Policy, Written Once

SPU	per master: domain, security state
NPU	per domain: coherent or non-coherent road
MPU	per memory region: who, read / write, code / data, security state
PPU	per peripheral: who, read / write, security state
IPU	per interrupt line: open or closed, for each GIC

+ debug access. One entry per guard, nothing else.

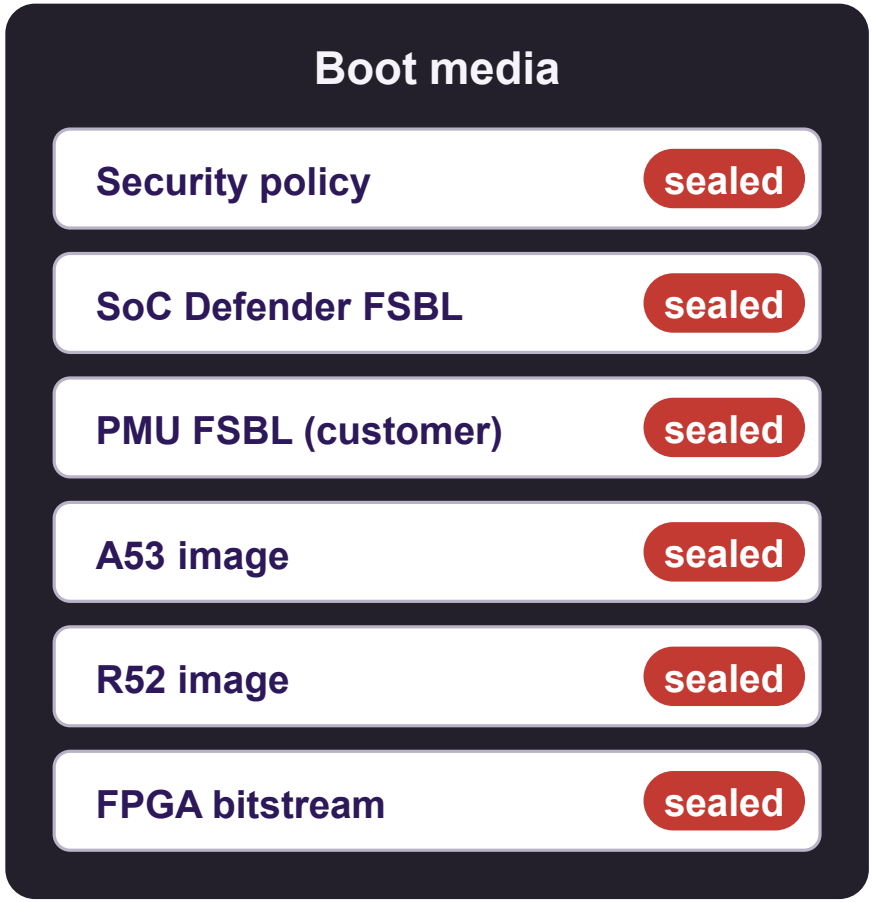
- **Author:** the product owner.
- **Content:** the configuration of every protection unit, and debug access.
- **Storage:** in the boot media (a NOR flash, for instance), encrypted and authenticated.
- **Loading:** by SoC Defender, before any customer code runs.
- **Default Deny:** the firewalls block anything not written.
- **The FSBL:** the customer's first-stage boot loader must match its own policy.

NanoXplore

How does it get into the chip?

Code, bitstream, policy, keys: everything lives outside the chip first.

