Bringing Arm Cortex-M to gem5: Enabling Embedded Robotics Research

Zhantong Qiu*, Derin Ozturk†, Jason Lowe-Power*, Christopher Batten†

*University of California, Davis

†Cornell University

CSL

DArchR

DAVIS ARCHITECTURE RESEARCH

CORNELL UNIVERSITY FOUNDED A.D. 1865

UCDAVIS

COMPUTER SCIENCE

Outline

1. Motivation
2. Contribution
3. Evaluation
4. On-going work

Constraints In Robotics

- Run on **Cortex-M MCUs** (e.g. STM32F4)
- Extreme size, weight, and power (SWaP)
- Reactive behaviors, such as simple obstacle avoidance, also require compute

How do we operate complex tasks, such as search and rescue with all these constraints?

Hardware-Software Co-design in Robotics

IEEE ICRA 2026
https://2026.ieee-icra.org › workshops-and-tutorials

Workshops & Tutorials

Workshop, RoboARCH: Robotics Acceleration with Computing Hardware and Systems, FULL, FULL DAY, Schubert 2. Workshop, RoboTac 2026: Embodied Tactile Intelligence ... Read more

IEEE ICRA 2025
https://2025.ieee-icra.org › Events

RoboARCH: Robotics Acceleration with Computing Hardware ...

This workshop is **designed to build a welcoming community and ignite innovation and collaboration** at the exciting intersection of robotics, computer software, ... Read more

The current gap in gem5

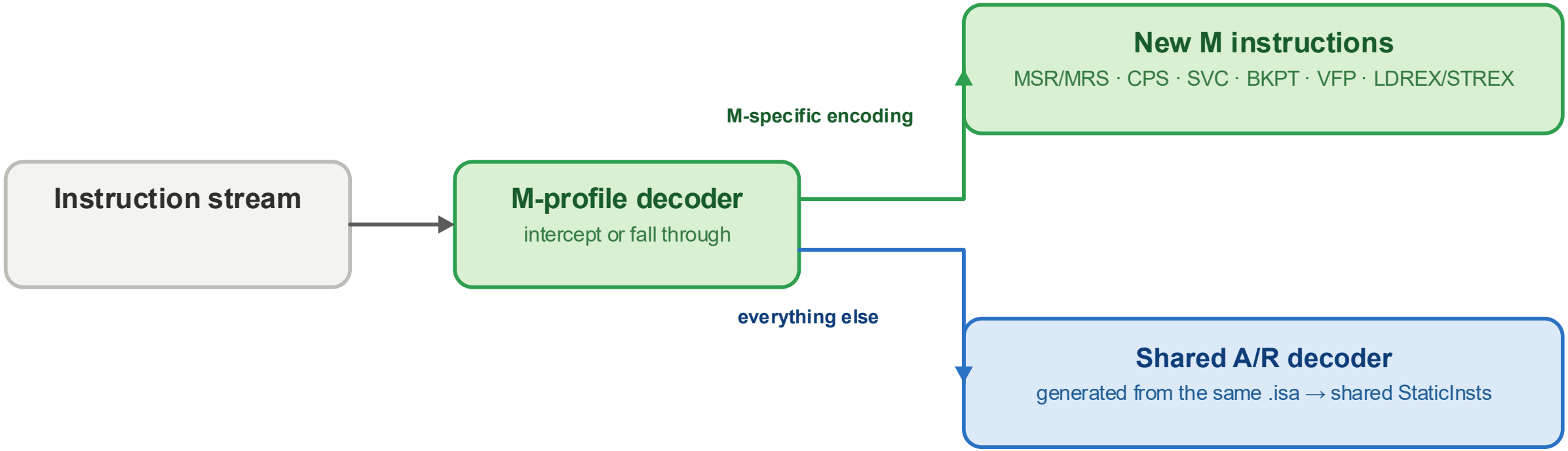
A/R-profile — what gem5 has		M-profile — what Cortex-M needs
GIC generic interrupt controller	≠	NVIC nested vectored interrupt controller
MMU virtual memory + page tables	≠	No MMU — optional MPU only
Table-based exception model	≠	Address vector table + Thread/Handler
(no equivalent)	≠	Tail-chaining · late-arriving · lazy FP

→ M-profile gets its own ISA, NVIC, MMU and fault model — not a restricted A/R core

