



FÓRUM DE
SISTEMAS
EMBARCADOS
E IOT 2026



Realização
EMBARCADOS



Do Código ao Campo

Validando firmware com Zephyr RTOS

Jorge Guzman



Firmware · IoT · Indústria

GZM
EMBEDDED SYSTEMS

Jorge Guzman

Fundador da GZM Embedded Systems

Especialista sênior em sistemas embarcados. Atendendo empresas no desenvolvimento de firmware.

+14 anos de experiência
em sistemas embarcados

WEB
gzm-emb.com

LINKEDIN
[/in/eng-jorge-guzman](https://in.eng-jorge-guzman)

Palestras anteriores

OUT/2025

Primeiros passos

Zephyr 4.2 · STM32H753 (Linum)

- O que é o Zephyr
- Instalando o ambiente (SDK, west)
- **Criando um projeto do zero**
- Estrutura de pastas
- DeviceTree, overlays e Kconfig
- Compilando com west build
- Gravação e debug no VS Code
- Relatórios de ROM e RAM

youtube.com/watch?v=KjpwVkJrcsM
github.com/JorgeGzm/Embarcadosslot2025

ABR/2026

Aplicação IoT completa

Zephyr 4.4 · Raspberry Pi Pico W

- **Criando um projeto iot completo**
- Wi-Fi como máquina de estados
- Página web embarcada
- API REST sobre HTTP
- Sistema de arquivos LittleFS
- Shell via serial e Telnet
- Relés, PWM e LED WS2812
- Build, gravação e debug

youtube.com/watch?v=Z43upH2acmo
github.com/JorgeGzm/WebinarEmbarcados2026

HOJE

Do código ao campo

Zephyr 4.4 · STM32H743

- manifest
- Ztest, Twister
- Simuladores: **native_sim** e **Renode**
- Bootloader e atualização em campo
- Automação de testes Robot Framework
- **Demo: Criar rotinas de testes para validar nosso projeto.**

Funcionou na bancada. E no campo?

01 · Introdução ao Zephyr

Zephyr RTOS

- **Open source:** mantido pela Linux Foundation, Apache 2.0
- **Multi-arquitetura:** ARM, RISC-V, Xtensa, x86
- **Mais que um RTOS:** um framework com várias libs
- **Mesmo código, outra placa:** troca o `-b <placa>`

Ambiente de desenvolvimento baseado em:

Zephyr

Kernel, bibliotecas, drivers, hal

SDK

Toolchains, openocd

west

Comando para baixar código, compilar, gravar, depurar e testes.

EM NÚMEROS Zephyr v4.4.1 deste workspace

1014**placas suportadas**
west boards**11****arquiteturas**
arch/archs.yml**610****exemplos prontos**
samples/**/sample.yml**1113****arquivos de teste**
tests/**/testcase.yml

Os arquivos de configuração do projeto

west.yml manifest/

west

Qual Zephyr (v4.4.1) e quais módulos o west update baixa.

prj.conf app/

Kconfig

Recursos da app em qualquer placa: shell, LVGL, log...

weact_stm32h743.conf app/boards/

Kconfig

Só nesta placa: USB, flash, MCUboot, watchdog.

debug.conf app/

Kconfig

Extra do build de debug: -Og e info de threads.

weact_stm32h743.dts boards/witte/.../

Devicetree

Descreve a placa: CPU, memórias, pinos e periféricos.

weact_stm32h743.overlay app/boards/

Devicetree

Ajusta o Devicetree para esta app: LittleFS na NOR SPI, slot0.

board.cmake boards/witte/.../

CMake

Gravadores do west flash (J-Link, CubeProgrammer, DFU) e Renode.

CMakeLists.txt app/

CMake

Fontes, headers e o módulo gzm/ (ZEPHYR_EXTRA_MODULES).

sysbuild.conf app/

MCUboot

Liga o MCUboot: modo de troca, assinatura e criptografia.

mcuboot.conf · **mcuboot.overlay** app/sysbuild/

MCUboot

Configura o próprio MCUboot: recovery na UART4, watchdog, LED.

VERSION app/

MCUboot

Versão da app: vai para o header da imagem (1.0.0+0).

demo-ecdsa-p256.pem tools/keys/

MCUboot

Assina o firmware; a parte pública vai para o MCUboot.

demo-encryption-p256.pem tools/keys/

MCUboot

Cifra o update; a parte privada vai dentro do MCUboot.

testcase.yaml tests/*/

Twister

Descreve cada teste: plataformas, tags e timeout.

O que este projeto traz de novo

☐ novo · capítulo onde é explicado

	para que serve	capítulo
☐ manifest/west.yml	Zephyr e módulos em versão fixa	03
app/	a aplicação	
☐ CMakeLists.txt · prj.conf · Kconfig	como em qualquer app Zephyr	
☐ src/ · inc/ · boards/ · debug.conf	código, overlays e build de debug	
☐ VERSION	versão gravada no header da imagem	05
☐ sysbuild.conf	liga o MCUboot, assinatura e criptografia	05
sysbuild/ mcuboot.conf · .overlay	configura o próprio MCUboot	05
boards/witte/weact_stm32h743/	a placa, fora da árvore do Zephyr	
support/ *.repl · *.resc	a placa virtual do Renode	02
gzm/ lib/user_mgr · lib/fs_mgr	biblioteca interna como módulo Zephyr	04
tests/ user_mgr · fs_mgr · user_mgr_mock	ztest + Twister	04
robot/	aceitação com Robot Framework	07
tools/keys/	chaves .pem: assinatura e criptografia	05
tools/flasher/	nosso atualizador (GUI e CLI)	06
tools/twister/	o Twister grava a placa pela UART	04
tools/debug/	Renode: rodar e depurar	02
zephyr/ · modules/ · bootloader/	baixados pelo west update (fora do git)	03

Em cinza, o que todo projeto Zephyr tem. Em azul, o que valida e atualiza o firmware em campo.

02 • Simuladores

native_sim: o firmware como programa do PC

SEM QEMU nenhuma instrução é emulada

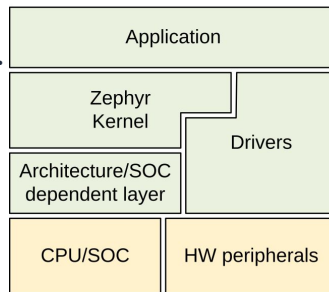
- Compila app + kernel com o **gcc do PC**: **zephyr.exe**
- Threads → Pthreads executadas de forma nativa pelo PC (uma por vez).
- UART4 → terminal **pty** simula comunicação serial
- Display ST7735R → janela **SDL** simula o driver gráfico na tela do PC
- Flash / LittleFS → Arquivo Local salva os dados no disco do PC

```
# Compila
$ west build -b native_sim/native/64 -s app \
  -d app/build_native -- -DBOARD_ROOT=$PWD

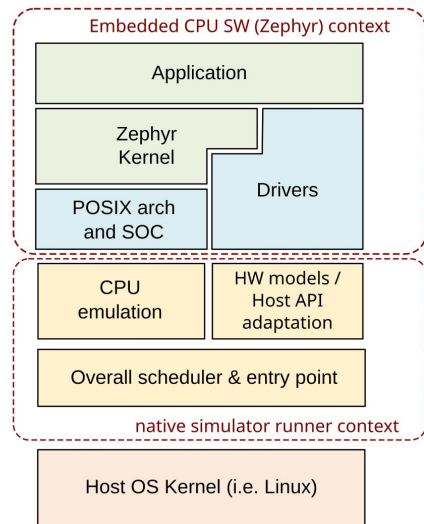
# Roda o programa
$ ./app/build_native/zephyr/zephyr.exe
*** Booting Zephyr OS build v4.4.1 ***
Do Codigo ao Campo demo v1.0.0
<inf> users_app: default admin 1 created
<inf> users_app: 1 users (0 blocked), max 50
<inf> setup: setup done
```

```
DO CODIGO AO CAMPO
usuarios   bloqueados
4/50      1
v1.0.0 SIM
```

Normal Zephyr layering



native_sim & _bsim boards Zephyr layering



Renode

Renode (Antmicro)

escrito em C# / .NET · código aberto · Linux, Windows e macOS

zephyr.elf

o mesmo binário ARM que vai para a placa

Periféricos · C#

modelos que imitam os registradores: UART, SPI, RCC, flash...

Plataforma · .repl

diz em que endereço fica cada modelo; o .resc carrega e executa

CPU · tlbb

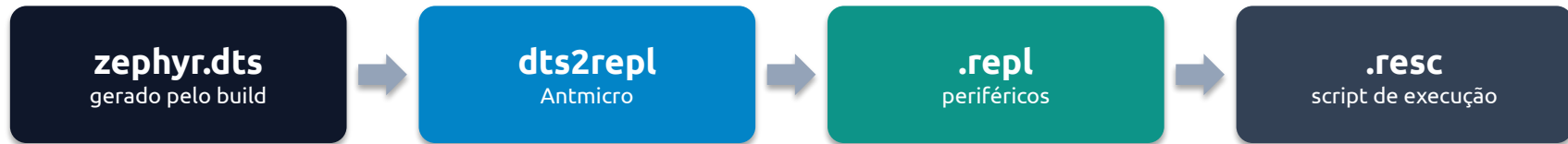
núcleo em C derivado do QEMU: traduz as instruções Cortex-M7 para o PC

```
# boards/witte/weact_stm32h743/support/  
# weact_stm32h743.repl  
cpu0: CPU.CortexM @ sysbus  
    cpuType: "cortex-m7"  
uart4: UART.STM32F7_USART @ {  
    sysbus 0x40004c00; ... }  
spi1: SPI.STM32H7_SPI @ {  
    sysbus 0x40013000; ... }  
flashController:  
    MTD.STM32H7_FlashController  
    @ sysbus 0x52002000
```

Registrador ou periférico sem modelo?

O Renode avisa no log e segue: **Unhandled read... / non existing peripheral**

Renode: do zephyr.dts à placa virtual



```
pip install git+https://github.com/antmicro/dts2repl
dts2repl app/build/app/zephyr/zephyr.dts \
  -o weact_stm32h743.repl

tools/debug/renode_run.sh --debug # GDB :3333
```

AJUSTES MANUAIS

O .repl gerado é o ponto de partida

SPI do H7 saiu com o modelo do F4 → SPI.STM32H7_SPI
Faltou o controlador de flash → STM32H7_FlashController
Nós sem modelo desligados no renode.overlay

native_sim × Renode

native_sim recompila o código para o PC

Qualquer arquitetura do Zephyr

Não interpreta o .elf da placa: o gcc do PC compila o mesmo código C para x86. O núcleo da MCU não importa.