Assume the Boot Media Is Hostile

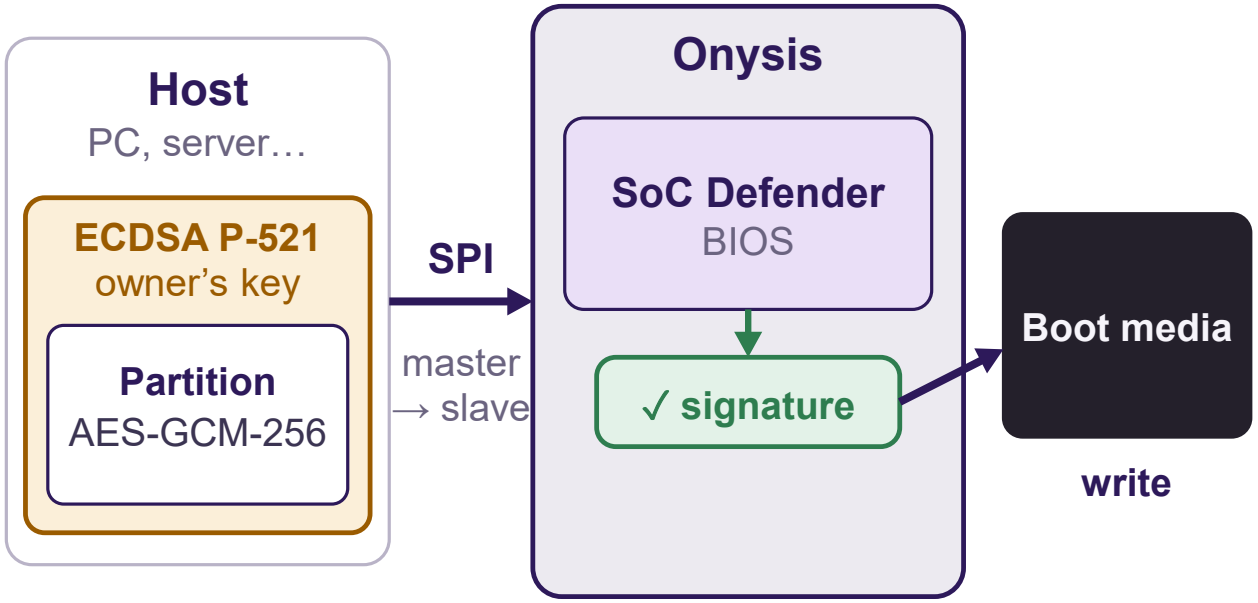


AES-GCM-256
storage:
secret + intact

ECDSA P-521
injection:
signed sender

- **The Threat:** the boot media sits outside the chip: it can be read, replaced or desoldered.
- **Partitions:** everything is a partition, the FPGA bitstream included.
- **Storage:** AES-GCM-256: what sits in the boot media stays confidential and intact.
- **Injection:** ECDSA P-521 with SHA-512: binaries sent to the product to be written must be signed.
- **Hardwired:** in silicon, not updatable: a design choice.
- **One More Layer:** one boot media can't even be moved to another chip. More on that in a moment.

