MCU-MDT

A Lightweight, Portable UART Debugging Monitor for Resource-Constrained Microcontrollers

Nicoleta-Alexandra Trancă · Mihai-Bogdan Stoica · Viorel Bostan

POLITEHNICA Bucharest
Technical University of Moldova

Debugging through an existing UART

The practical problem

A compatible probe or accessible JTAG/SWD connection may be unavailable.

The proposed approach

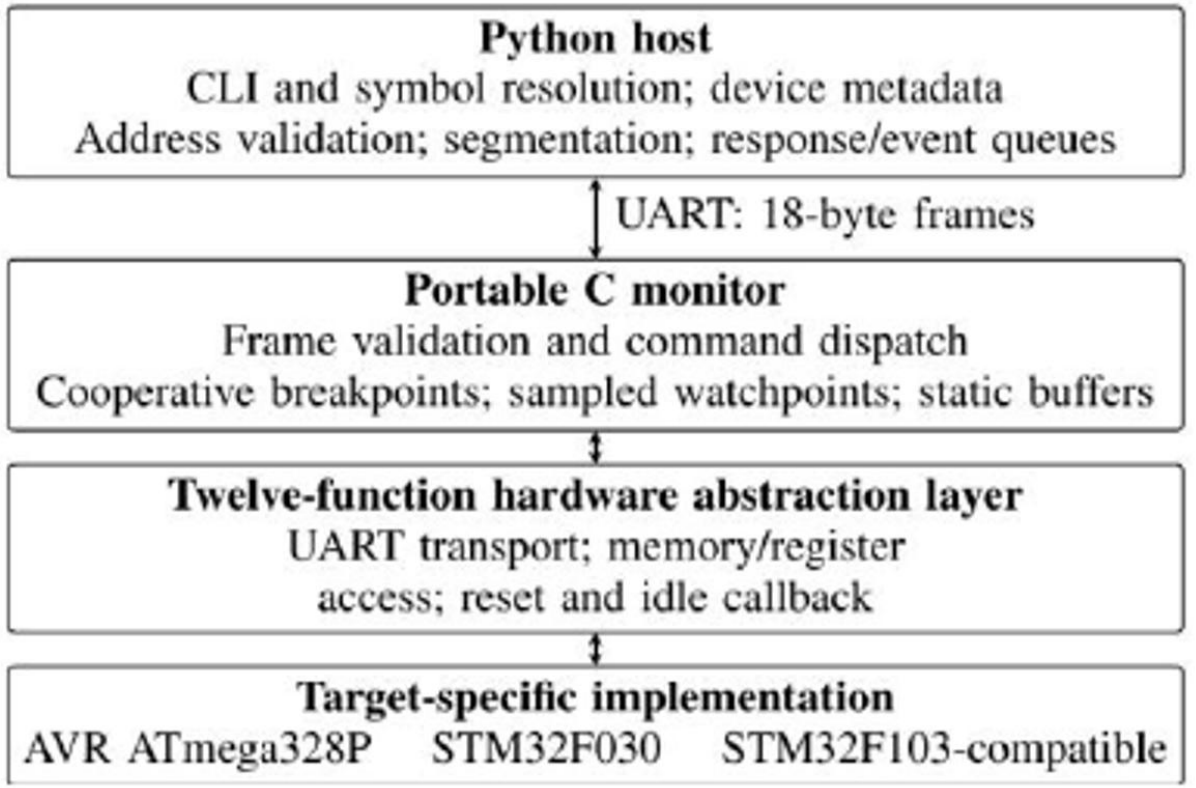
A Python host and resident C monitor expose memory and cooperative execution control.

Design budgets: <3.5 kB Flash and <384 B static storage

Host and target responsibilities

Host

Device metadata and ELF symbols
Address validation
Command segmentation
Response and event queues



The MCU processes binary commands without parsing text or device metadata

Portability through a twelve-function HAL

HAL group	Functions	Responsibility
UART	6	Transport and queue handling
Memory/registers	4	Platform-specific reads and writes
Control	2	Idle callback and reset

Shared monitor core across AVR and STM32

AVR supports Flash reads and EEPROM access. STM32 adds Flash programming and page erasure. All buffers and debug slots use static allocation.

A fixed 18-byte binary protocol

Bytes	Fields
0–4	Start, command, flags, sequence, memory ID
5–10	32-bit address and 16-bit length
11–14	Payload or parameters, up to 4 bytes
15–17	CRC-16 and end delimiter

CRC-16/CCITT-FALSE covers bytes 1–14

Little-endian fields. The host segments larger transfers.
Payload occupies at most 22.2% of one frame.

Cooperative breakpoints and sampled watchpoints

Breakpoints

Instrumented MDT_BREAKPOINT(id) sites

Application progress pauses while the monitor continues serving commands

Interrupts and peripherals may continue

Watchpoints

Compare a masked value with its previous sample

$(v_t \,\&\, m) \neq (v_{t-1} \,\&\, m)$

A change between samples can be missed

No arbitrary instruction stepping or architectural halt

Minimal application integration

```
mcu_mdt_init();
for (;;) {
    mcu_mdt_poll();
    ++counter;
    MDT_BREAKPOINT(0);
}
```