ARM ★

ARM64

RISC-V

Xtensa

x86

ARC

MIPS

Renesas RX

SPARC

OpenRISC

O preço: os drivers do SoC não entram. SPI, QSPI e DMA do STM32 não rodam aqui.

Renode executa o .elf da placa

Núcleos maduros, com placas prontas

Cortex-M ★

Cortex-A/R

RISC-V

x86 / x64

STM32 · nRF52

i.MX8MP · R52

HiFive · LiteX

UP Squared

Fora: sem CPU ou sem placa real

Xtensa

ARC

MIPS

Renesas RX

OpenRISC

AVR

PIC

8051

Núcleo ≠ chip: o Renode tem um Xtensa genérico, mas nenhuma placa ESP32.

native_sim x Renode

native_sim recompila para o PC

- ✓ **Display:** janela SDL, touch pelo mouse
- ✓ **Flash e EEPROM:** em arquivo no PC
- ✓ **Serial:** UART em PTY, um terminal do PC
- ✓ **CAN:** SocketCAN do Linux
- ✓ **Internet:** Ethernet TAP ou sockets do PC
- ✓ **BLE:** adaptador Bluetooth do PC
- ✓ **Emuladores:** SPI, I²C, EEPROM AT24
- ✓ **Wi-Fi:** rede pelo PC; scan e connect só com driver falso
- ✗ Drivers do STM32 não compilam

Renode 1.17 roda o .elf da placa

- ✓ **Núcleo:** Cortex-M7, SPI, QSPI e flash do H7
- ✓ **Serial:** USART/UART do H7
- ✓ **CAN:** FDCAN do H7
- ✓ **Internet:** Ethernet do H7, ligada ao PC por TAP
- ✓ **Tela e SD:** LTDC, DMA2D, SDMMC e touch FT5336
- ✓ **BLE:** só em chips com rádio (nRF52840)
- ✗ **Wi-Fi:** sem modelo; a rede pode vir pelo Ethernet
- ✗ **Display SPI:** ST7735R e ILI9341 sem modelo
- ✗ **NOR W25Q:** só Micron, Macronix e genérica
- ✗ EEPROM AT24, RTC do H7 e USB: sem modelo

03 • Manifest

O arquivo de manifesto (west.yml)

- O arquivo **west.yml** gerencia repositórios Git do projeto com Zephyr
- O comando **west init -l manifest** registra o manifest (.west)
- O comando **west update** clona os repositórios e módulos
- Baixa só os módulos do **allowlist**
- Permite clonar repositórios Git adicionais:
 - Módulos Externos
 - Ferramentas extras

```
# manifest/west.yml
manifest:
  self:                # onde ele está
    path: manifest
  remotes:             # de onde clonar
    - name: zephyrproject-rtos
      url-base:
        https://github.com/zephyrproject-rtos
  projects:
    - name: zephyr
      remote: zephyrproject-rtos
      revision: v4.4.1 # versão fixa
      import:
        name-allowlist: # só estes
          - hal_stm32
          - mcuboot
          - lvgl # ... 10 módulos
```

As três topologias de workspace

T1

o Zephyr é o manifest

```
~/zephyr-sdk-1.0.1/ ← toolchain
~/zephyrproject/ ← west init
├ .west/
├ zephyr/ ← west.yml (v4.4.1)
├ modules/
├ bootloader/
└ .venv/
~/work/app-A/ ← repo git (v4.4.1)
~/work/app-B/ ← repo git (v4.0.0)

zephyr$ git checkout v4.0.0
zephyr$ west update
```

Um Zephyr compartilhado por todas as apps. Bom para aprender e rodar exemplos.

```
west init
```

T2 este projeto

a aplicação é o manifest

```
~/zephyr-sdk-1.0.1/ ← toolchain
~/work/demo-A/ ← repo git
├ .west/
├ manifest/west.yml
├ app/ boards/ tests/
├ zephyr/ (v4.4.1)
├ modules/
├ bootloader/
└ .venv/
~/work/demo-B/ ← repo git
├ .west/
├ manifest/west.yml
├ app/ boards/ tests/
└ zephyr/ (v4.0.0)
```

Cada produto com o seu Zephyr fixo. zephyr/ e modules/ ficam fora do git.

```
west init -l manifest
```

T3

floresta: repo só de manifest

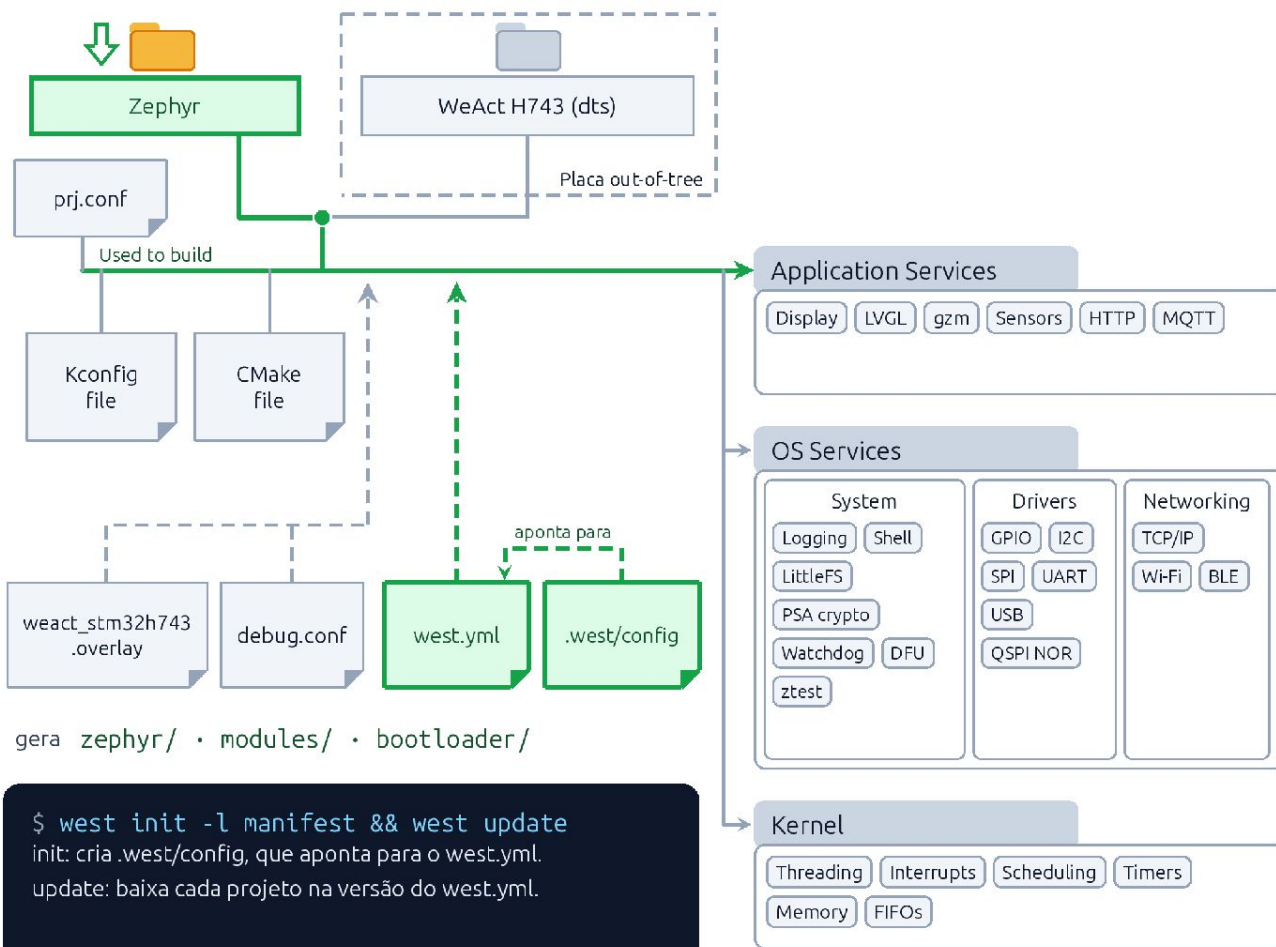
```
~/zephyr-sdk-1.0.1/ ← toolchain
~/work/produtos/
├ .west/
├ manifest/west.yml
├ demo-A/ ← repo git (v4.4.1)
├ demo-B/ ← repo git (v4.4.1)
├ zephyr/ (v4.4.1)
└ modules/
```

Um repositório só com o west.yml junta vários produtos e equipes.

```
west init -m <url>
```

Download do Zephyr via manifest

☒ usado nesta etapa ☐ não usado



Workspace depois do west update

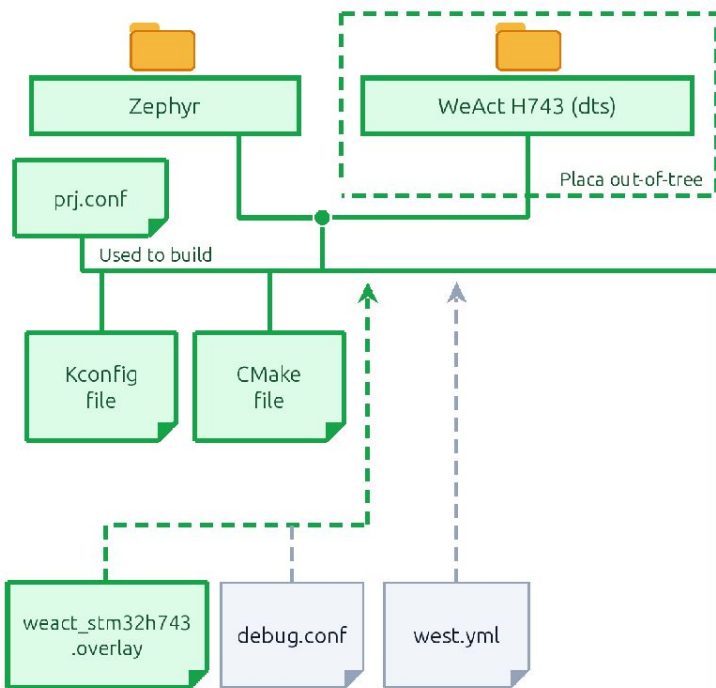
☒ .west/config	west init	✓
☒ manifest/west.yml		✓
app/ · boards/ · gzm/ · tests/ · tools/		
☒ zephyr/	v4.4.1	✓
☒ modules/		✓
☒ hal/cmsis	512cc7e895e8	✓
☒ hal/cmsis_6	30a859f44ef8	✓
☒ hal/stm32	fc11896dd39c	✓
☒ fs/littlefs	8f5ca3478433	✓
☒ lib/gui/lvgl	85aa60d18b3d	✓
☒ lib/picolibc	01254932e8e8	✓
☒ lib/zcbor	9164bd18dcd8	✓
☒ crypto/mbedtls	a3e190fe44c7	✓
☒ crypto/tf-psa-crypto	dc575a2ddcc8	✓
☒ bootloader/mcuboot/	6d3b3d2c38ab	✓

```
$ west init -l manifest && west update
```

init: cria .west/config, que aponta para o west.yml.
update: baixa cada projeto na versão do west.yml.

