

From Reset To main(): Porting Infineon TriCore AURIX To Zephyr

Parthiban, Linumiz & Christoph Seitz, Infineon Technologies AG



Who are we

Parthiban, Linumiz

- 15+ years Embedded Software
- Linux kernel, U-Boot, Zephyr RTOS, recently NuttX
- Active upstream contributor: U-Boot, kernel, Zephyr
- Frequent OSS / Embedded Open Source Summit speaker

Christoph Seitz, Infineon Technologies AG

- Field Application Engineer, automotive EMEA
- FPGA high-speed data platforms, now Ethernet-based automotive solutions
- Ethernet, TSN, CAN; hardware-software interaction and industry standards
- Driving Zephyr for AURIX adoption in automotive

Agenda

- **Framing**

why TriCore, why Zephyr

- **Reset to first thread**

bootflow, linker, kernel init

- **Runtime arch services**

CSA context switch, interrupts, traps

- **Beyond the arch layer**

SoC, board, QEMU

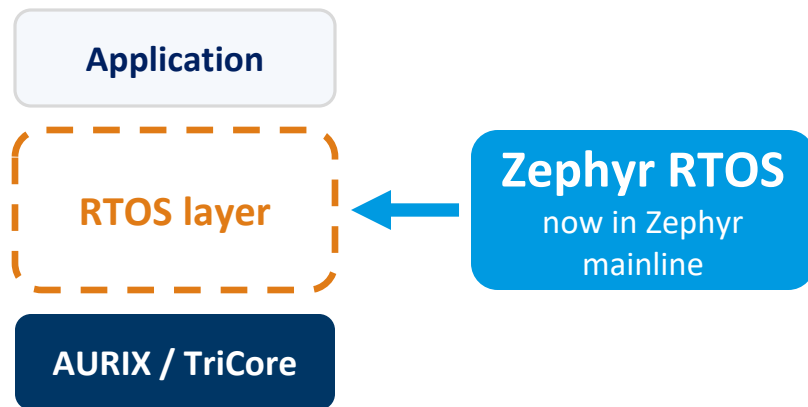
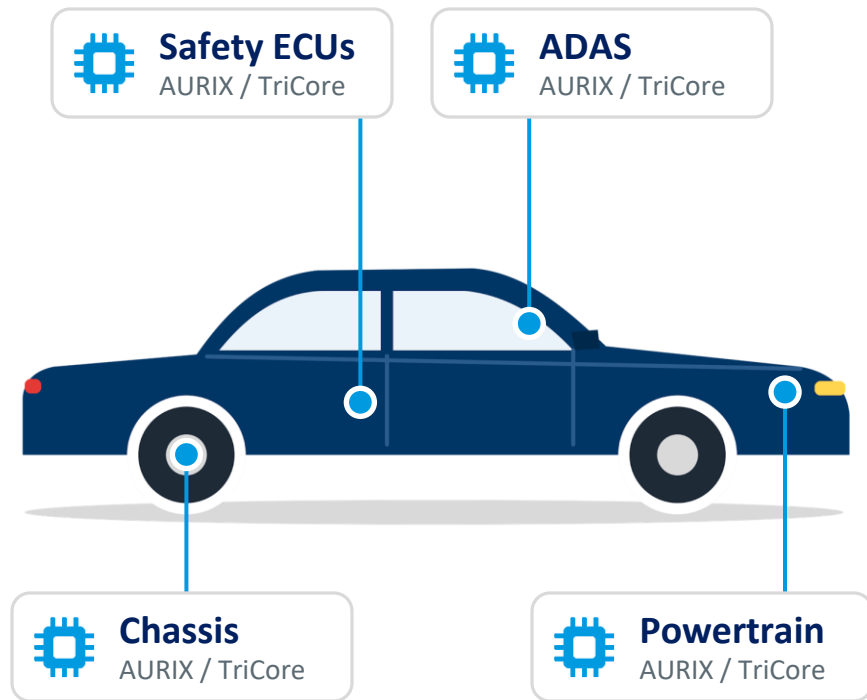
- **Upstreaming challenges**