Host workflow

1. Enable breakpoint slot 0
2. Resolve counter from ELF
3. Read four SRAM bytes
4. Send NEXT to continue

The application must give the monitor time to run

Evaluation across three configurations

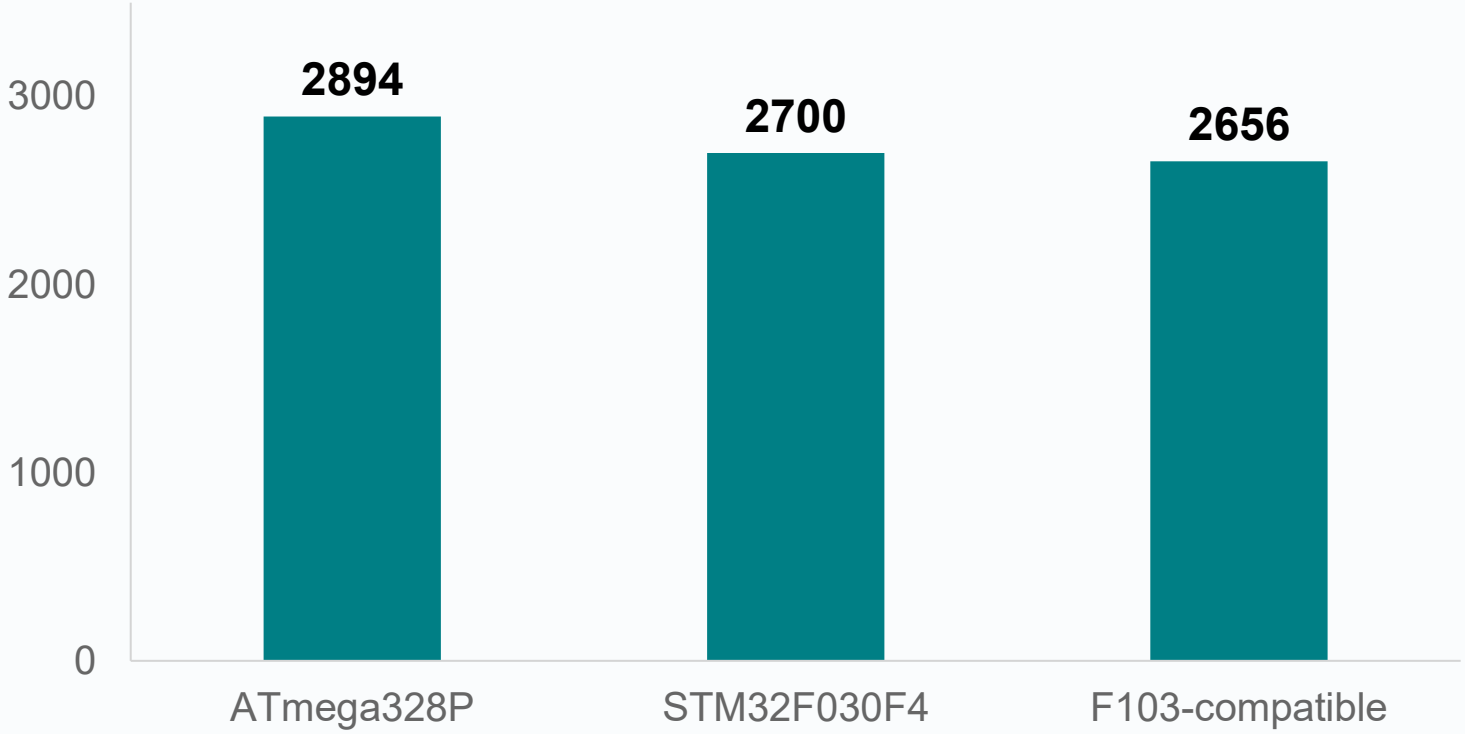
Target	Clock	Physical	Renode
ATmega328P	16 MHz	Yes	Not reported
STM32F030F4	8 MHz	Yes	Yes
F103-compatible	8 MHz	Clone	F103 model

248 cases in the suite

151 unit + 44 integration + 53 hardware cases
Executed and skipped counts per target are not reported.

The physical clone does not establish validation on a genuine STM32F103

Optimized polling builds meet the memory budgets



248–263 B

Static SRAM
Polling builds, -Os

-O0 exceeds the Flash budget on every target

Reported firmware-build sizes, not isolated monitor overhead.
Static SRAM excludes stack usage.

UART transmission dominates read latency

18.75 ms

Serialization-only lower bound
18-byte request + 18-byte response
19,200 baud, 8N1

Configuration	Reported round trip
AVR	18.80 ms
F030	18.85 ms
F103-compatible	18.80 ms

Twire = 2 × 18 × 10 / 19,200

Values are approximate. A separate gap estimate does not reconcile exactly.
Burst capture boundaries remain unverified, so no precise CPU overhead follows.

A proposed two-hour student laboratory

Time	Activity	Learning focus
20 min	Build and connect	Link C symbols to SRAM bytes
30 min	Break and continue	Distinguish main-loop pause from halt
30 min	Change watchpoint masks	Observe sampled detection limits
40 min	Diagnose an injected fault	Explain evidence in a command log

Pairs use a supported board, USB–UART adapter, and Python host

Proposed exercise and assessment. No classroom learning gains have been measured.

Conclusions and next steps

One UART interface across AVR and STM32

Optimized polling builds use <2.9 kB Flash and at most 263 B static SRAM.
Cooperative control keeps command service available during a pause.

Practical limits

Instrumented stop sites, sampled watchpoints, one pending event, and possible duplicate execution after retries

Next: higher-rate timing, event buffering, recovery tests, and verified devices