Build da aplicação

☒ usado nesta etapa ☐ não usado



gera zephyr.elf (sem MCUboot: análise / Renode)

```
$ west build -b weact_stm32h743 -p always \
  -s app -d app/build \
  -- -DBOARD_ROOT=$PWD
Devicetree: .dts + .overlay → zephyr.dts
Kconfig: prj.conf + boards/*.conf → .config
```

Application Services

Display LVGL gsm Sensors HTTP MQTT

OS Services

System

Logging Shell
LittleFS
PSA crypto
Watchdog DFU
ztest

Drivers

GPIO I2C
SPI UART
USB
QSPI NOR

Networking

TCP/IP
Wi-Fi BLE

Kernel

Threading Interrupts Scheduling Timers
Memory FIFOs

Arquivos do projeto

☐ manifest/west.yml

- ☒ app/ ✓
 - ☐ CMakeLists.txt ✓
 - ☒ prj.conf ✓
 - ☐ debug.conf
- ☒ boards/ weact_stm32h743 · native_sim ✓
- ☒ src/ main.c · users_app.c · ui_app.c ✓
- ☒ inc/ *.h ✓
- ☒ gsm/ lib/user_mgr · lib/fs_mgr ✓
- ☒ boards/witte/weact_stm32h743/ ✓
 - ☒ *.dts · _defconfig · board.yml ✓
 - ☒ fs_mgr/ testes do fs_mgr
- ☒ zephyr/ ✓
- ☒ modules/lib/gui/lvgl ✓
- ☒ modules/fs/littlefs ✓
- ☒ modules/crypto/ mbedtls · tf-psa-crypto ✓
- ☒ modules/hal/ stm32 · cmsis_6 ✓

04 • Ztest e Twister

Ztest

- Framework de testes **em C**, nativo do Zephyr
- Compara o valor obtido com o valor esperado
- Válida a lógica: módulos, drivers e bibliotecas
- Roda pelo **Twister**, que lê o **testcase.yaml**
- Mock: funções falsas com **FFF**

ASSERTÇÕES o que acontece quando a condição falha

zassert_* falha e encerra o teste

zexpect_* registra a falha e continua

zassume_* pula o teste (marca skipped)

KCONFIG prj.conf da suíte

CONFIG_ZTEST=y

liga o framework (seleciona CONFIG_TEST)

demo

CONFIG_ZTEST_STACK_SIZE=8192

pilha da thread de teste (padrão: 1024; 2048 em x86/gcov)

demo

CONFIG_ZTEST_FAIL_FAST

para no primeiro teste que falhar

CONFIG_ZTEST_SHUFFLE

sorteia a ordem: acha teste que depende de outro

CONFIG_ZTEST MOCKING

mock nativo do Ztest (aqui usamos o FFF)

Twister

- Executor de testes oficial do Zephyr, em **Python**
- Acha os **testcase.yaml** e monta testes x placas
- Compila, executa e lê o **PASS/FAIL**, em paralelo
- Build rigoroso: **warning vira erro**

ONDE RODA quem decide é a placa do -p

native_sim

no PC: executável Linux, sem hardware

placa real

--device-testing grava e lê a serial

CI

no GitHub do Zephyr, a cada Pull Request

PARÂMETROS west twister

-T tests

onde procurar os testcase.yaml

demo

-p native_sim/native/64

em qual placa rodar (pode repetir)

demo

--board-root .

a nossa placa fica fora do Zephyr

demo

-O twister-out

pasta de saída: twister.json e JUnit

--device-testing

roda na placa real (com --device-serial)

--coverage

cobertura com gcov, no PC

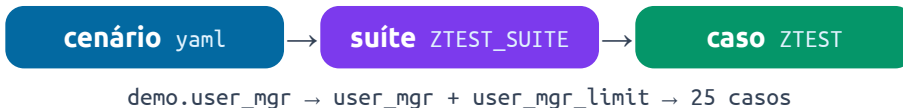
testcase.yaml

tests/user_mgr/testcase.yaml

```
tests:
  demo.user_mgr:          # cenário 1: PC
    platform_allow:
      - native_sim
      - native_sim/native/64
    integration_platforms:
      - native_sim/native/64
    tags: [demo, user_mgr]
  demo.user_mgr.board:    # cenário 2: placa
    platform_allow:
      - weact_stm32h743
    sysbuild: true
    simulation_exclude: [renode]
    timeout: 300
    tags: [demo, user_mgr]
```

O que o arquivo define

Cada nome sob **tests:** é um **cenário**: em quais placas compilar, com quais opções e com quais tags. Os testes em si são os **ZTEST()** do código C.



O -p escolhe o cenário

-p native_sim/native/64 → demo.user_mgr

-p weact_stm32h743 → demo.user_mgr.board

Um projeto de teste, arquivo por arquivo

```
tests/  
  user_mgr/  
    CMakeLists.txt  
    prj.conf  
    testcase.yaml  
    src/main.c · test_limit.c  
  boards/  
    native_sim.overlay  
    weact_stm32h743.overlay  
    sysbuild.conf · sysbuild/
```

uma pasta por projeto de teste · o Twister encontra todos
a biblioteca gzm/user_mgr · roda no PC e na placa
projeto Zephyr + módulo gzm/ (ZEPHYR_EXTRA_MODULES)
ZTEST · LITTLEFS · PSA_CRYPTIO · GZM_USER_MGR
dois cenários: native_sim e a placa
16 testes da lib + 5 do limite de 50 usuários
overlay + conf por placa: só entram no build dela
flash simulada: partição de 1 MB para a LittleFS
LittleFS na NOR SPI · console na UART4
MCUboot também no teste da placa

```
$ west twister --board-root . -T tests/user_mgr -p native_sim/native/64
```

novo teste = nova pasta

Rodando comando Twister

1 Testes criados

PC · PLACA

user_mgr

Usuários: CRUD, login e o limite de 50 (22 testes)

PC · PLACA

fs_mgr

Arquivos: salvar, ler e a queda de energia (20 testes)

SÓ NO PC

user_mgr_mock

Mock com FFF: falhas de armazenamento sob demanda (4 testes)

2 O comando

```
# Roda todos os testes, com cobertura
$ west twister --board-root . -T tests \
  -p native_sim/native/64 --coverage \
  --coverage-tool gcovr \
  --coverage-basedir gzm
```

SUITE PASS [user_mgr] 17/17

SUITE PASS [user_mgr_limit] 5/5

SUITE PASS [fs_mgr] 20/20

SUITE PASS [user_mgr_mock] 4/4

3 of 3 configurations passed

46 of 46 test cases passed (100.00%)

PC · native_sim · ~40 s com gcov

ztest na prática: a regra da senha

```
ZTEST(user_mgr, test_password_rules)
{
    /* sem senha e senha vazia: recusadas */
    zassert_equal(user_mgr_add(1, "Joao", "", NULL, ACC_LEVEL_USER), -EINVAL);
    zassert_equal(user_mgr_add(1, "Joao", "", "", ACC_LEVEL_USER), -EINVAL);

    /* senha válida: aceita */
    zassert_ok(user_mgr_add(1, "Joao", "", "Senha01", ACC_LEVEL_USER));
}
```

retorno 0 = ok · erro = -errno

gcov: o que o teste realmente executou

```
347 292 static int user_mgr_validate_password(const char *pwd, size_t *len_out)
348 {
349 292     size_t n = 0;
350
351 ▶ 2/2 292     if (!pwd) {
352         1     return -EINVAL;
353     }
354 291     n = user_mgr_strlen(pwd, USER_MGR_PASSWORD_MAX_LEN + 1);
355 ▶ 3/4 291     if (n == 0 || n > USER_MGR_PASSWORD_MAX_LEN) {
356         2     return -EINVAL;
357     }
358 ▶ 1/2 289     if (len_out) {
359         289     *len_out = n;
360     }
361 289     return 0;
362 }
```

verde executada · amarelo ramo parcial · vermelho nunca executou

FLASH 140 KB → 820 KB

5,9× maior com gcov

RAM 46 KB → 335 KB

7,3× maior com gcov

- GCC `--coverage`: um contador por ramo
- `.gcno` + `.gcda` → o **gcovr** gera o HTML
- `CONFIG_COVERAGE=y`, ligado pelo **Twister**
- Placa: `CONFIG_FORCE_COVERAGE` → **UART**

O PREÇO só em build de teste

-00

código maior e mais lento

contadores

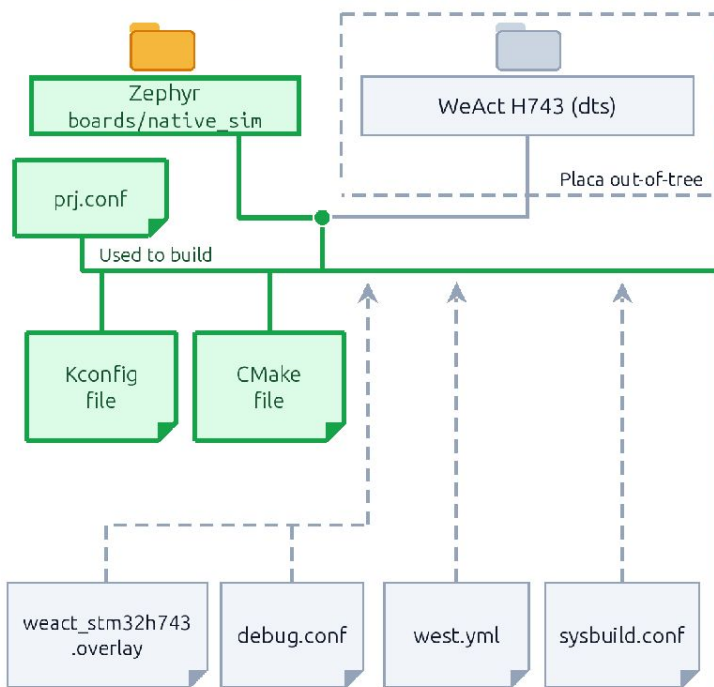
~164 KB + ~80 KB de metadados

produção

nunca: só em build de teste

Twister + ztest no PC (native_sim)

☒ usado nesta etapa ☐ não usado



gera twister-out/ • 46/46 casos

```
$ west twister --board-root . -T tests \
  -p native_sim/native/64 \
  -O twister-out
```

native_sim é uma placa do próprio Zephyr.
Testa a gzm/: user_mgr (real e mock FFF) e fs_mgr.