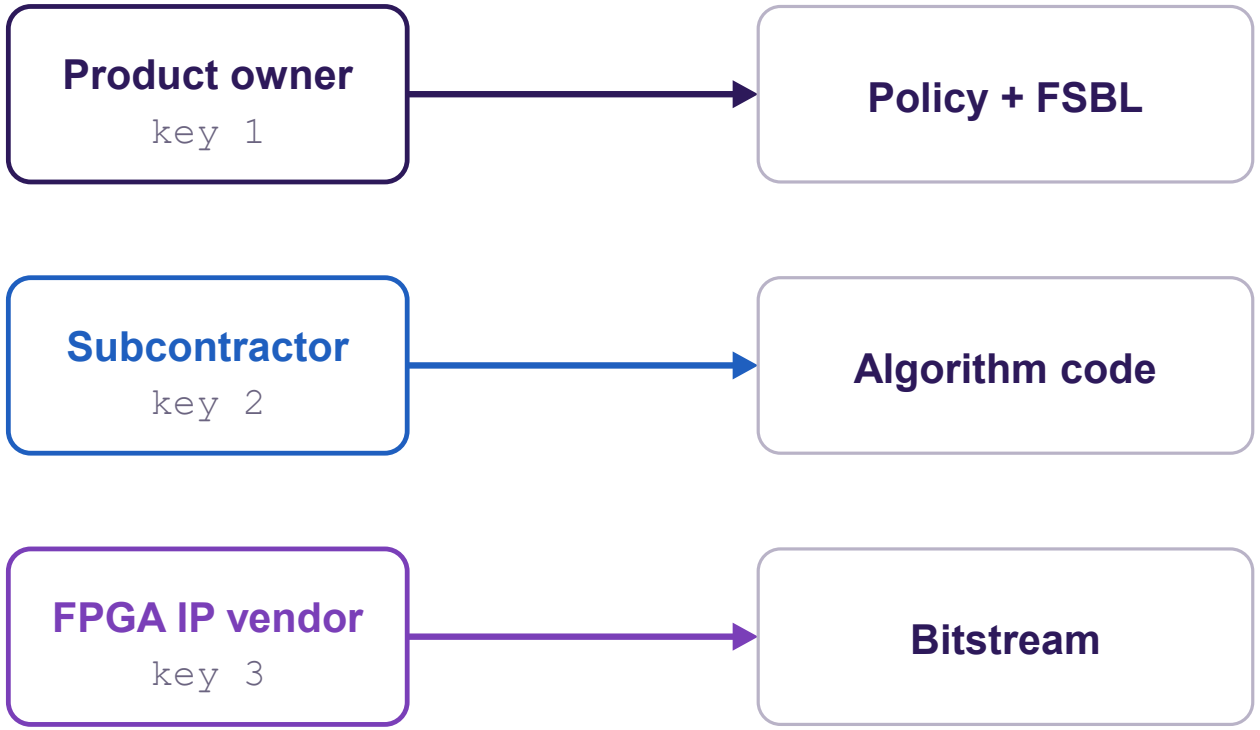
First Injection, Over SPI



Every owner signs their own partitions.

- **One Shot:** after production and test, the chip is ready but empty: the BIOS accepts binaries over SPI, once. Later writes go through the update services.
- **Host as Master:** a PC or a server drives the SPI link; the chip is the slave.
- **Two Layers:** each partition is already encrypted and authenticated (AES-GCM-256); for the transfer, it is also signed with ECDSA P-521.
- **One Key per Owner:** every partition owner signs with their own key.
- **Checked, Then Written:** SoC Defender verifies each signature before writing to the boot media.