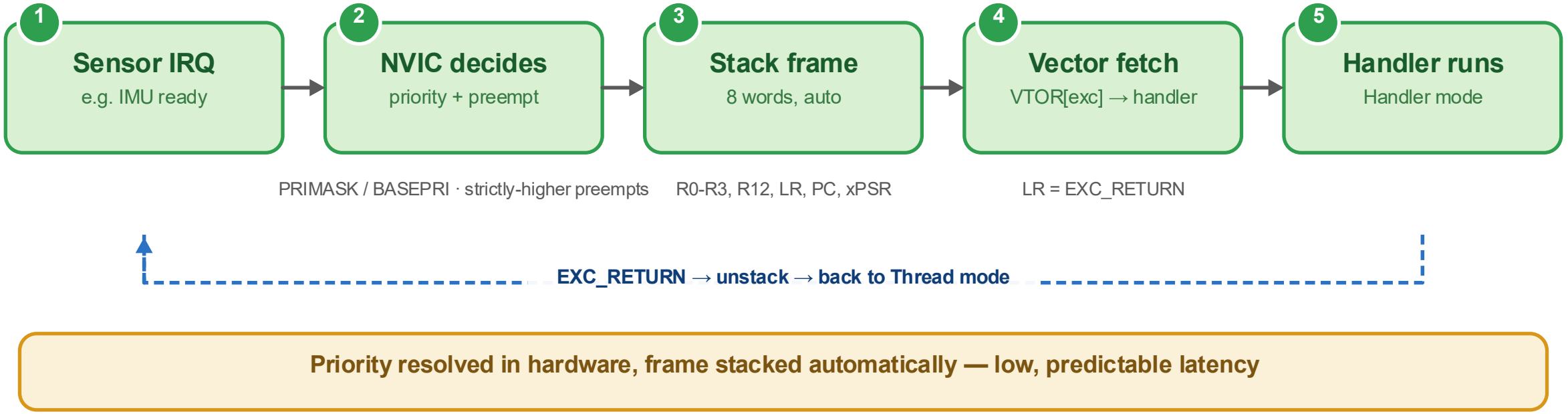
Contribution Of This Work

- Arm-M class decoder, registers, and instructions
- M-profile fault model, System Control Space, thread handling, and NVIC
- M-profile gem5 MMU and partial stacking optimization
- M-profile Platform
  - Flexibly describe the memory and device layout of the board in Python
- gem5 stdlib Cortex-M board and STM32G474RE platform

Arm-M class decoder, registers, and instructions



M-profile fault model, System Control Space, thread handling, and NVIC

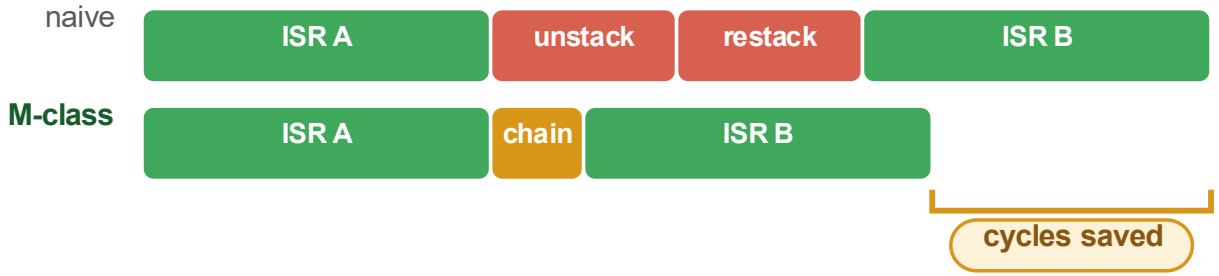


Priority resolved in hardware, frame stacked automatically — low, predictable latency