Application Services

Display LVGL **gzm** Sensors HTTP MQTT

OS Services

System

Logging Shell
LittleFS
PSA crypto
Watchdog DFU
ztest

Drivers

GPIO I2C
SPI UART
USB
QSPI NOR

Networking

TCP/IP
Wi-Fi BLE

Kernel

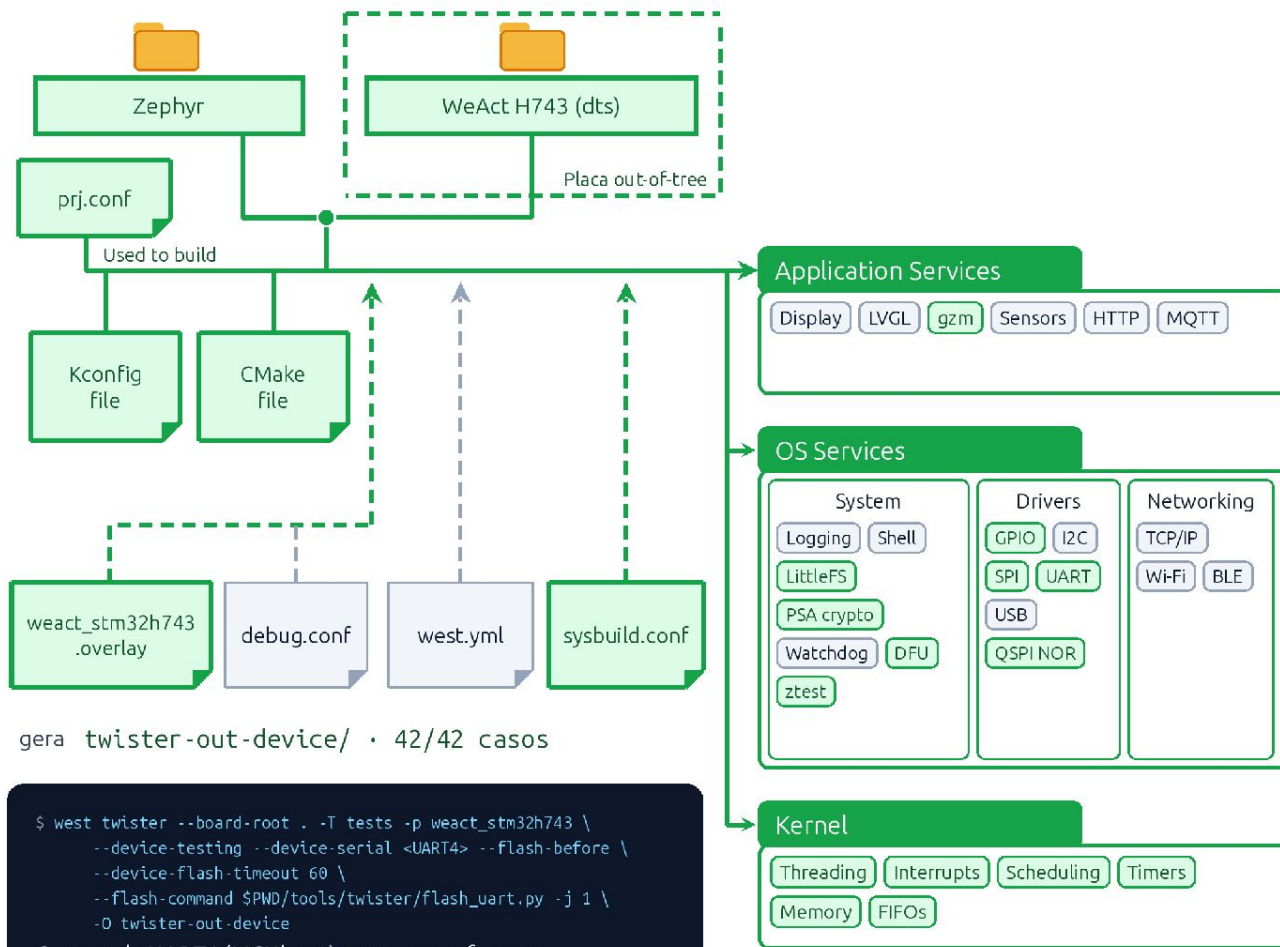
Threading Interrupts Scheduling Timers
Memory FIFOs

Arquivos do projeto

- ☐ manifest/west.yml
- ☒ app/
 - ☐ CMakeLists.txt
 - ☐ prj.conf
 - ☐ debug.conf
 - ☐ VERSION
 - ☐ sysbuild.conf
 - ☒ sysbuild/ mcuboot.conf • .overlay
 - ☒ boards/ weact_stm32h743 • native_sim
 - ☒ src/ main.c • users_app.c • ui_app.c
 - ☒ inc/ *.h
- ☒ gzm/ lib/user_mgr • lib/fs_mgr ✓
- ☒ boards/witte/weact_stm32h743/
 - ☐ *.dts • _defconfig • board.yml
- ☒ tests/ ✓
 - ☒ user_mgr/ testcase.yaml • prj.conf ✓
 - ☒ src/*.c • boards/ • sysbuild/ ✓
 - ☒ fs_mgr/ testcase.yaml • prj.conf ✓
 - ☒ src/*.c • boards/ • sysbuild/ ✓
 - ☒ user_mgr_mock/ mock FFF ✓
- ☒ tools/keys/ *.pem
- ☐ tools/twister/flash_uart.py
- ☐ tools/flasher/fw_update.py
- ☒ zephyr/ ✓
 - ☒ modules/lib/gui/lvgl ✓
 - ☒ modules/fs/littlefs ✓
 - ☒ modules/crypto/ mbedtls • tf-psa-crypto ✓
 - ☒ modules/hal/ stm32 • cmsis_6
 - ☒ bootloader/mcuboot/ • lib/zcbor

Twister na placa real

☒ usado nesta etapa ☐ não usado



Arquivos do projeto

- ☐ manifest/west.yml
- ☒ app/
 - ☐ CMakeLists.txt
 - ☐ prj.conf
 - ☐ debug.conf
 - ☐ VERSION
 - ☐ sysbuild.conf
 - ☒ sysbuild/ mcuboot.conf • .overlay
 - ☒ boards/ weact_stm32h743 • native_sim
 - ☒ src/ main.c • users_app.c • ui_app.c
 - ☒ inc/ *.h
- ☒ gzm/ lib/user_mgr • lib/fs_mgr ✓
- ☒ boards/witte/weact_stm32h743/ ✓
- ☒ *.dts • _defconfig • board.yml ✓
- ☒ tests/ ✓
 - ☒ user_mgr/ testcase.yaml • prj.conf ✓
 - ☒ src/*.c • boards/ • sysbuild/ ✓
 - ☒ fs_mgr/ testcase.yaml • prj.conf ✓
 - ☒ src/*.c • boards/ • sysbuild/ ✓
 - ☐ user_mgr_mock/ mock FFF
- ☒ tools/keys/ *.pem ✓
- ☒ tools/twister/flash_uart.py ✓
- ☐ tools/flasher/fw_update.py
- ☒ zephyr/ ✓
 - ☒ modules/lib/gui/lvgl
 - ☒ modules/fs/littlefs ✓
 - ☒ modules/crypto/ mbedtls • tf-psa-crypto ✓
 - ☒ modules/hal/ stm32 • cmsis_6 ✓
 - ☒ bootloader/mcuboot/ • lib/zcbor ✓

05 • Bootloader

MCUboot

- Bootloader seguro e **open source**
- Gerência processo de update usando arquivo **.bin**
- Utiliza e válida **assinatura** e **criptografia** da imagem
- Suporte a **Rollback** em caso de falha
- Configuração centralizada do MCUboot e app via `sysbuild.conf`

COMO A IMAGEM CHEGA três caminhos

app + MCUmgr

UART, USB, BLE ou rede

serial recovery

SMP direto no MCUboot

Gravador

gravação pelo SWD (west flash)

MCUboot: os Kconfig que mais importam

MODO DE TROCA sysbuild.conf

SB_CONFIG_MCUBOOT_MODE_SWAP_SCRATCH

troca com rollback; a antiga vira backup no slot1

[demo](#)

SB_CONFIG_MCUBOOT_MODE_SWAP_USING_MOVE

igual, sem scratch; slots com setores iguais

SB_CONFIG_MCUBOOT_MODE_OVERWRITE_ONLY

copia por cima: sem backup, sem rollback

ROBUSTEZ mcuboot.conf

CONFIG_BOOT_WATCHDOG_TIMEOUT_MS=20000

app travado → reset → rollback

[demo](#)

CONFIG_BOOT_WATCHDOG_FEED

alimenta o watchdog durante a troca

[demo](#)