HAL, review, what's next

Framing

why TriCore, why Zephyr

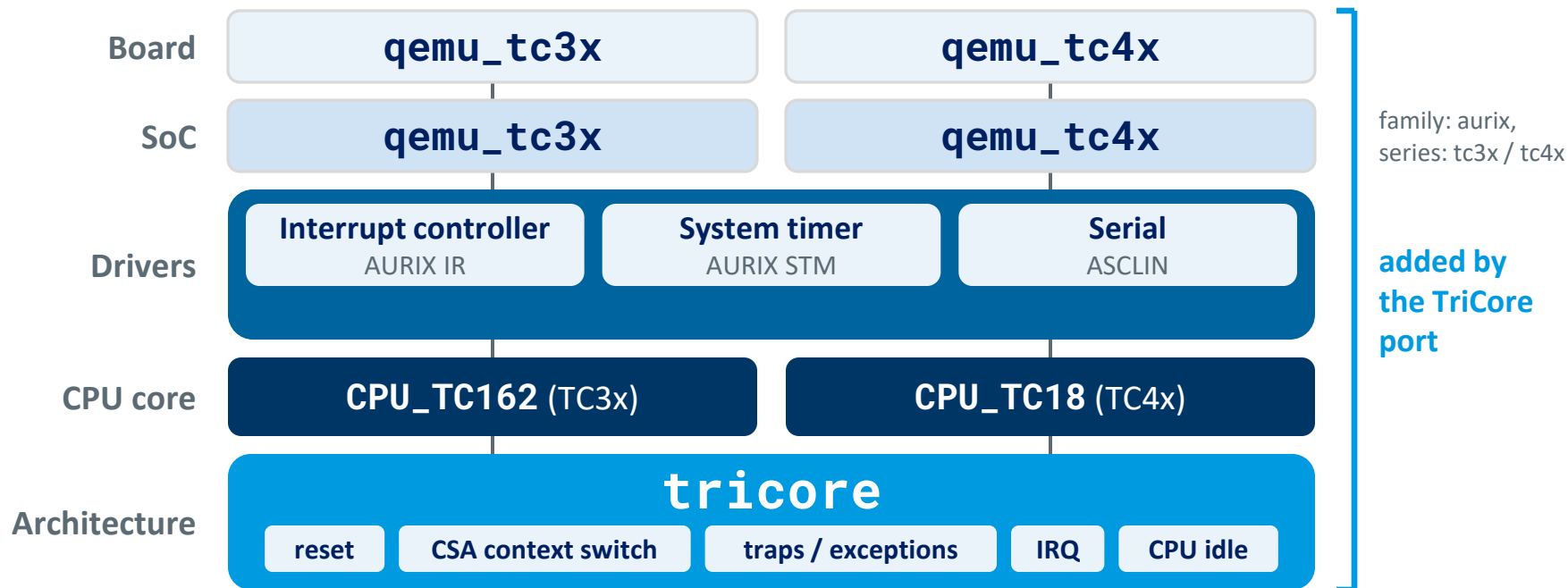
Why TriCore, why Zephyr



The upstream effort

zephyrproject-rtos/zephyr [PR #107516](#) - merged

QEMU-first; silicon boards downstream



TriCore vs ARM vs RISC-V

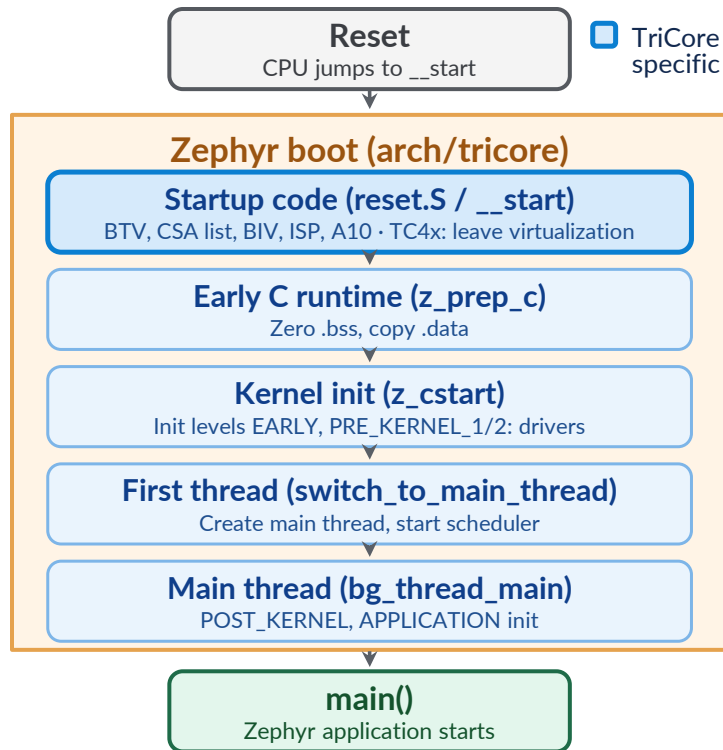
Concept	ARM Cortex-M	RISC-V	TriCore
Context save on exception	HW pushes 8 regs to stack (basic frame)	None in HW, software saves	HW saves upper context into a CSA (not the stack)
Vector table entry holds	An address (VTOR)	One instruction, usually a jump (mtvec vectored)	Code: a 32-byte slot per entry
Trap vs interrupt tables	One table	One base (mtvec)	Two bases: BTV (8 trap classes) + BIV (per priority)
Interrupt stack	MSP / PSP, HW switched	One SP, software swap (mscratch)	ISP: HW switches A10 on entry (if PSW.IS = 0)
Interrupt controller	NVIC	PLIC / CLIC / AIA	IR + Service Request Nodes

TriCore keeps context off the stack, and its vector tables hold code, not addresses.