M-profile gem5 MMU and partial stacking optimization

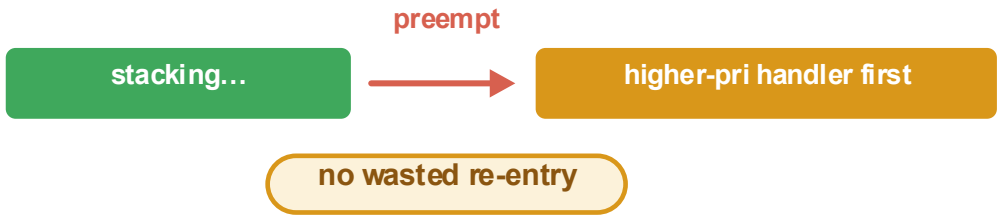
1 Tail-chaining

back-to-back ISRs skip
unstack + restack



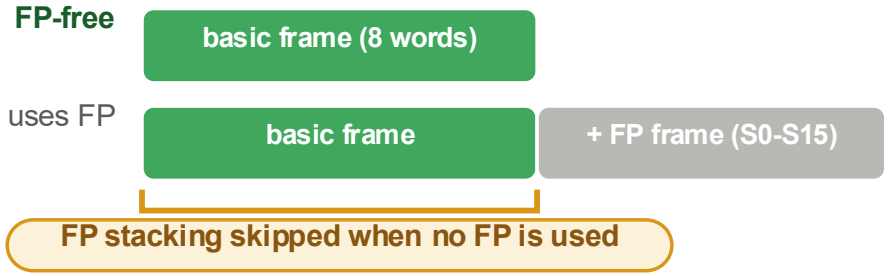