CONFIG_BOOT_BOOTSTRAP

slot0 vazio? instala a partir do slot1

LOG DO MCUBOOT mcuboot.conf

CONFIG_MCUBOOT_LOG_LEVEL_INF

nível do log do boot (ou_DBG) · na demo: CONFIG_LOG=n

SEGURANÇA sysbuild.conf / mcuboot.conf

SB_CONFIG_BOOT_SIGNATURE_TYPE_ECDSA_P256

só roda imagem assinada

[demo](#)

SB_CONFIG_BOOT_ENCRYPTION

imagem cifrada no slot1

[demo](#)

CONFIG_MCUBOOT_DOWNGRADE_PREVENTION

recusa versão menor que a atual

CONFIG_BOOT_VALIDATE_SLOT0

revalida o slot0 a cada boot

[demo](#)

RECUPERAÇÃO E STATUS mcuboot.conf

CONFIG_MCUBOOT_SERIAL

recovery pela UART com mcumgr

[demo](#)

CONFIG_BOOT_SERIAL_WAIT_FOR_DFU

janela após o reset, sem botão

[demo](#)

CONFIG_BOOT_SERIAL_ENTRANCE_GPIO

entra em recovery segurando um botão

CONFIG_MCUBOOT_INDICATION_LED

LED aceso enquanto está em recovery

[demo](#)

Mapeamento de Memória

Flash interna

STM32H743 · 0x0800_0000

MCUboot

128 KB

0x0800_0000 - 0x0801_FFFF

slot0 · atual

896 KB

0x0802_0000 - 0x080F_FFFF

scratch

256 KB

0x0810_0000 - 0x0813_FFFF

NOR QSPI

W25Q64 externa · offset

slot1 · novo

896 KB

0x00_0000 - 0x0D_FFFF

NOR SPI

W25Q64 externa · offset

LittleFS

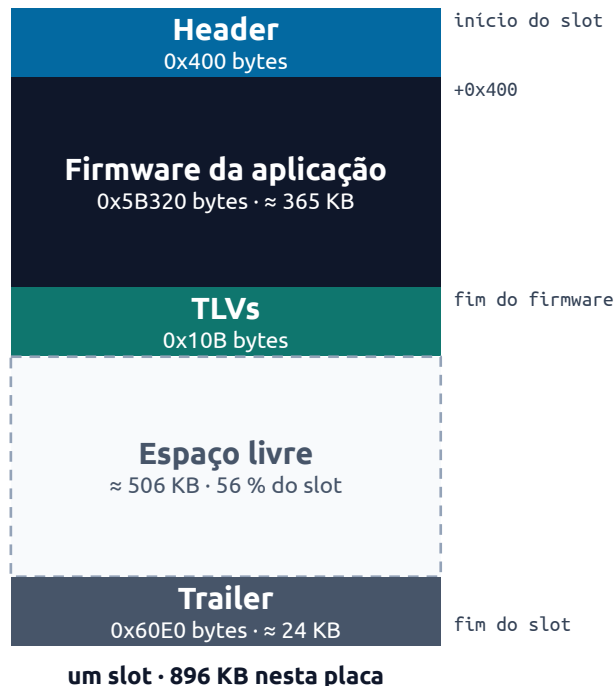
8 MB

0x00_0000 - 0x7F_FFFF

- **MCUboot** flash interna · 128 KB
Valida, decifra e troca as imagens. Não é atualizado em campo.
- **slot0 · atual** flash interna · 896 KB
Onde o app roda. Header, código e trailer de controle.
- **scratch** flash interna · 256 KB
Área temporária da troca, setor a setor. Com o swap status do trailer, a troca retoma após queda de energia.
- **slot1 · novo** NOR QSPI · 896 KB
Recebe a imagem nova, cifrada. Depois da troca, guarda a antiga: é o backup.
- **LittleFS** NOR SPI · 8 MB
Dados do app. Pode guardar um .bin baixado por Wi-Fi, Ethernet ou USB: o app copia para o slot1 e, após o reset, o MCUboot instala.

Endereços do **zephyr.dts**: na flash interna é o endereço real da CPU; nas NOR externas é o offset dentro do chip.

Estrutura Interna de um Slot do MCUboot



início do slot

+0x400

fim do firmware

fim do slot

Header magic, versão (o app/VERSION) e tamanho do firmware

Firmware o binário da aplicação

TLVs hash SHA-256, assinatura e a chave AES cifrada

Espaço livre o resto do slot: margem para o firmware crescer

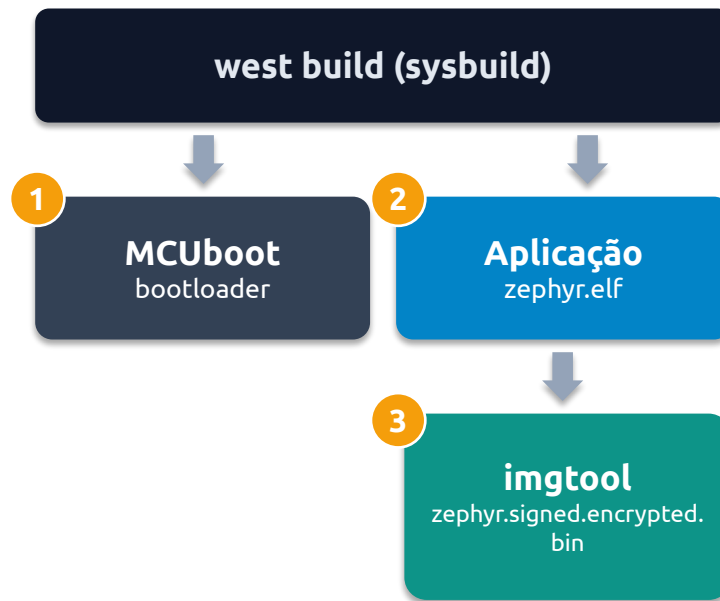
Trailer estado e progresso da troca: em teste ou confirmada

Processo de Compilação Unificada com Sysbuild

```
# app/sysbuild.conf: vale para as etapas 1 e 2
SB_CONFIG_BOOTLOADER_MCUBOOT=y
SB_CONFIG_MCUBOOT_MODE_SWAP_SCRATCH=y
SB_CONFIG_BOOT_SIGNATURE_TYPE_ECDSA_P256=y
SB_CONFIG_BOOT_ENCRYPTION=y

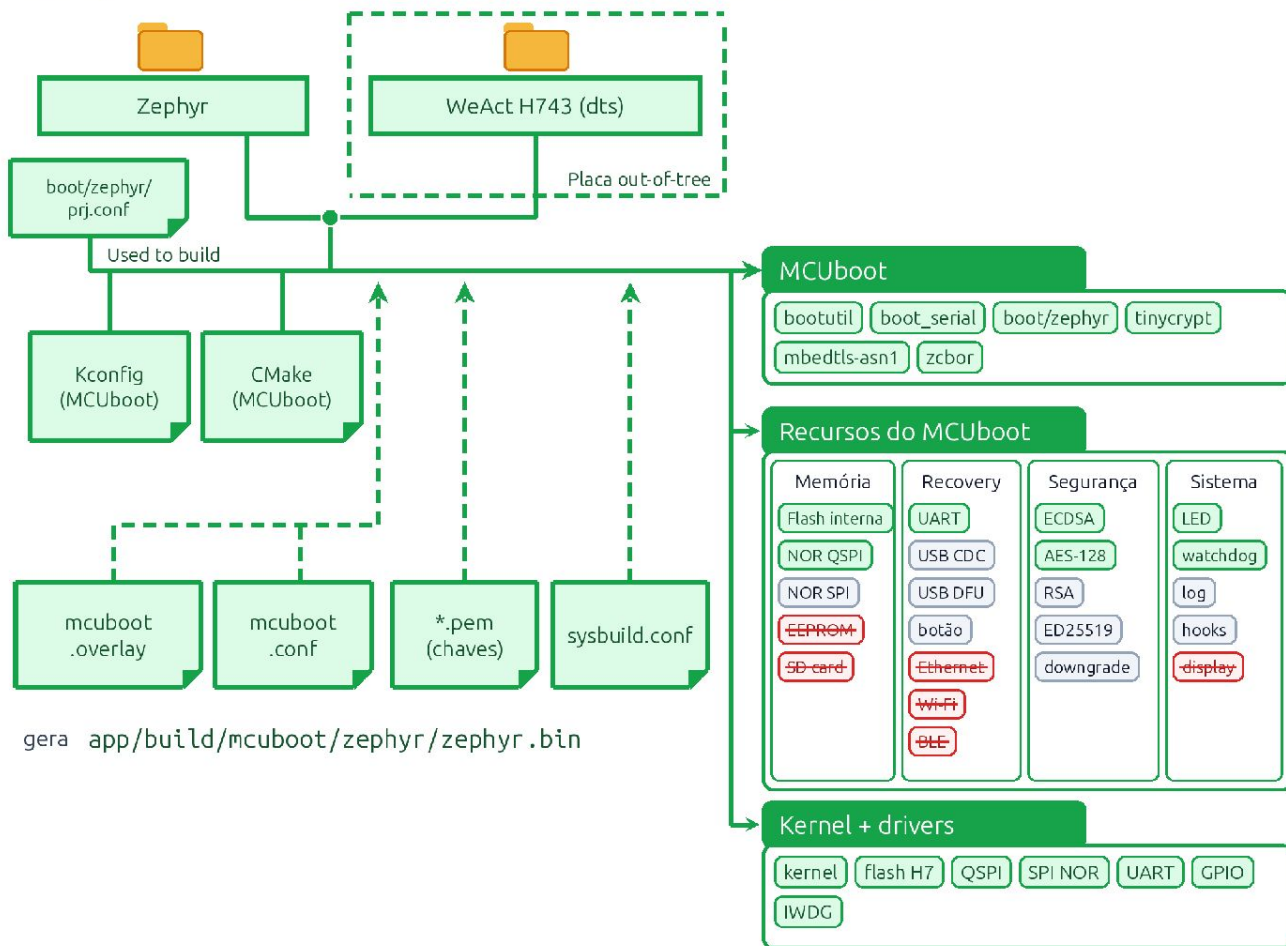
# um comando faz as 3 etapas
$ west build -b weact_stm32h743 --sysbuild \
  -p always -s app -d app/build \
  -- -DBOARD_ROOT=$PWD
```

sysbuild: MCUboot + app em um único processo de build
Imagem cifrada na NOR externa (ECIES P-256 + AES-128)