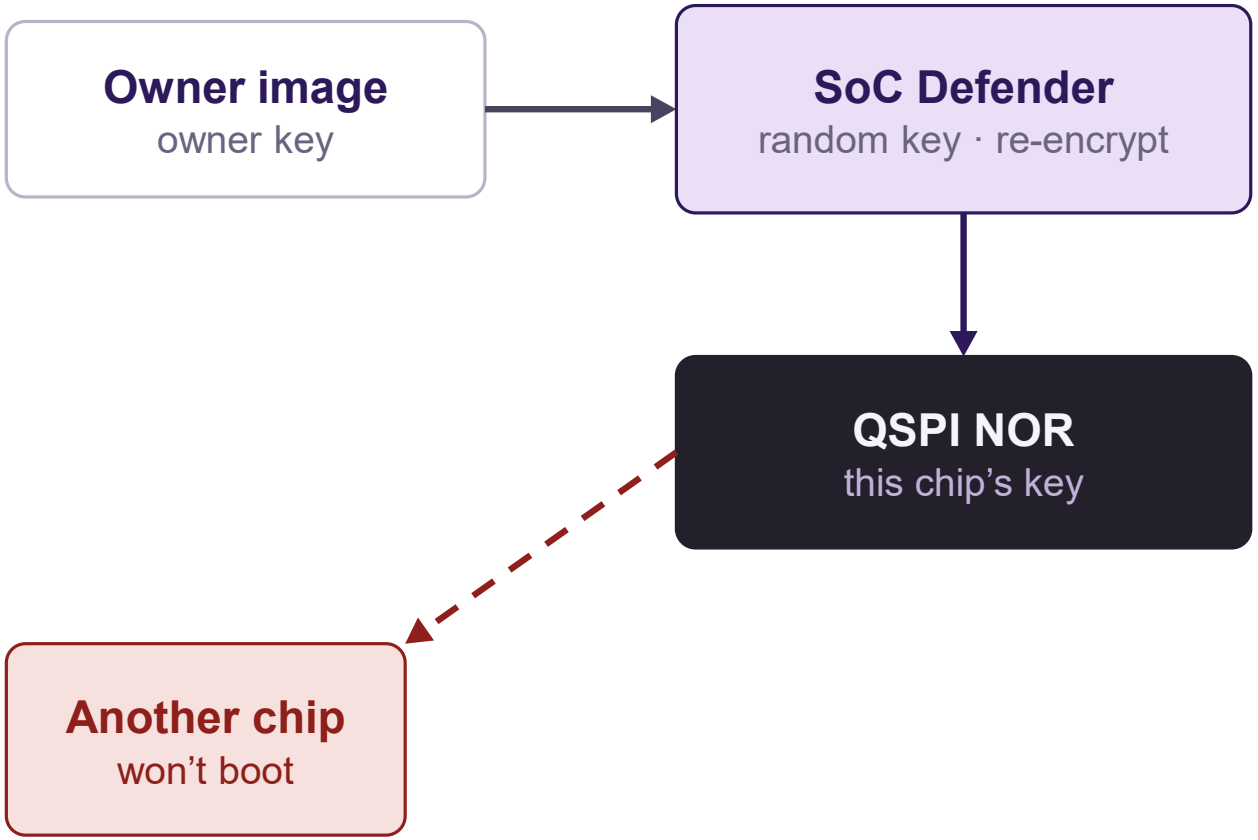
Many Key Holders, One Chip



up to 9 key sets, one per domain

- **Independent Keys:** up to 9 key sets, one per domain.
- **Subcontractor:** encrypts its own code with its own key. Only the product can decrypt it.
- **Personalization:** keys injected in a secure facility, at a trusted third party.
- **Trade-off:** trust moves to that facility.
- **Honest Limit:** the product owner writes the policy, so it can dump the subcontractor's code.
- **Storage:** keys live in OTP or BBRAM, depending on the product's configuration; never in flash.

Device Binding: This Flash, This Chip



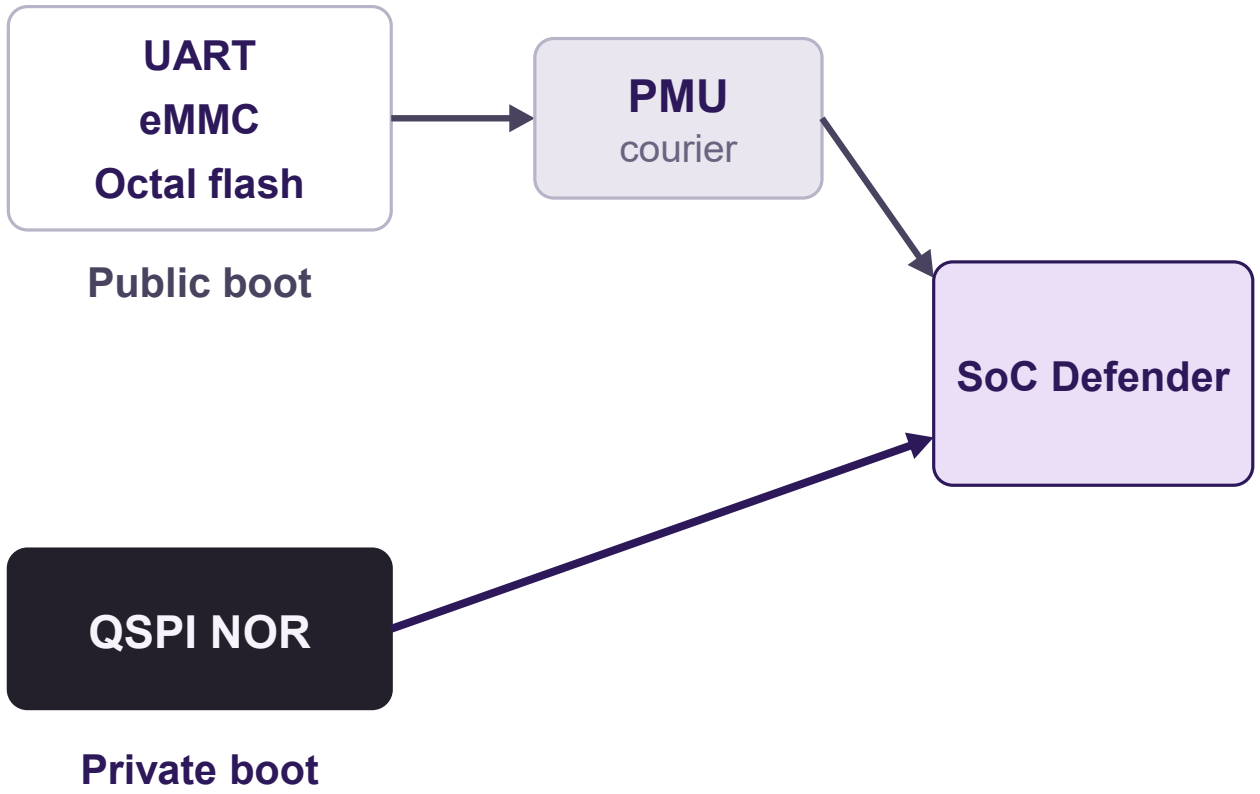
- **Private Boot:** QSPI NOR, read directly by SoC Defender.
- **First Boot:** SoC Defender draws a random key, then re-encrypts every file with it before writing it back.
- **Result:** desolder the flash and put it on another chip: it won't boot.