Reset to first thread

bootflow, linker, kernel init

Bootflow and the linker script



```
MEMORY {  
    ROM      (rx) : ORIGIN = 0x80000000, LENGTH = 4M  
    DSRAM0   (rw) : ORIGIN = 0x70000000, LENGTH = 240K  
}  
  
ENTRY("__start")  
  
SECTIONS {  
    .reset : {  
        KEEP(*(.reset.*))  
    } > ROM  
  
    .vectors.trap : {  
        . = ALIGN(256);  
        __btv = .;  
        KEEP(*(.vectors.trap*))  
    } > ROM  
  
    .vectors.irq : {  
        . = ALIGN(8192);  
        __biv = .;  
        KEEP(*(.vectors.irq*))  
    } > ROM  
}
```

Arch init (reset.S)

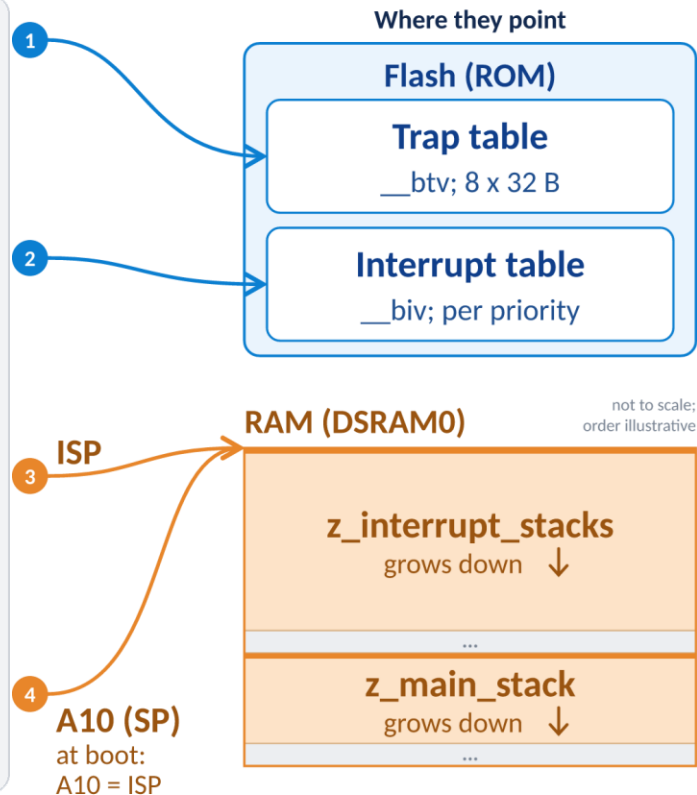
- 1

```
/* Set base trap vector table pointer */  
mov    %d10, lo:__btv  
addih  %d10, %d10, hi:__btv  
mcr    TRICORE_BTV, %d10
```
- 2

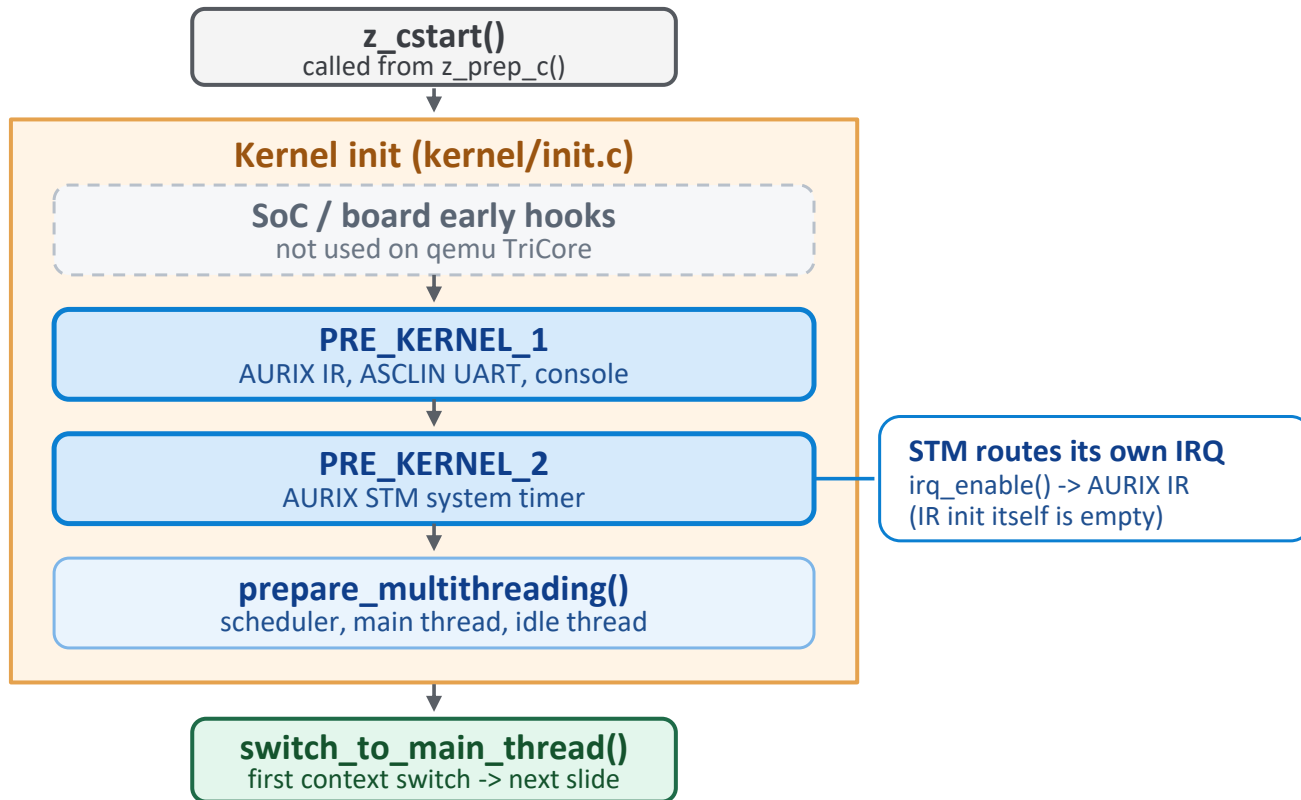
```
/* Set base interrupt vector table pointer */  
mov    %d15, lo:__biv  
addih  %d15, %d15, hi:__biv  
mcr    TRICORE_BIV, %d15
```
- 3

```
/* Set interrupt stack pointer */  
mov    %d15, lo:(z_interrupt_stacks + CONFIG_ISR_STACK_SIZE)  
addih  %d15, %d15, hi:(z_interrupt_stacks + CONFIG_ISR_STACK_SIZE)  
mcr    TRICORE_ISP, %d15
```
- 4