Etapa 1: build do firmware do MCUboot

☒ usado / habilitado ☐ suportado, desligado
☐ sem suporte no MCUboot

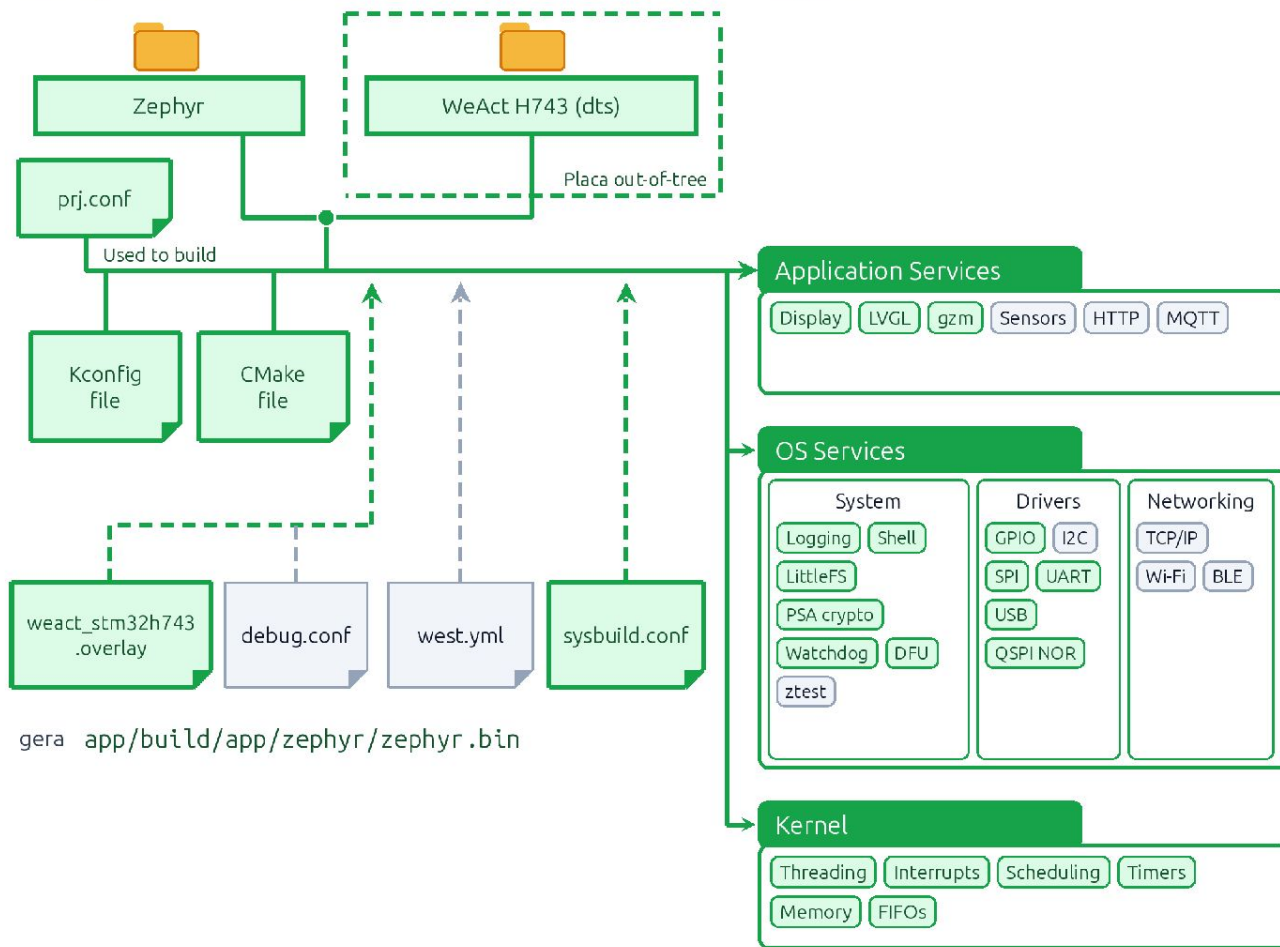


Arquivos do build do MCUboot

- ☐ manifest/west.yml
- ☒ app/ src/ · inc/ · prj.conf · boards/ ...
 - ☒ sysbuild.conf
 - ☒ sysbuild/mcuboot.conf
 - ☒ sysbuild/mcuboot.overlay
 - ☒ boards/witte/weact_stm32h743/
 - ☒ weact_stm32h743.dts · _defconfig
 - ☒ tools/keys/
 - ☒ demo-ecdsa-p256.pem
 - ☒ demo-encryption-p256.pem
 - ☒ bootloader/mcuboot/
 - ☒ boot/zephyr/ main.c · Kconfig
 - ☒ boot/bootutil/ loader.c
 - ☒ swap_scratch.c · encrypted.c ...
 - ☒ boot/boot_serial/ boot_serial.c
 - ☒ ext/tinycrypt/ · mbedtls-asn1/
 - ☒ zephyr/
 - ☒ kernel/ · drivers/flash · spi
 - ☒ drivers/serial · gpio · watchdog
 - ☒ modules/
 - ☒ hal/stm32 · hal/cmsis_6
 - ☒ lib/zcbor (CBOR do SMP)
- ☒ lvgl · littlefs · mbedtls (só app)
- ☒ tests/ · tools/flasher/

Etapa 2: build do firmware da aplicação

☒ usado nesta etapa ☐ não usado



Arquivos do projeto

- ☐ manifest/west.yml
- ☒ app/
 - ☐ CMakeLists.txt
 - ☒ prj.conf
 - ☐ debug.conf
 - ☒ VERSION
 - ☒ sysbuild.conf
 - ☒ sysbuild/ mcuboot.conf · .overlay
 - ☒ boards/ weact_stm32h743 · native_sim
 - ☒ src/ main.c · users_app.c · ui_app.c
 - ☒ inc/ *.h
- ☒ gzm/ lib/user_mgr · lib/fs_mgr
- ☒ boards/witte/weact_stm32h743/
 - ☒ *.dts · _defconfig · board.yml
- ☐ tests/
 - ☐ user_mgr/ testcase.yaml · prj.conf
 - ☐ src/*.c · boards/ · sysbuild/
 - ☐ fs_mgr/ testes do fs_mgr
 - ☐ storage_nor/ · boot_info/
 - ☐ user_mgr_mock/ mock FFF
- ☒ tools/keys/ *.pem
- ☐ tools/twister/flash_uart.py
- ☐ tools/flasher/fw_update.py
- ☒ zephyr/
- ☒ modules/lib/gui/lvgl
- ☒ modules/fs/littlefs
- ☒ modules/crypto/ mbedtls · tf-psa-crypto
- ☒ modules/hal/ stm32 · cmsis_6
- ☒ bootloader/mcuboot/ · lib/zcbor

Etapa 3: imgtool assina e cifra a aplicação

```
# o build da app chama sozinho, depois da etapa 2
$ python3 bootloader/mcuboot/scripts/imgtool.py sign \
  --version 2.0.0 --header-size 0x400 \
  --slot-size 917504 --align 32 \
  --key tools/keys/demo-ecdsa-p256.pem \
  --encrypt tools/keys/demo-encryption-p256.pem \
  app/build/app/zephyr/zephyr.bin \
  zephyr.signed.encrypted.bin

$ python3 bootloader/mcuboot/scripts/imgtool.py \
  dumpinfo zephyr.signed.encrypted.bin
flags:  ENCRYPTED_AES128 (0x4)
version: 2.0.0+0
  type: SHA256 (0x10)
  type: KEYHASH (0x1)
  type: ECDSASIG (0x22)
  type: ENCEC256 (0x32)
```

+277 B: o zephyr.bin já reserva 0x400 B de header (zeros no início); o imgtool preenche o header e anexa os TLVs no fim.

AS CHAVES geradas pelo genkeys.sh, fora do git

assinatura

privada no PC assina; pública no MCUboot verifica

criptografia

pública no PC cifra; privada no MCUboot decifra

SAI 4 arquivos; estes 3 importam

.signed.encrypted.bin

serial → slot1

.signed.hex

west flash → slot0

.signed.bin

assinado, sem cifra

Produção: trave a flash com RDP nível 1: a privada de cifra mora no MCUboot.

06 • Atualização em campo

MCUmgr e SMP: quem é quem

QUEM É QUEM

MCUmgr

o sistema: o cliente pede, o servidor executa

SMP

a mensagem: header de 8 B + corpo CBOR

transporte

UART, USB, BLE ou UDP

NESTE PROJETO

- PC: `fw_update_gui.py` usa o **smpclient**
- Placa: **MCUboot** em serial recovery, na UART4
- O app **não tem** MCUmgr
- MCUboot atende só: `reset` • `list` • `test` • `confirm` • `upload`

CLIENTE • no PC

smpclient

`fw_update_gui.py`



SMP pela UART4
pedido → resposta, cada um um pacote

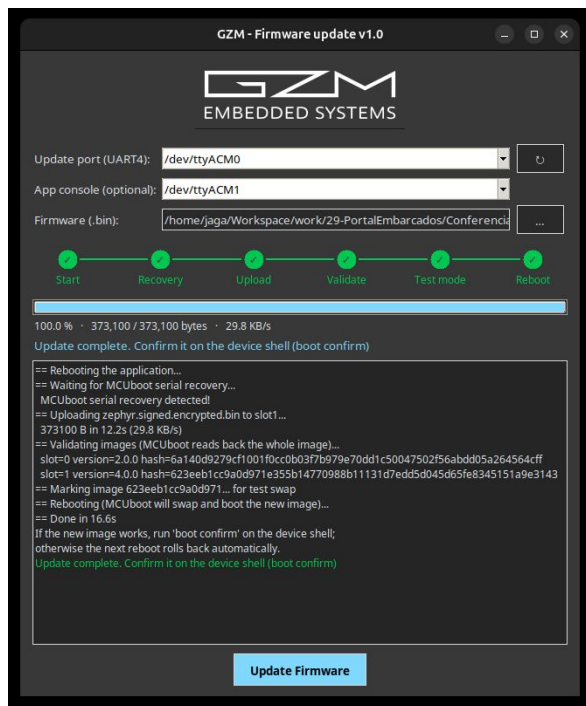
SERVIDOR • na placa

MCUboot

serial recovery

Resumindo: **MCUmgr** é o sistema (cliente + servidor). **SMP** é a língua que os dois falam.