NanoXplore

And how does it boot?

Two ways in, one order that never changes.

Two Ways In



- **Public Boot:** UART, eMMC or Octal flash, read by the PMU.
- **The PMU:** a courier: it carries data it can neither read nor modify unnoticed.
- **Private Boot:** QSPI NOR only, read directly by SoC Defender, with device binding.
- **Same Crypto:** AES-GCM-256 on both paths.

The Boot Sequence

1. SoC Defender ROM
2. PMU ROM, acknowledges SD
3. **Policy applied and locked by SoC Defender**
4. SoC Defender FSBL
5. PMU FSBL, written by the customer
6. Partitions loaded via SD services, by the PMU FSBL
7. SoC Defender firmware, only if needed, by the PMU FSBL
8. PMU firmware, only if needed
9. PMU releases the resets

■ **ROM** ■ **SoC Defender** ■ **PMU** ■ **customer's PMU FSBL** The FPGA bitstream is just another partition.

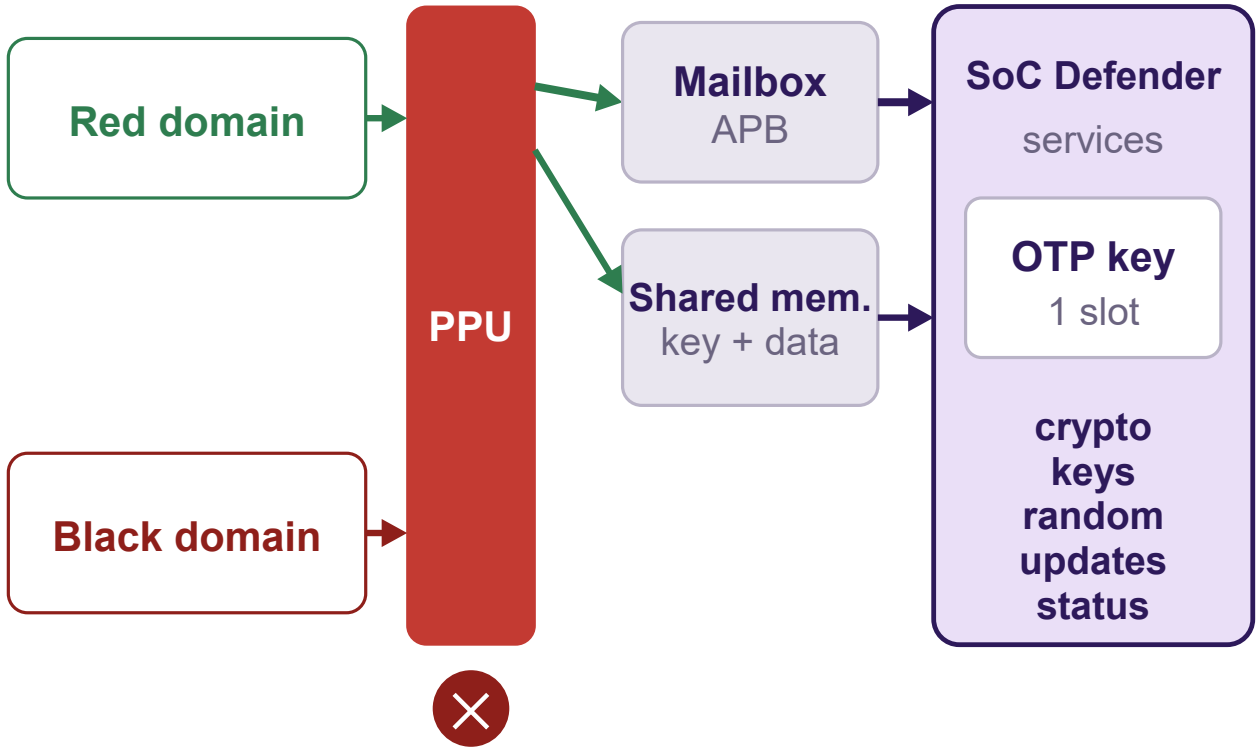
PART 5 · AT RUN TIME

NanoXplore

And once it runs?

Keys to use, and a chip that watches itself.