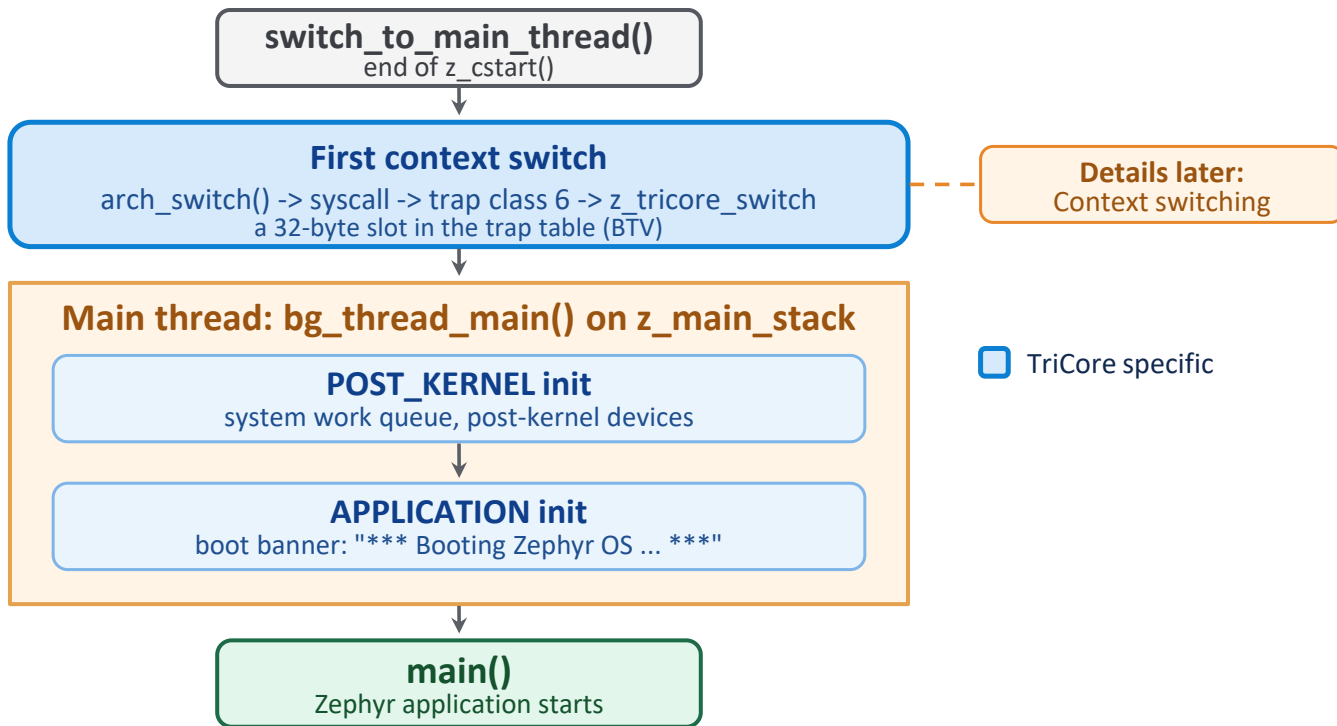
```
/* Set main stack pointer */  
movh.a %a10, hi:(z_interrupt_stacks + CONFIG_ISR_STACK_SIZE)  
lea    %a10, [%a10], lo:(z_interrupt_stacks + CONFIG_ISR_STACK_SIZE)
```



z_cstart(): kernel init



From kernel init to main()