Software de update



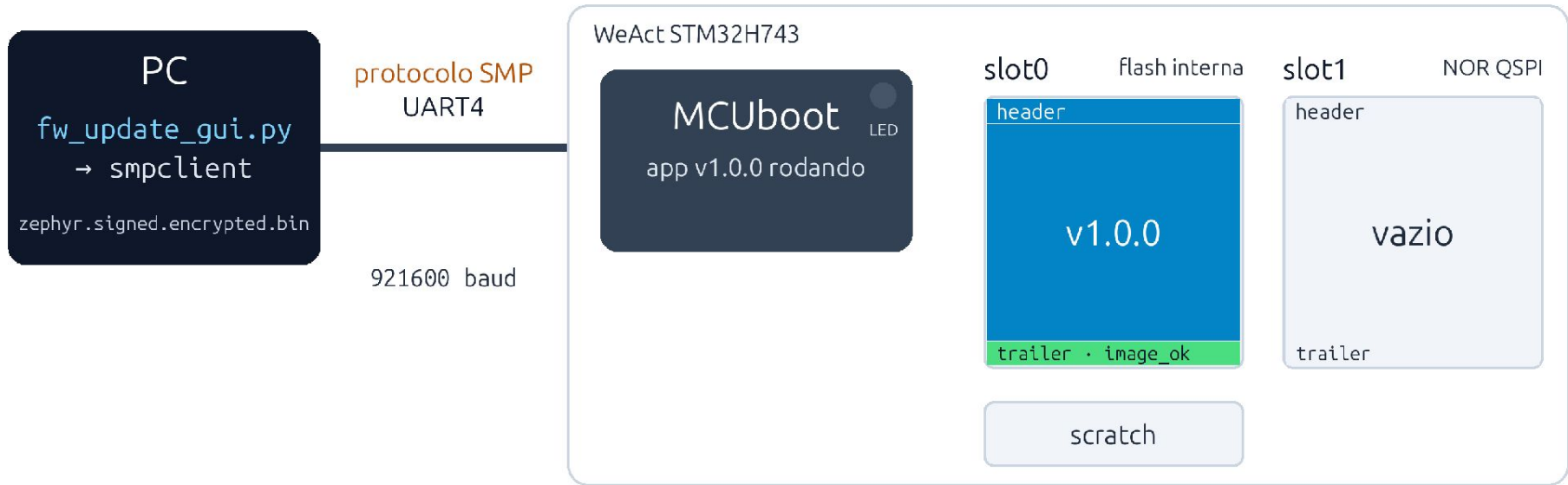
O que ele faz, em sequência · um clique em Update Firmware

- 1 **Reinicia a placa**
pelo console USB do app (opcional)
- 2 **Espera o MCUboot**
janela de 5 s do serial recovery
- 3 **Envia a imagem**
smplclient: o .bin vai para o slot1 (NOR)
- 4 **Confere**
smplclient: lê o hash da imagem nova
- 5 **Marca para teste**
smplclient: troca no próximo boot
- 6 **Reinicia**
o MCUboot valida, troca e inicia

Itens 1 a 6: os círculos da tela, na mesma ordem.

README · cap. 07: a mesma coisa pela linha de comando (fw_update.py)

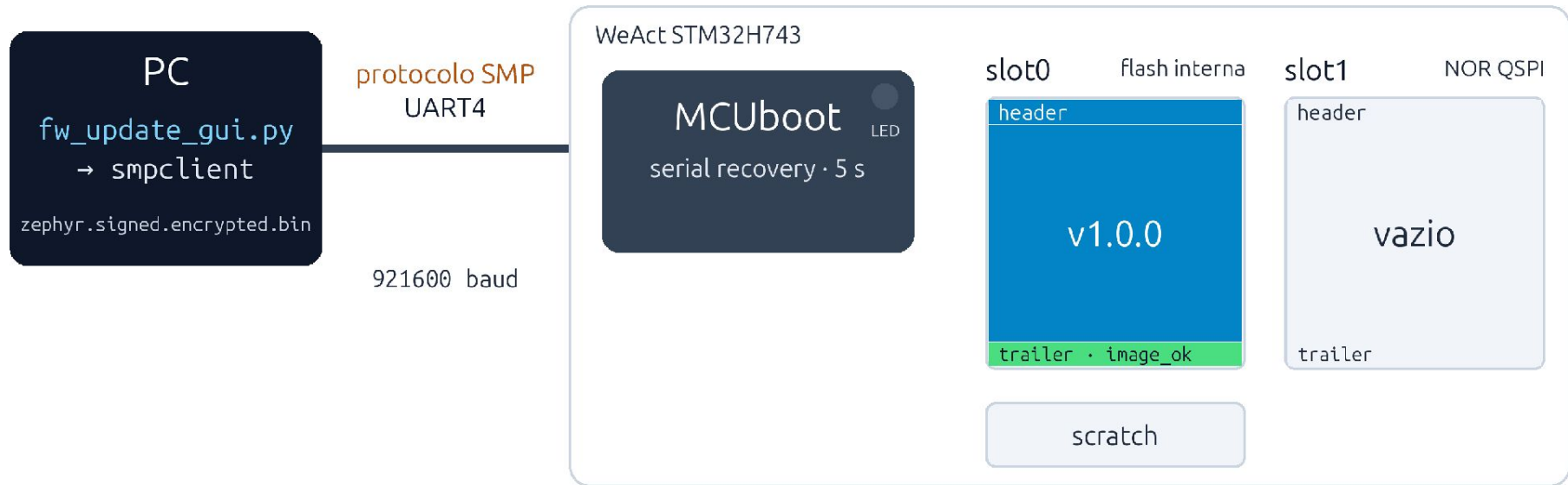
Antes: app v1.0.0 rodando do slot0



```
slot0 v1.0.0  confirmado  ·  slot1 vazio
```

O MCUboot só aceita comandos na UART durante a janela de recovery.

1 "boot reboot" pela USB abre a janela de recovery

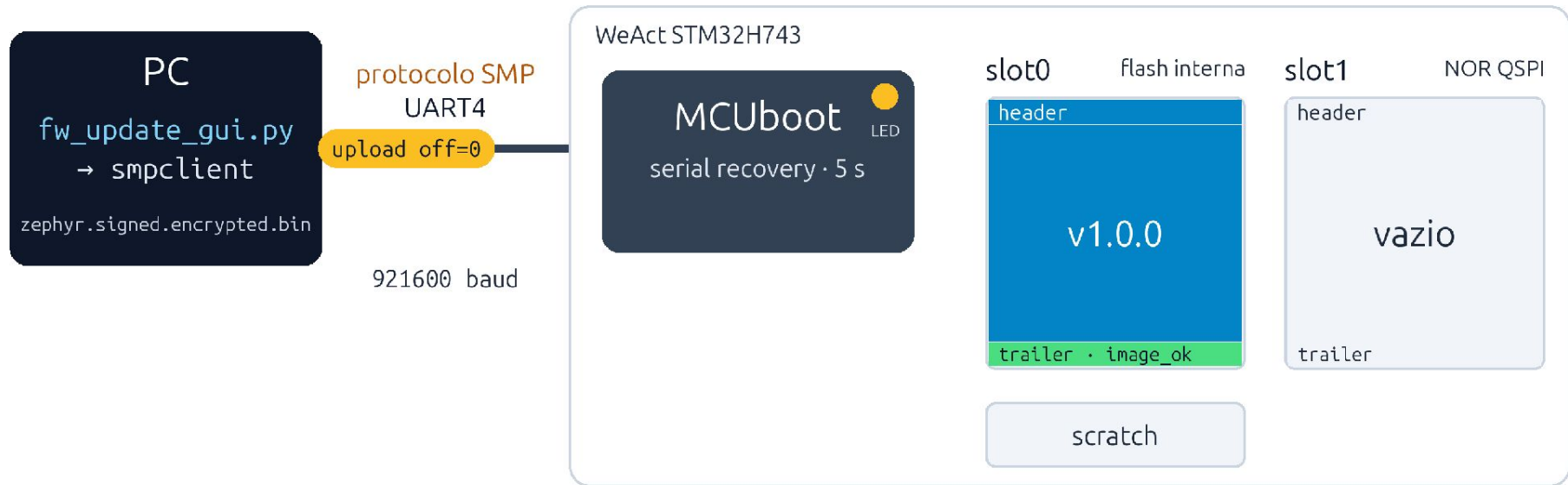


```
atualizador envia "boot reboot" no console USB do app
```

O MCUboot sobe e espera ~5 s por um pacote SMP na UART4.

```
LED aceso = janela de recovery aberta (CONFIG_MCUBOOT_INDICATION_LED)
```

