LIGHTNING TALK · FOSS INDIA

From Architecture to Tasks

Designing real-time MCU firmware in heterogeneous APU-MCU systems.

SPEAKER
Udhay, Co-Founder & CTO, ThynkLoop and Thynkure

DATE
Sept 27, 2026

LICENSE
CC BY-SA 4.0

WHO I AM

Udhay, Co-Founder and CTO



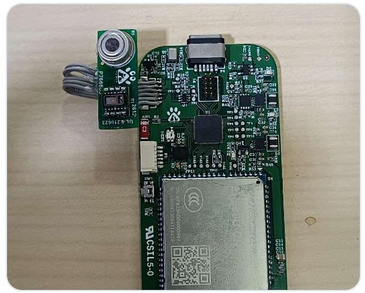
Udhayadhithan
Co-Founder & CTO

ThynkLoop Engineering services company, turnkey product development across embedded hardware, firmware, mechanical design, and AIoT.

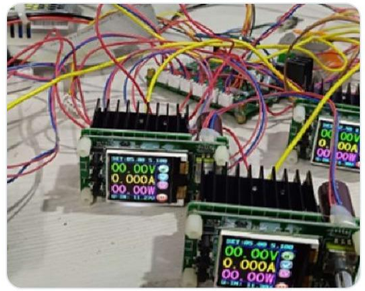
Thynkure A Class C medical device venture, real-time critical by definition, walking the CDSCO to CE to FDA path.

WHAT I HAVE BUILT

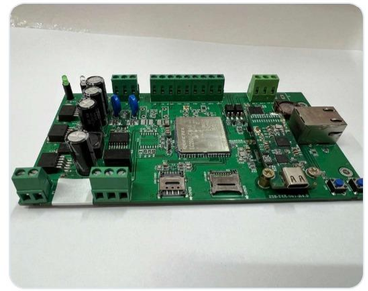
A few things that shaped this talk

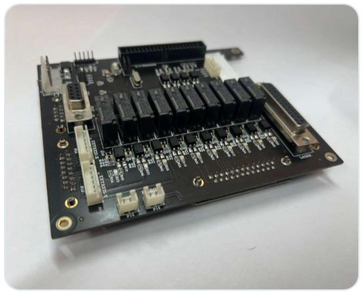
















EXAMPLE

One system, walked through three lenses

01

SYSTEM

A product needs a rich UI, connectivity, and app logic, and hard real-time control, at the same time.

e.g. Digital anesthesia workstation

→

02

LIMITATION

A single chip rarely delivers both well. Linux isn't deterministic, bare-metal isn't built for rich UI.

→

03

MITIGATION

Split the compute, one chip for each job, connected by a protocol both sides trust.

That mitigation is the architecture we'll walk through next.

CROSS ARCH

Splitting the compute across two chips

APU · LINUX

STM32MP1

Display, connectivity, application logic

- display
- connectivity
- application logic

IPC



MCU · RTOS / BARE-METAL

STM32H7

Real-time subsystem

- sensor acquisition
- motor control
- peripheral I/O

DESIGNING SUCH SYSTEMS

Things to consider before you split

01 CLOSER LOOK →

Timing criticality

Can any jitter or delay be tolerated by the control loop?

02

UI and connectivity needs

Does the product genuinely need a full OS?

03 CLOSER LOOK →

Failure isolation

Should a crash on one side be able to take down the other?

04

Team and tooling readiness

Can you maintain both a Linux BSP and RTOS firmware?

Two of these are worth a closer look.

CROSS ARCH

Keeping the real-time side deterministic

STEP 1

ISR

minimal work, signal only

→

STEP 2

Queue / semaphore

defers the handoff

→

STEP 3

Task

does the real work, on schedule

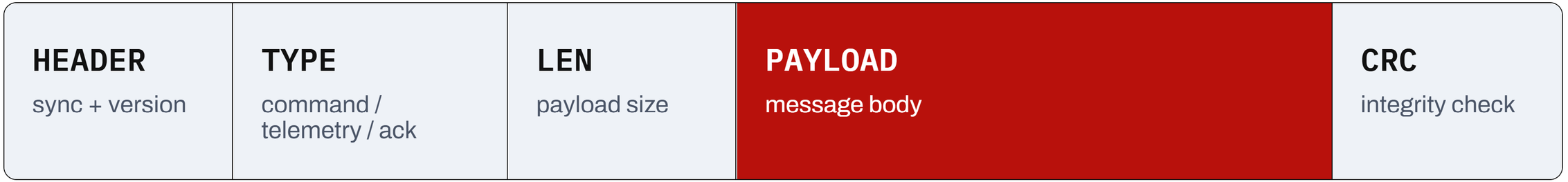
ONE LAYER UP

ISR-to-task deferral for sensor sampling, and the same idea one layer up: incoming Linux commands land in a queue too, so a slow or bursty APU never gets a direct line into the control loop.

CROSS ARCH

A protocol built for two independent chips

ONE IPC FRAME



Fixed framing

A small set of message types

A checksum on every message

Fixed framing, a small set of message types, a checksum on every message, so either side can tell a valid message from noise or a partial write, and recover on its own if the other side resets.

WHERE IT HELPS

Why a stable protocol pays off later

SWAPPED

APU · SWAPPED

Different MPU/APU

e.g. tuned for display or AI workloads

SAME IPC



UNCHANGED

MCU · UNCHANGED

STM32H7

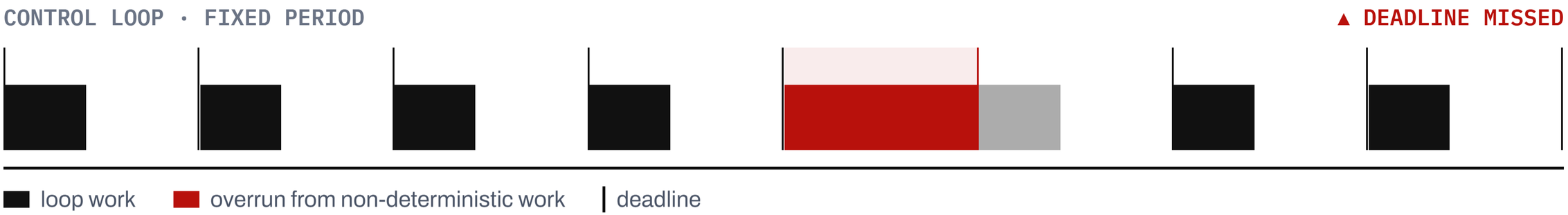
Real-time subsystem, untouched

As long as the new application processor speaks the same protocol, the real-time side never has to change, and the same pattern extends to scaling a single design into a product family.

CLOSER LOOK · 01

Timing criticality

Ask what happens if the loop **misses a deadline**, not whether it's likely to. Microsecond tolerance means no sharing a chip with non-deterministic work.



CLOSER LOOK · 03

Failure isolation

01
Heartbeat
between the two chips

→

02
Timeout
when it's missed

→

03
Resync
once the link is back

A heartbeat between the two chips, a timeout when it's missed, and a resync once the link is back, means the H7 can reset without the display freezing, and Linux can reboot without the motor losing control. Neither side has to assume the other is always there.

H7 RESETS
Display keeps running

LINUX REBOOTS
Motor keeps control

THANK YOU

Let's stay in touch

01

Evaluate the APU/MCU split for your own design

02

Apply the RTOS task patterns to your real-time side

03

Design your IPC protocol to be the stable boundary

04

Reuse this framework on your next HMP decision

Thank you.

udhay@thynkloop.com
+91-8148113569
www.thynkloop.com

MY PROFILE