2 Late-arriving

a higher-priority IRQ
during entry preempts



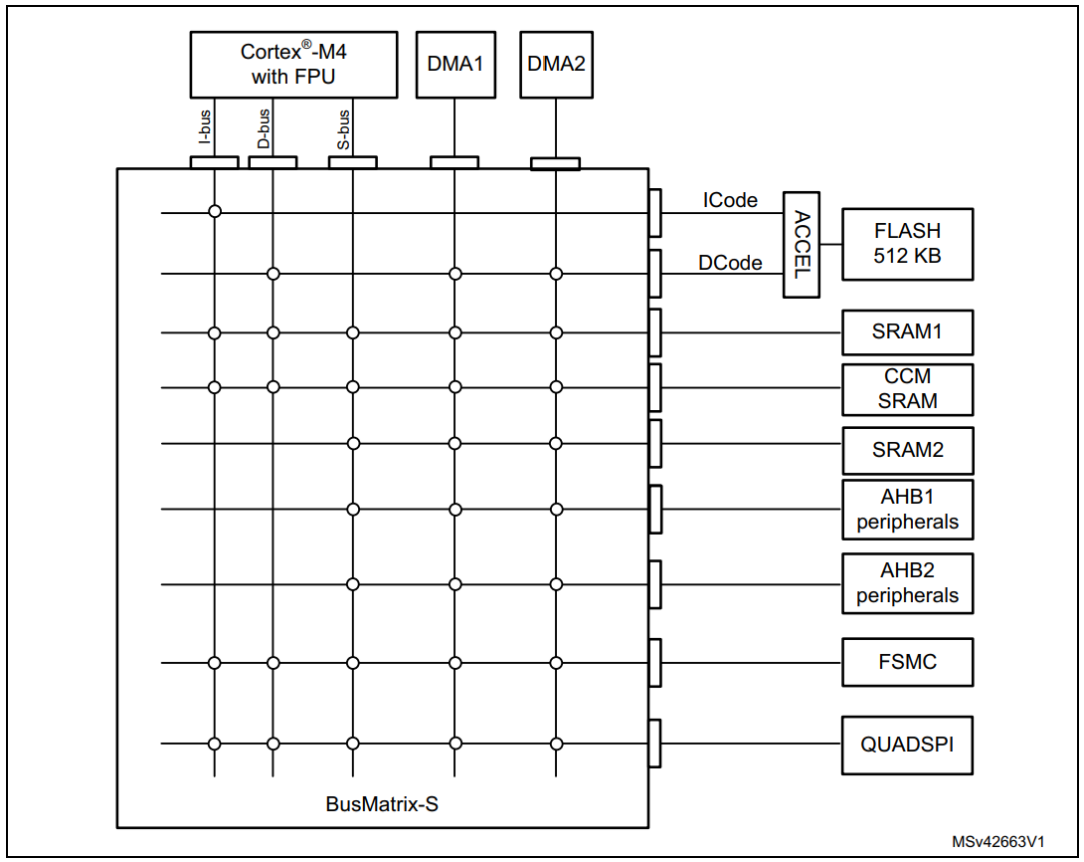
3 Lazy FP save

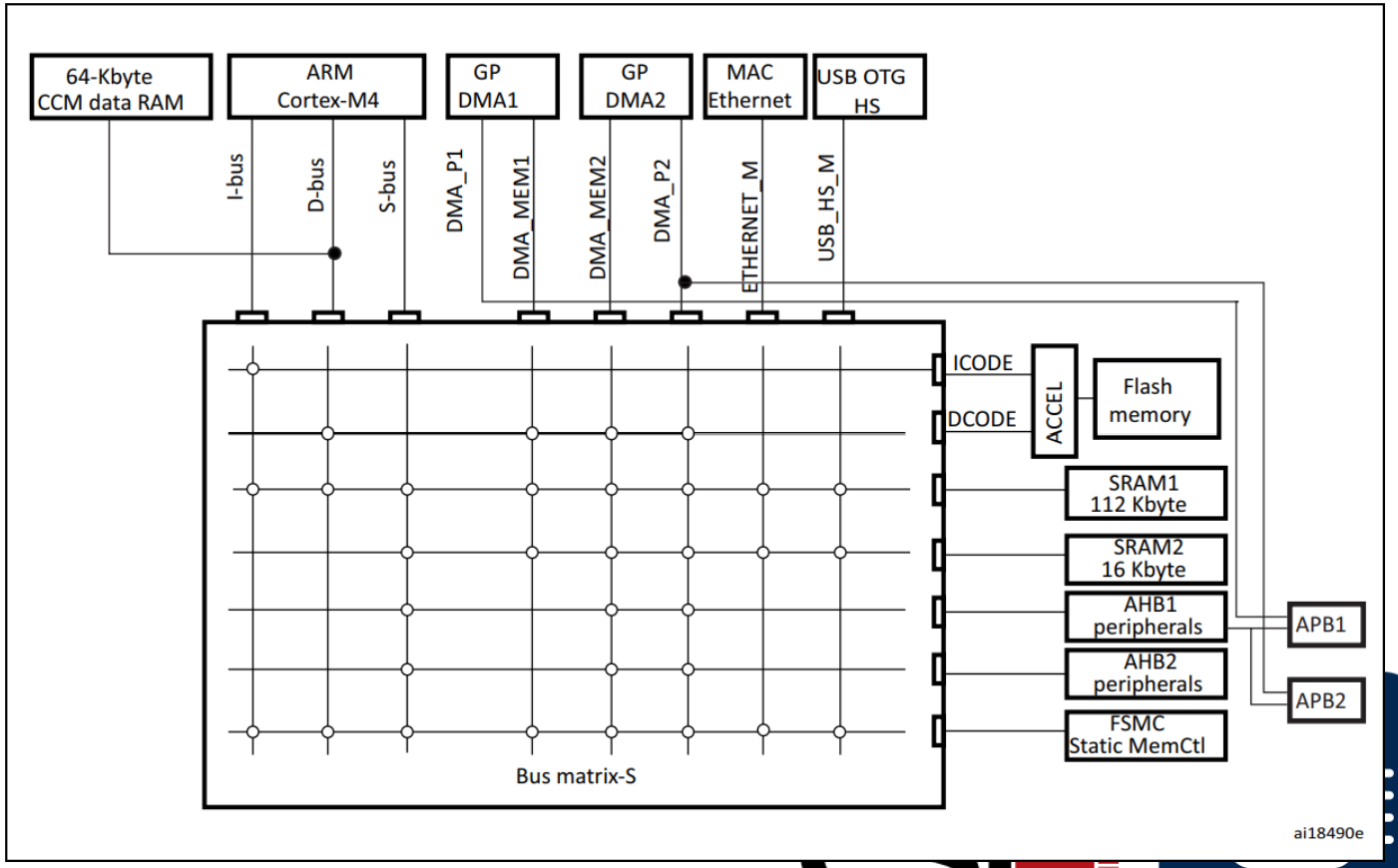
skip stacking S0-S15
unless the handler uses FP