Crypto Services: Two Ways to Hold a Key



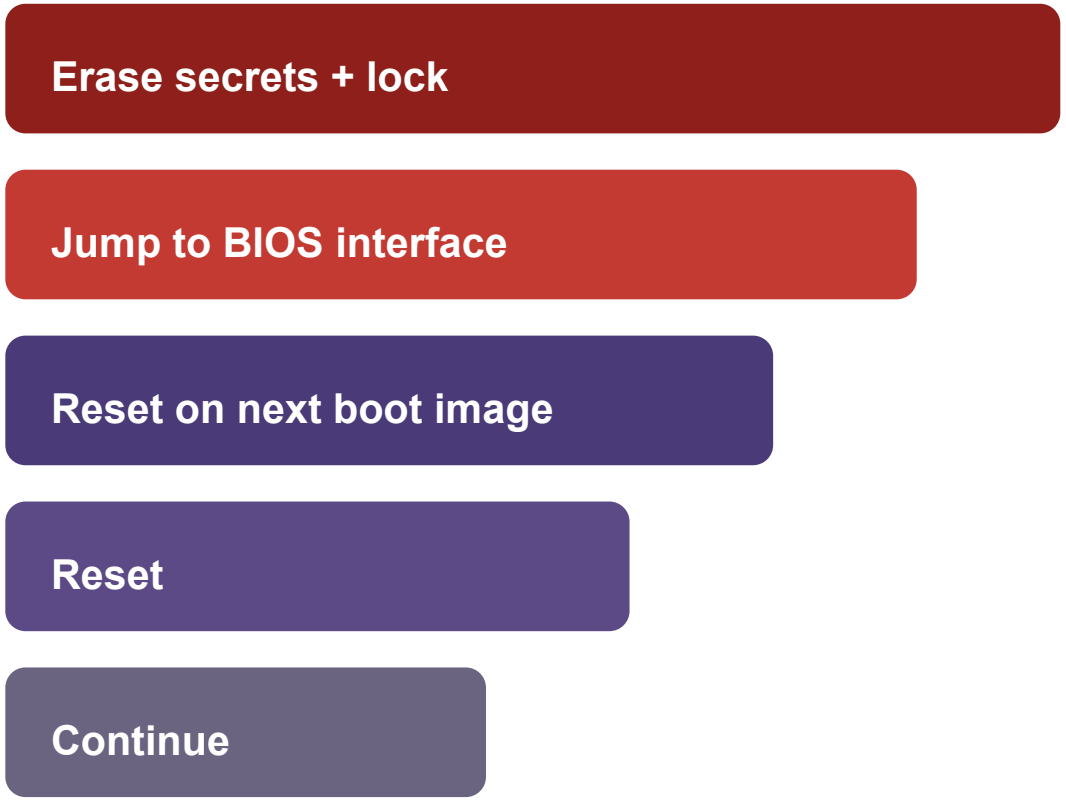
Mailbox access = right to use the services

- **Two Key Modes:** the caller passes its own key in shared memory, with the data; or SoC Defender uses a key from OTP.
- **One OTP Slot:** a single OTP key slot is usable by these services.
- **In Silicon:** a crypto CPU and a crypto DMA, with AES-GCM-256, ECDSA P-521 and SHA-512 hardwired.
- **Services:** symmetric, public-key, AEAD, hash, MAC, signature; key generation, derivation, exchange; DRNG and PTRNG random.
- **Tailored:** key injection and provisioning, binary updates in the field, status detection, reaction to errors.
- **Access:** through a mailbox: whoever the PPU lets in may call the services.

No new mechanism: the guards you already know.

Even When It's Off

each one with or without a journal entry



- **Always-On:** sensors stay powered while the chip is off.
- **Alarms:** up to 7 inputs, intrusion detectors included, detected at the next boot.
- **Reaction:** the same five as for any security event, chosen by the user, stored in OTP or BBRAM.
- **Zeroization:** keys destroyed, OTP included, except the log key.
- **The Log:** in the boot media, encrypted and authenticated.
- **BIOS Interface:** a minimal SPI interface, every command authenticated by an ECDSA challenge-response; it can read the journal.

A bricked chip can still tell you why it died.

How We Verified It

Simulation
reduced testbench
~2 hours

Palladium
full SoC
~3 min

Secure boot, end to end (including launch)

- **UVM:** block by block.
- **Emulation:** the full SoC, running the real C code of the embedded software team.
- **Real Partitions:** built with the user tools, then booted end to end.
- **Alarms:** triggered on purpose, to check that they fire.
- **Complementary:** each method finds different bugs.
- **Bonus:** embedded software joins chip verification: more coverage without hiring.