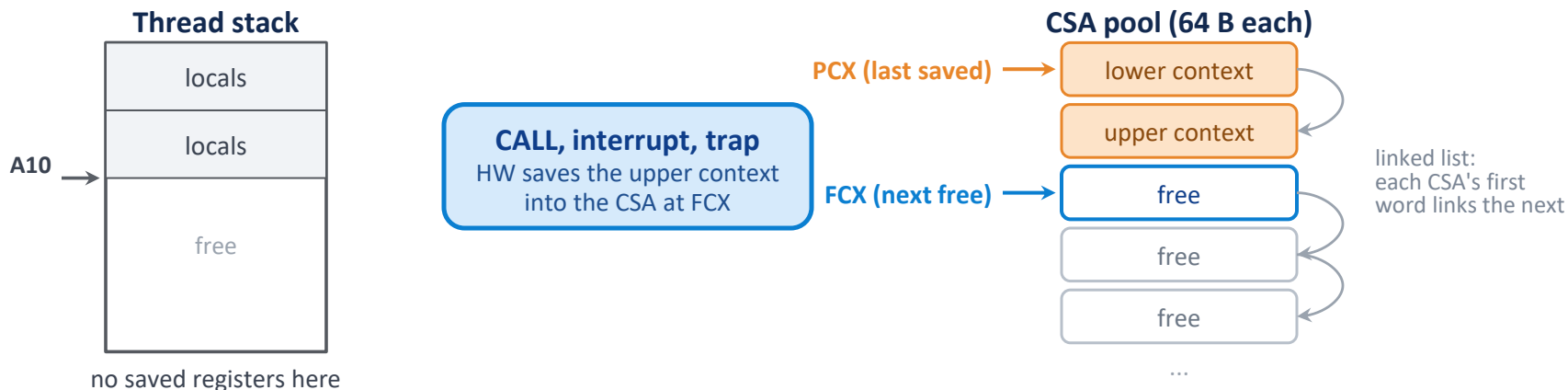
Runtime arch services

CSA context switch, interrupts, traps

Context Save Areas (CSA)

- Context saved in CSAs, not on the stack
- CSA: 64-byte block from a RAM pool
- Free list: FCX next free, PCX last saved
- Upper context: saved by hardware
- Lower context: saved by software

Context lives in CSAs, not on the stack



Upper context (HW)

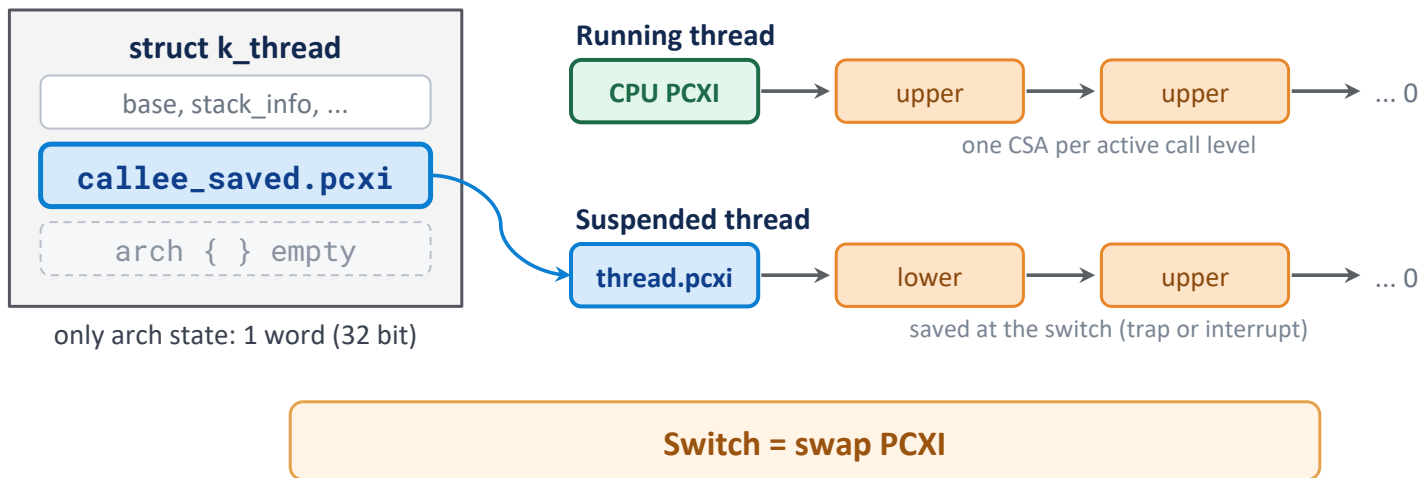
PCXI	PSW	A10	A11	D8 - D11	A12 - A15	D12 - D15
------	-----	-----	-----	----------	-----------	-----------

Lower context (svlcx)

PCXI	A11	A2	A3	D0 - D3	A4 - A7	D4 - D7
------	-----	----	----	---------	---------	---------

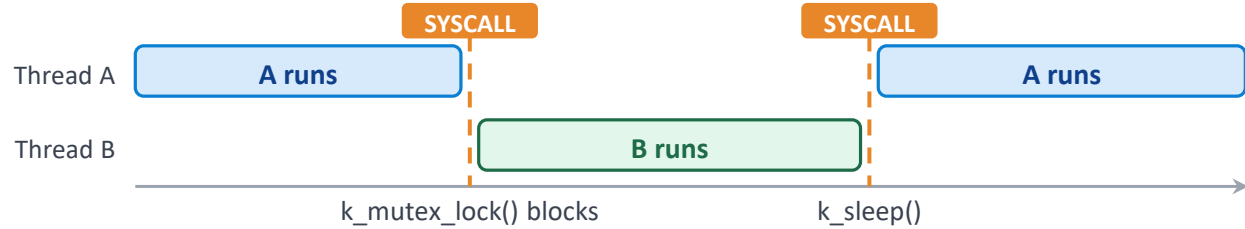
16 words = 64 bytes per CSA, word 0 links to the next CSA