M-profile Platform

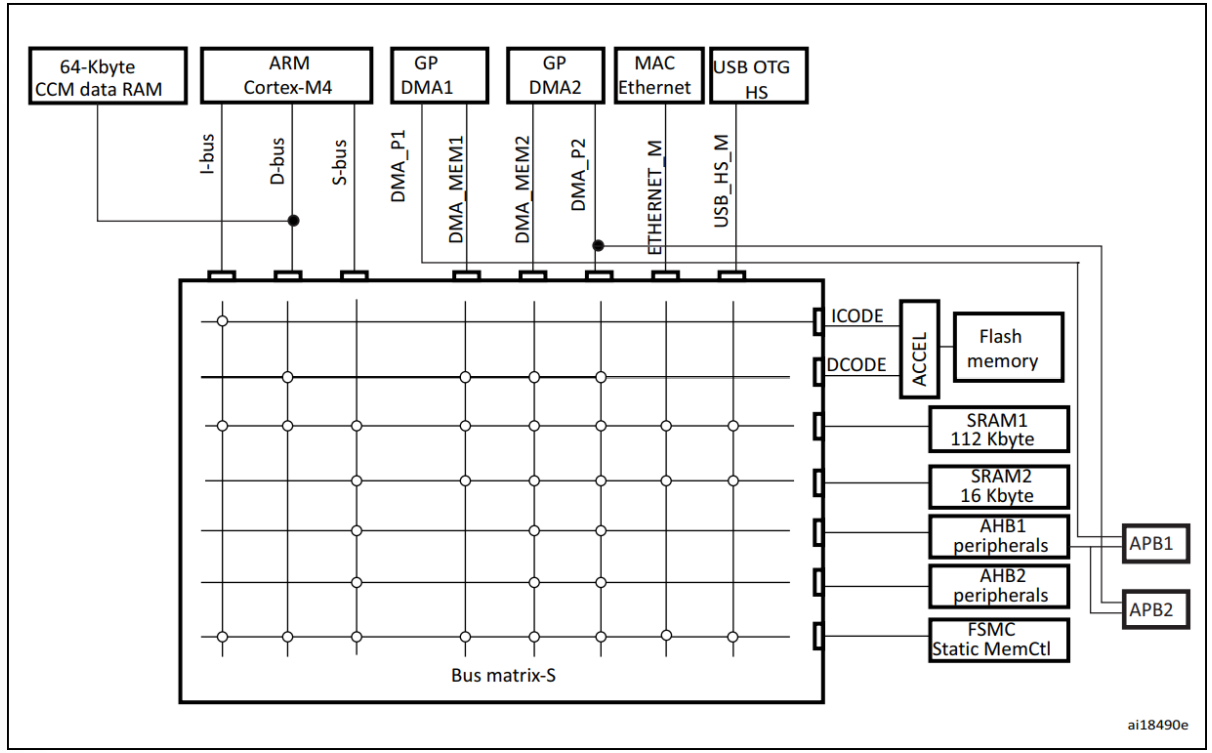
Why do we need a M-profile platform?





M-profile Platform

```python
class STM32F429Platform(ArmMPlatform):
    code_ranges = [
        AddrRange(0x08000000, size="1MiB"),    # Flash bank 1
        AddrRange(0x08100000, size="1MiB"),    # Flash bank 2
    ]
    sram_ranges = [
        AddrRange(0x20000000, size="112KiB"), # SRAM1
        AddrRange(0x10000000, size="64KiB"),  # CCM (outside SRAM region)
    ]
    periph_ranges = [
        AddrRange(0x40020000, size="448KiB"), # AHB1
    ]
    external_ram_ranges = [
        AddrRange(0x60000000, size="8MiB"),    # external PSRAM via FMC
    ]
    external_device_ranges = [
        AddrRange(0xC0000000, size="16MiB"),  # memory-mapped LCD controller
    ]
    ppb_range = AddrRange(0xE0000000, size="1MiB")
    vendor_ranges = [
        AddrRange(0xE0100000, size="4KiB"),    # proprietary OTP/debug regs
    ]
    boot_alias_ranges = [
        AddrRange(0x00000000, size="1MiB"),
    ]
    # -- System Control Space (mandatory device) ----------------------
    scs = MProfileSCS(num_irqs=91, priority_bits=4)
    # -- Pluggable PIO devices ----------------------------------------
    devices = [
        MProfileDWT(pio_addr=0xE0001000),      # Data Watchpoint & Trace
        Pl011(pio_addr=0x40011000),            # USART1
    ]
```



gem5 stdlib Cortex-M board & STM32G474RE platform

```
board = STM32G474REBoard(
        cpu_cls=ArmMSignalCPU,
        flash_latency=args.flash_latency,
        flash_address_phase_latency=args.flash_address_phase_latency,
        flash_buffer_hit_latency=args.flash_buffer_hit_latency,
        flash_read_buffer_size=args.flash_read_buffer_size,
    )
```