How We Choose

1 Coherency

a second interconnect, not a dynamic exit sequence.

2 Partitioning

configured at boot, then locked.

3 Faults

hardened islands and armored wires, not a hardened chip.

Simple, static, verifiable beats clever and dynamic.

If we can’t verify it before tape-out, we don’t build it.

Questions?

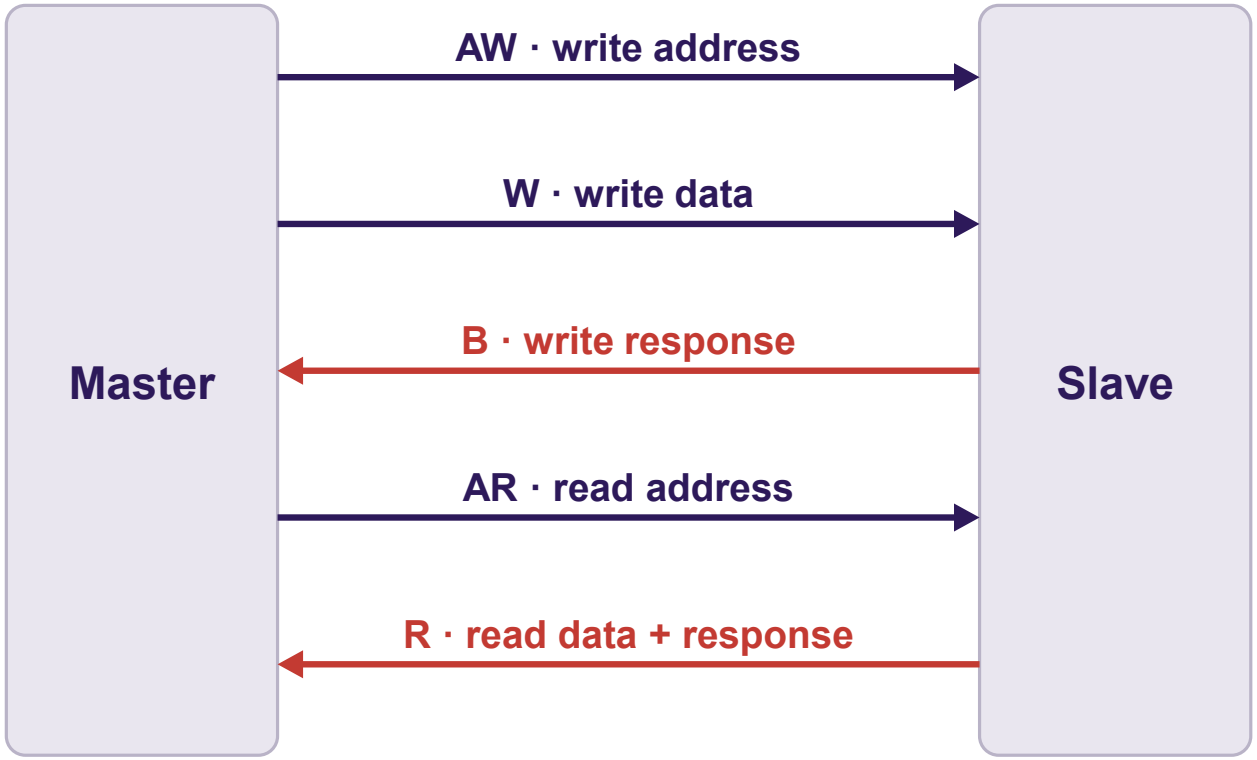


Adrien Grassein · NanoXplore

Cryptographic Building Blocks

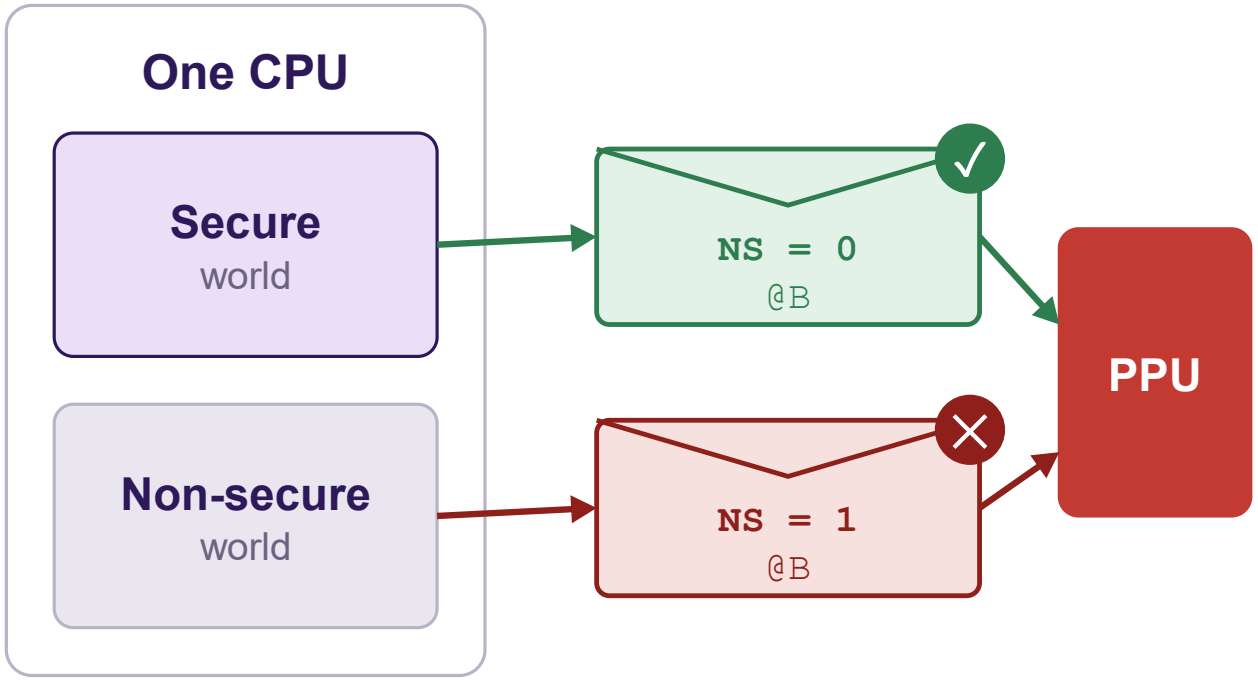
Block	Gives	Meaning
ECDSA P-521 + SHA-512	**injection**	binaries sent to the product must be signed; the private key never sits on the chip
AES-GCM-256	**storage**	nobody reads it, nobody changes it unnoticed
Hardwired	**not updatable**	agility is decided per product generation

AXI in Detail



- **Five Channels:** write address, write data, write response, read address, read data.
- **Handshake:** VALID and READY on every channel.
- **Attributes:** protection, cache and user fields travel with the address.
- **Independent:** reads and writes progress in parallel.

Arm® TrustZone®: One CPU, Two Worlds



The security state travels with every transaction.

- **The Need:** the CPU must reach B, just not from any code.
- **Privilege Level:** user, kernel or hypervisor: what the code may do on the CPU.
- **Security State:** Secure or Non-secure: which world the code belongs to. That is TrustZone.