A thread is one word: PCXI

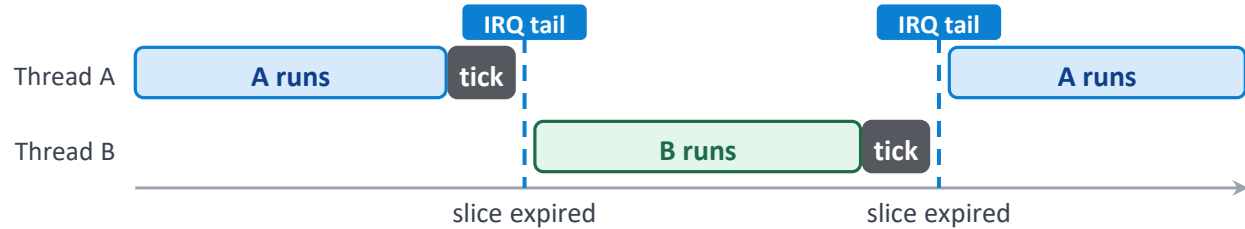


When a thread gives up the CPU

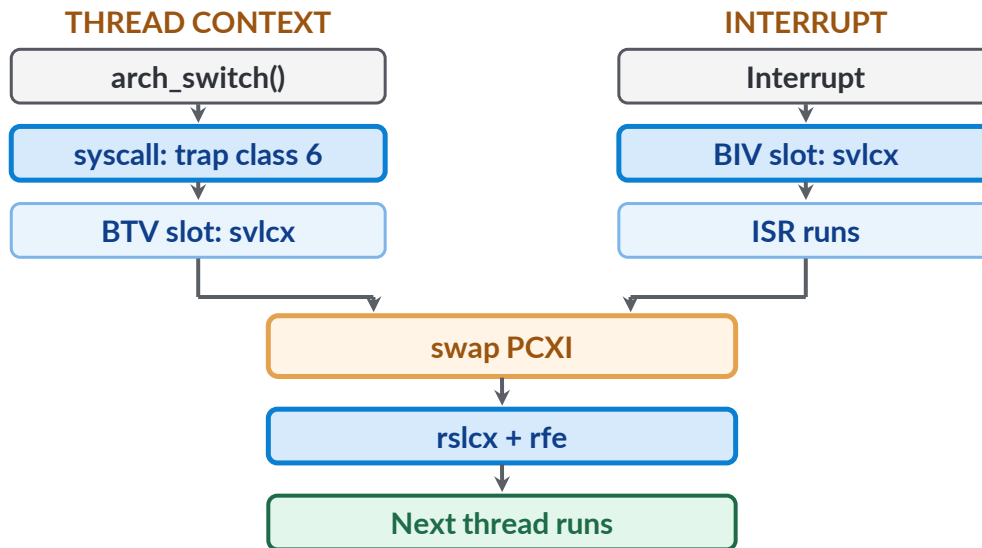
Voluntary
sleep or lock



Time slice
equal priority



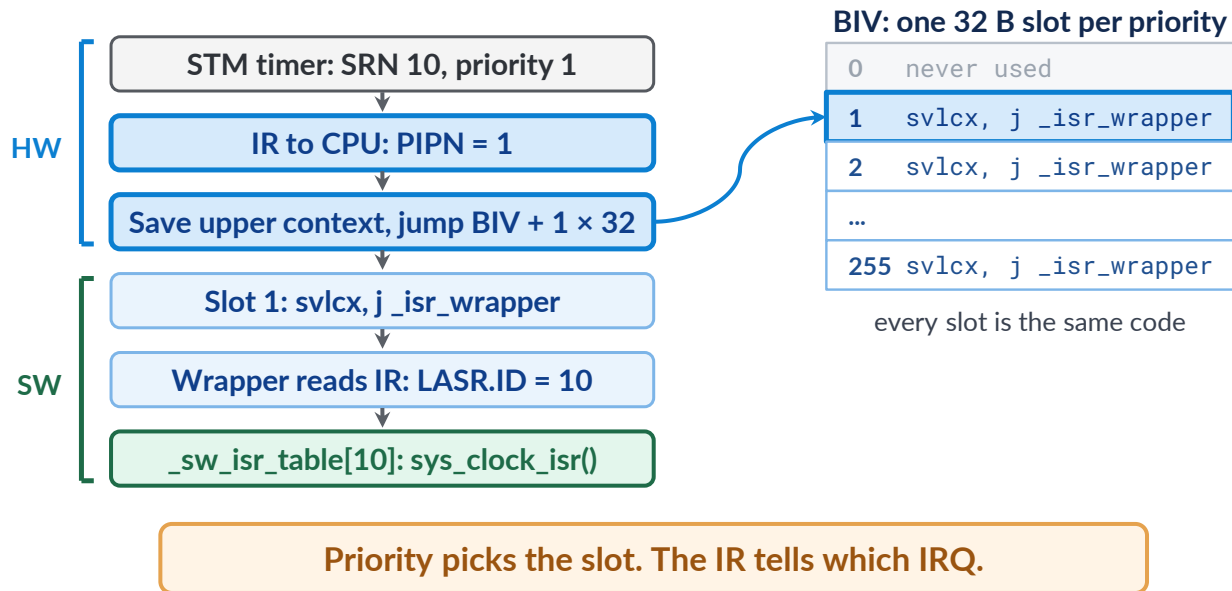
Two entries, one exit



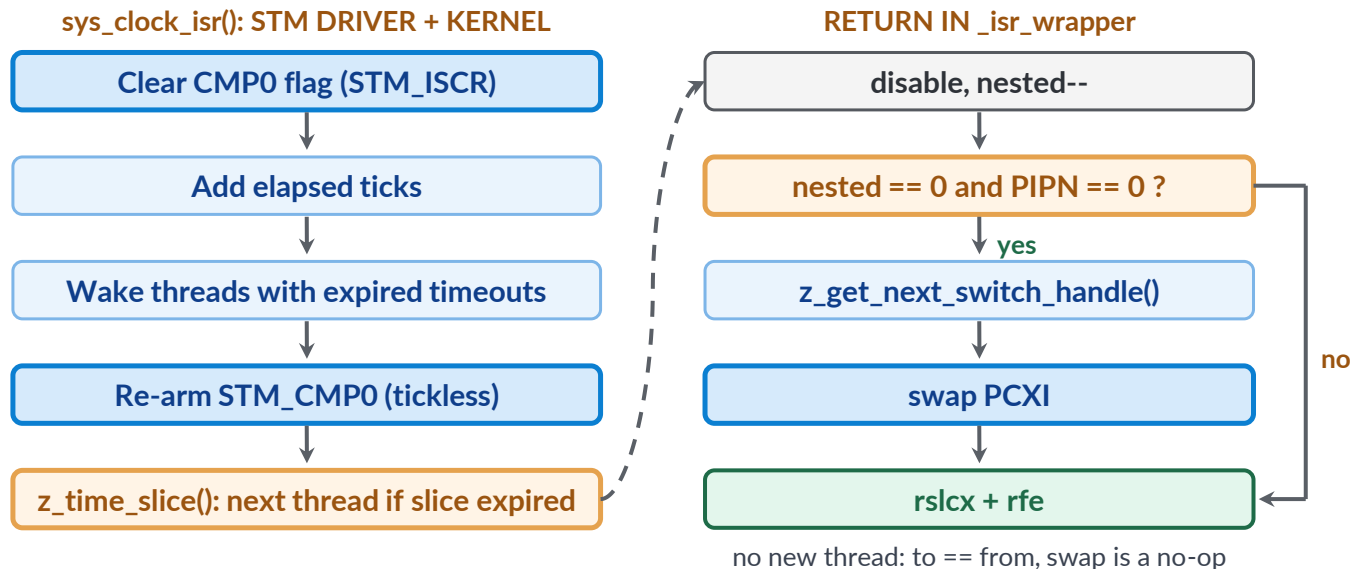
Interrupts and traps on TriCore