Evaluation

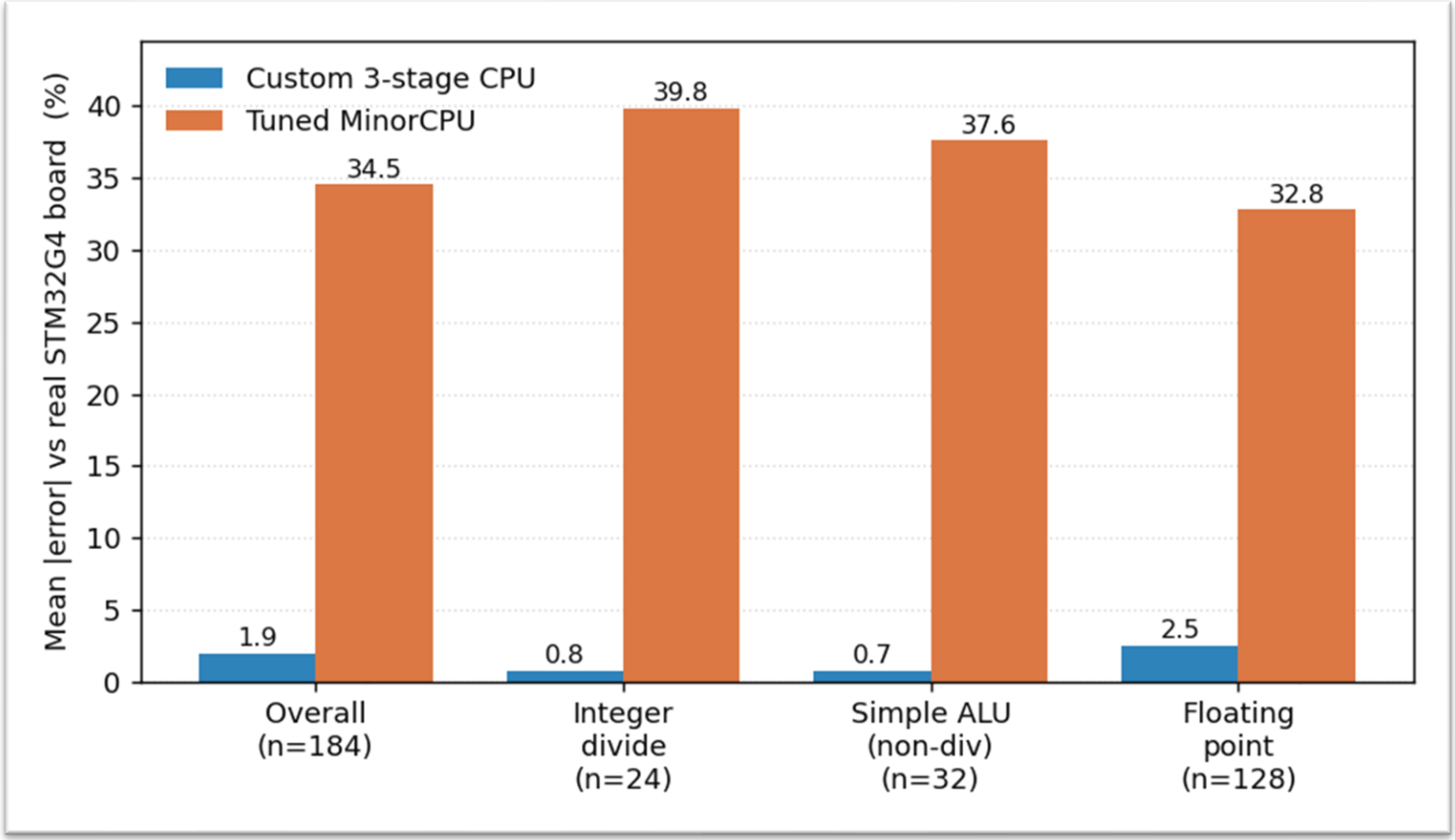
Functional Correctness

- Validated correctness with real hardware using microbenchmarks, random generated instruction sequences, and selected kernels from Entobench, which involve heavy fp instructions, interrupts, and semihosting mechanisms.
- Validated thread handling by running FreeRTOs with multithread workloads.

Evaluation

Performance Error

- We found that the contribution of this work doesn't impact the performance error.
- Comparing numbers of cycles.
- The same binaries ran on both real hardware board and gem5 simulation.