- **On the Bus:** the security state travels with every transaction.
- **The Firewalls:** MPU and PPU rules now include it.

Who Is Secure? Whoever the Policy Says

TrustZone security state (NS bit)

Pass	keep what the IP says
Force 0	always Secure
Force 1	always Non-secure

…and the PPU rules can finally say who:

Who	Where	R/W
red domain	Periph. A	✓
red · Secure	Periph. B	✓
anyone else	Periph. B	✗

- **The Risk:** if each IP decides its own TrustZone security state, any master can claim Secure.
- **The SPU:** passes it, or forces it to TrustZone Secure or Non-secure, as the policy says.
- **Bonus:** the R52 has no TrustZone, yet its SPU can make its transactions Secure on the bus.
- **Result:** TrustZone becomes a property of the path, not of the IP.

No IP gets to write its own badge. Not even PCIe.

Where Secrets Live

	BBRAM	OTP
Rewritable	yes	no
Needs a battery	yes	no
Re-provisioning	possible	a slot is lost

- **Two Stores:** BBRAM or OTP, interchangeable: anything in one can live in the other, per product.
- **Configurable:** almost everything is the user's choice.
- **Anti-rollback:** exists, so the number of updates is bounded.
- **Never in Flash:** provisioned keys and alarm reactions stay on-chip.
- **User OTP:** separate from SoC Defender, the SoC also has user OTP where the product can store its own keys.