- Each request: an SRN with a priority
- IR picks the highest, CPU jumps to its slot
- Slots hold code, not addresses
- Traps: 8 classes, separate table (BTV)
- Every entry: svlcx, then C

One interrupt, end to end

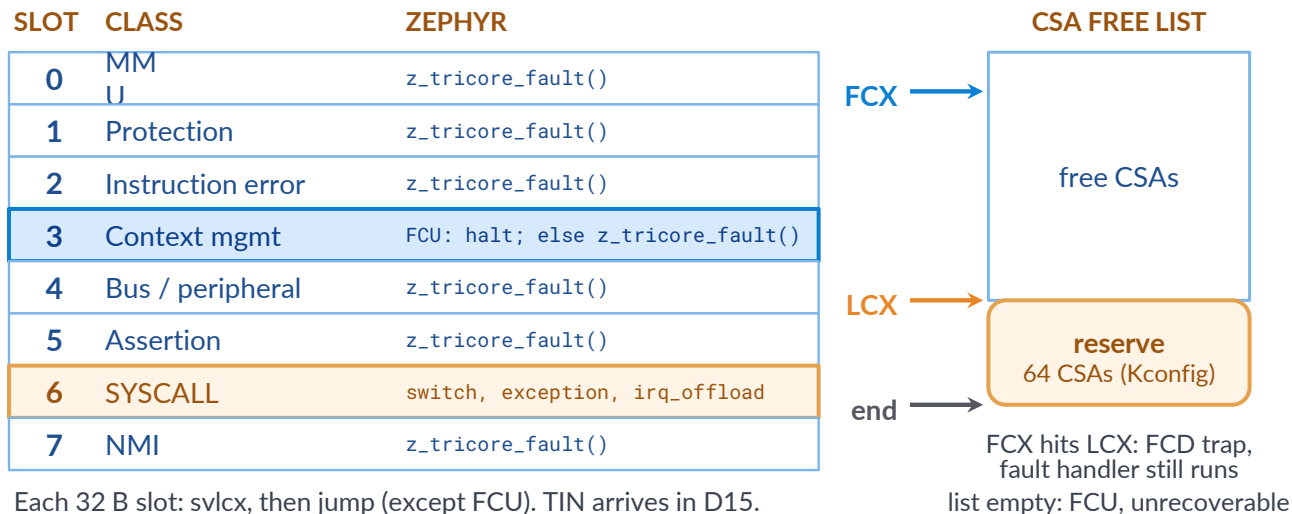


Timer ISR: tick work and return



The ISR only picks the next thread. The wrapper switches.