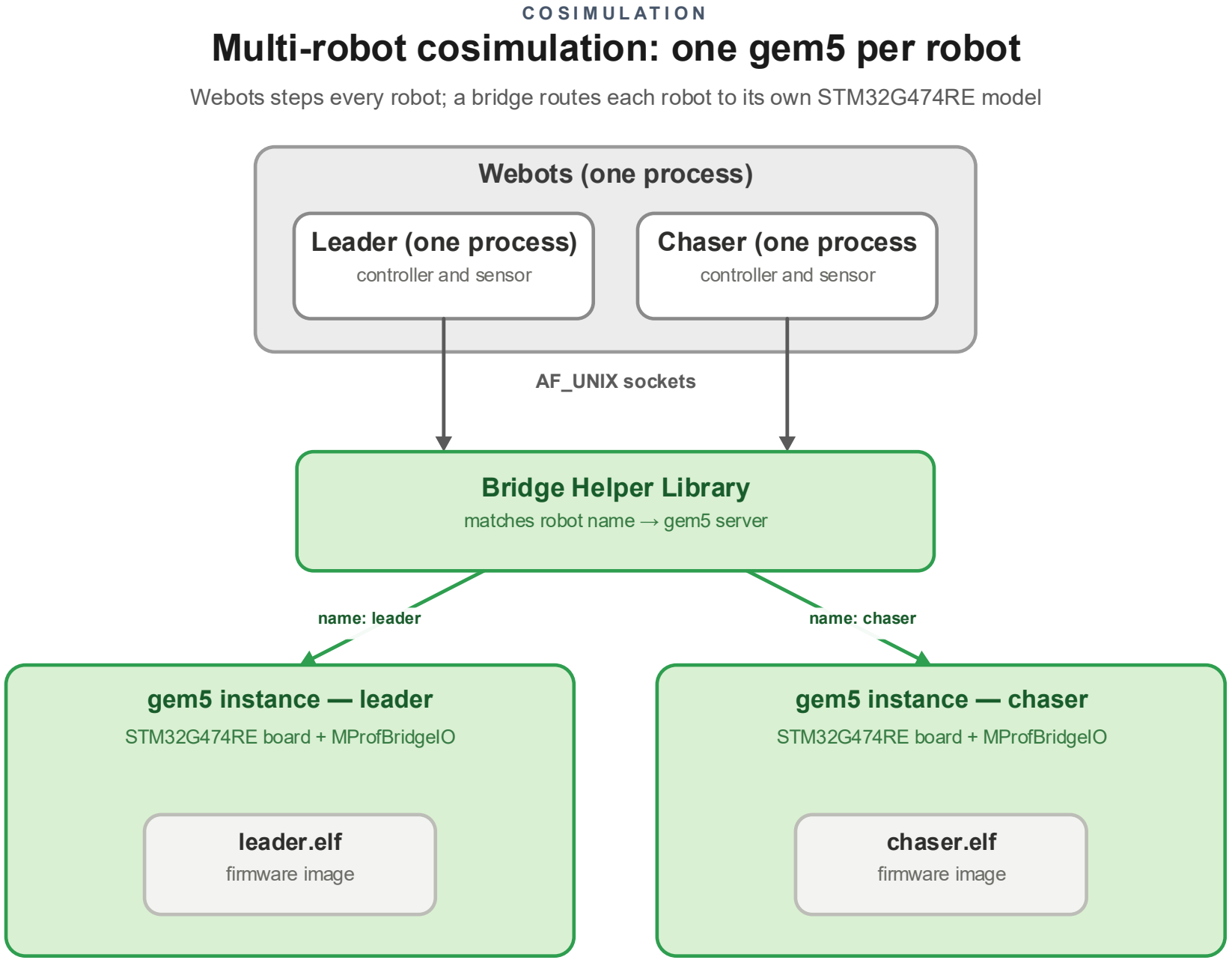


Evaluation

Can we use our model to do SoC-in-the-loop robotic simulation?

Link to the repo of the example:

https://github.com/studyztp/webots-gem5-example



Evaluation

Co-sim Slowdown

- Webots alone completes in 34.60 s; adding gem5 raises this to 1900.49 s **(~55×),** and skipping idle periods where the firmware spins waiting for interrupts reduces it to 464.71 s **(~13×).**

On-going Work

- MPU region protection
- Tail-chaining
- Full support of lazy FP stacking
- Improve the adaptive real-time memory accelerator
- AHB bus modeling

Upstream Plan

- Discuss with the community to improve the design