Traps: one slot per class



CPU idle: enable; wait (ARM / RISC-V: wfi)

Runtime: takeaways

- Context lives in CSAs; a thread is one PCXI
- Switch = SYSCALL trap or IRQ tail
- One exit: swap PCXI, rslcx, rfe
- Priority picks the slot; IR names the IRQ
- Out of CSAs: FCD warns, FCU is fatal

Beyond the arch layer

SoC, board, QEMU

SoC and board: thin by design

- Two boards: qemu_tc3x, qemu_tc4x
- SoC: Kconfig + linker only
- No SoC C code
- Silicon boards: downstream

QEMU and toolchain

- QEMU: downstream, binary for now
- Toolchain: Infineon tricore-elf-gcc
- CI: docker-image PR #301
- Next: upstream QEMU

Upstreaming challenges

HAL, review, what's next

HAL: iLLD register headers

- hal_infineon: TC3x / TC4x headers + pinctrl
- BSL-1.0: first in Zephyr, TSC + Board approved
- Arch and base drivers: no iLLD
- iLLD uses C bit-fields (RFC #113762)
- iLLD in drivers? Open

Upstreaming: review and ecosystem

- Apr to Sep 2026: 7+ review rounds
- ~15K lines, 5 boards cut to QEMU-only
- Longest wait: toolchain in CI
- Twister found generic bugs
- Lesson: patience, toolchain early

Remaining and future work

- Silicon SoCs and boards upstream
- QEMU upstream, TriCore in Zephyr CI
- Userspace, MPU, multi-core
- CSA hardening: limit checks, FCD recovery
- iLLD in drivers: decide
- Safety and MISRA work

Visit us on Zephyr Booth

Zephyr Booth 33, Floor 2

Open Source Summit, Europe 2026

7–9 October 2026, Prague Congress Centre, Prague



References

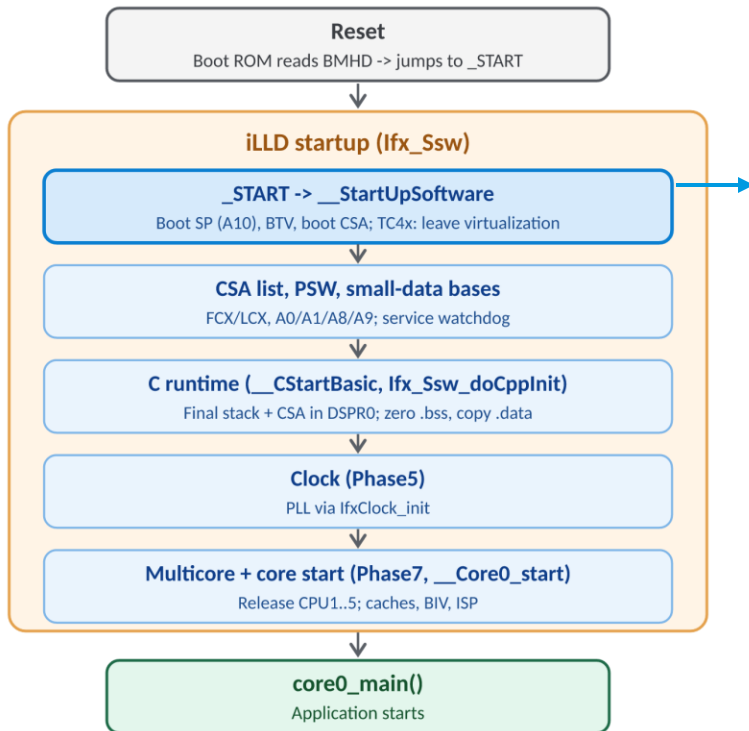
Arch port:	https://github.com/zephyrproject-rtos/zephyr/pull/107516
Downstream (silicon):	https://gitlab.com/linumiz/infineon/release/zephyr-aurix
QEMU for AURIX:	https://gitlab.com/linumiz/infineon/release/qemu-aurix
CI image:	https://github.com/zephyrproject-rtos/docker-image/pull/301
HAL:	https://github.com/zephyrproject-rtos/hal_infineon/pull/51
BSL-1.0 RFC:	https://github.com/zephyrproject-rtos/zephyr/issues/107847
Bit-field RFC:	https://github.com/zephyrproject-rtos/zephyr/issues/113762

Thank You

contact@linumiz.com



Backup: Baremetal bootflow (iLLD)



ENTRY(_START)

MEMORY {

```
pfls0_nc (rx) : org = 0xA0000000, len = 4M
pfls0     (rx) : org = 0x80000000, len = 4M
dsram0    (w) : org = 0x70000000, len = 240K
```

}

/* BMHD in UCB @ 0xAE404800 -> start address 0xA0000000 */

SECTIONS {

```
.start_tc0 (0xA0000000) : {
    KEEP(*(.start))
} > pfls0_nc
.traptab_tc0 (0x80000100) : {
    __TRAPTAB_CPU0 = .;
} > pfls0
.inttab_tc0 (0x803FE000) : {
    __INTTAB_CPU0 = .;
} > pfls0
}
```

Backup: Why early arch init is in assembly

1. Program control registers directly (mtcr): FCX/LCX, BTV, BIV, ISP, PSW
2. Set the stack pointers (ISP and A10) before any C code
3. Start at a fixed reset address, with no C runtime or stack yet
4. .data/.bss not initialised yet, so no globals

5. **TriCore specific: every C "call" saves context into a CSA, so no C function can run until the CSA list exists**