2 1º pacote SMP: "image upload" com off = 0



PROTOCOLO SMP (Simple Management Protocol) · transporte serial do MCUmgr

SMP na serial `06 09` `base64 .....` `\n`

linhas de até 128 B · continuação começa com 04 14

pacote SMP `len 2B` `header SMP 8B` `CBOR` `CRC16`

base64 → bytes · o CRC16 confere tudo

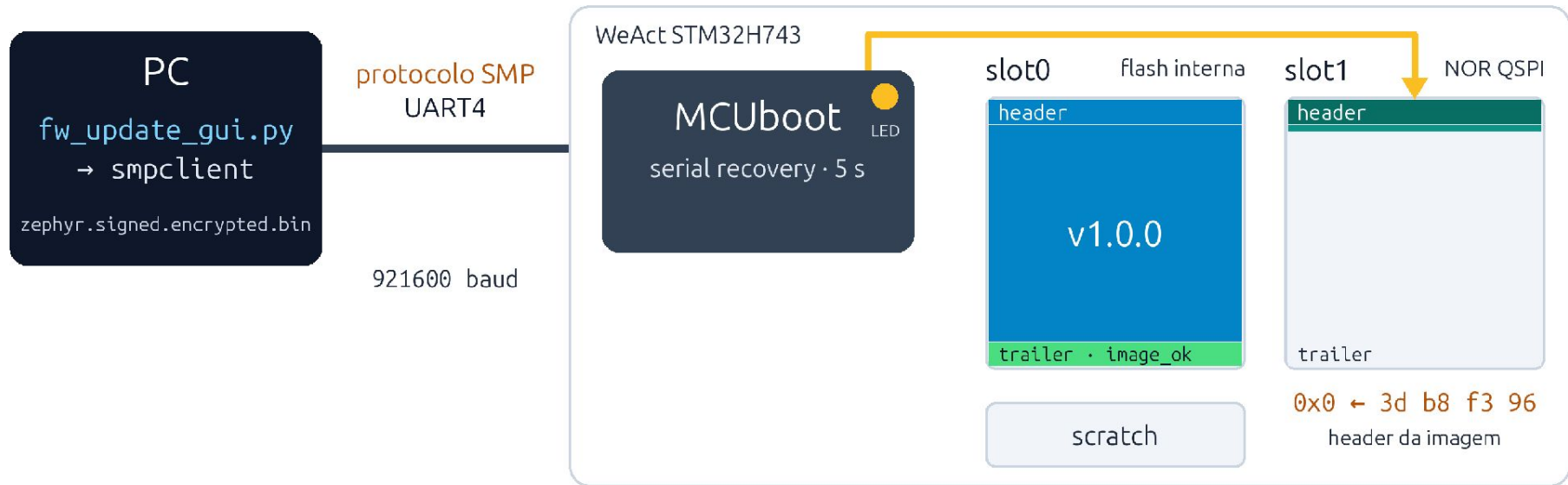
header SMP `op=2 write · group=1 image · id=1 upload · seq`

comando "image upload"

corpo SMP `{ image: 2, off: 0, len: 366444, data: 3d b8 f3 96 ... }`

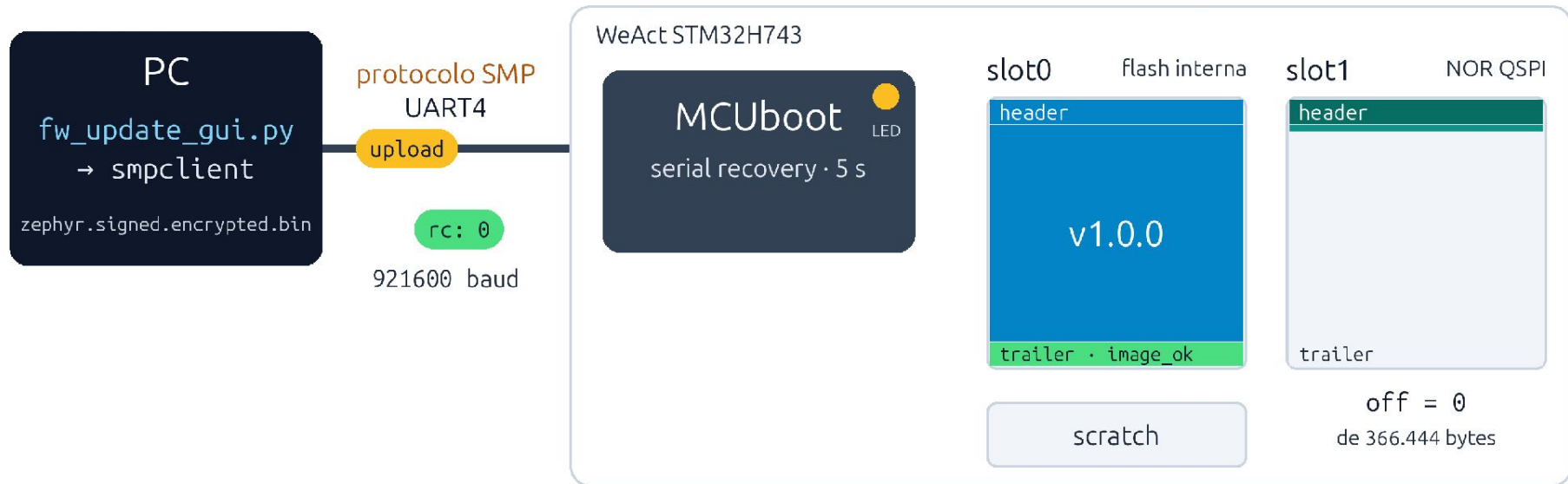
1º pedaço do arquivo

4 image = 2 → slot1 · off = 0 → começa do zero



```
image = 2 → slot1 (MCUBOOT_SERIAL_DIRECT_IMAGE_UPLOAD)
off = 0 → zera o upload; confere len ≤ tamanho do slot
data → gravado no offset 0x0 do slot1 = o header
3d b8 f3 96 = 0x96f3b83d (little endian): o magic do MCUboot
```

5 Resposta { rc: 0, off } → o PC manda o próximo pedaço



protocolo SMP

upload

 $\Gamma C: \emptyset$

921600 baud

WeAct STM32H743

MCUboot

serial recovery · 5 s

slot0

flash interna

header

v1.0.0

```
trailer . image_ok
```

scratch

slot1

NOR QSPI

header

trailer

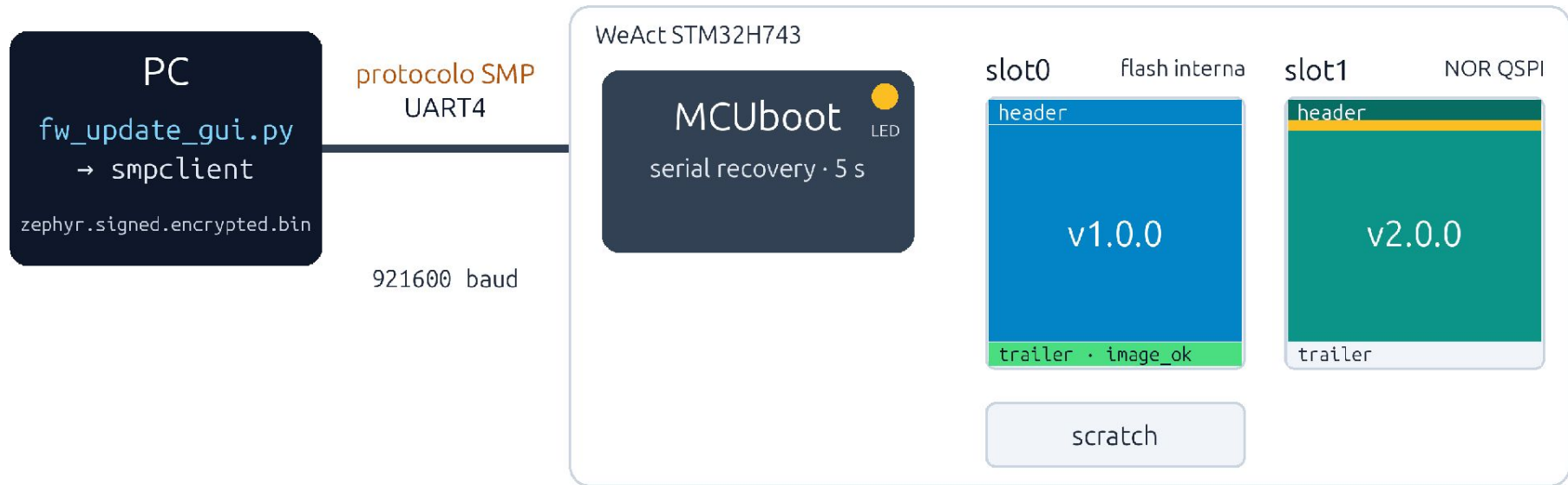
$$\text{off} = 0$$

de 366.444 bytes

```
PC → { off: N, data: ... }
```

Setor a setor, a NOR é apagada e gravada (BOOT_ERASE_PROGRESSIVELY).

6 image list: o MCUboot valida o slot1 e devolve o hash do TLV



protocolo SMP
UART4

WeAct STM32H743

slot0

slot1

header

```
trailer . image_ok
```

scratch

header

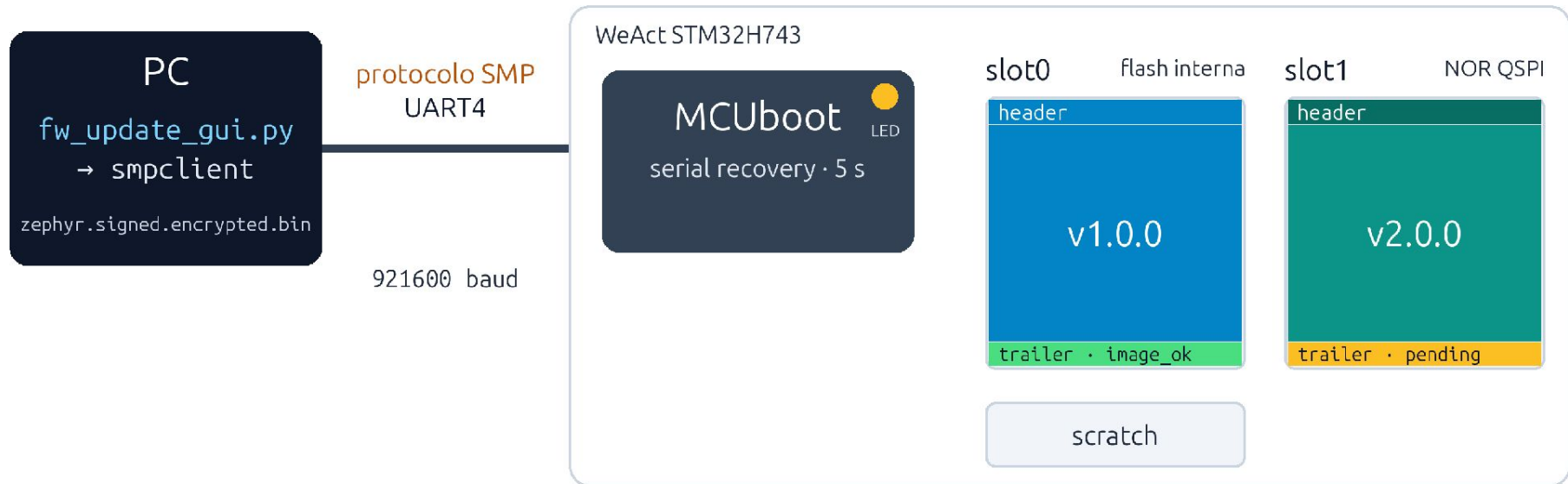
trailer

PC → image list

```
slot=1  2.0.0  flags:
```

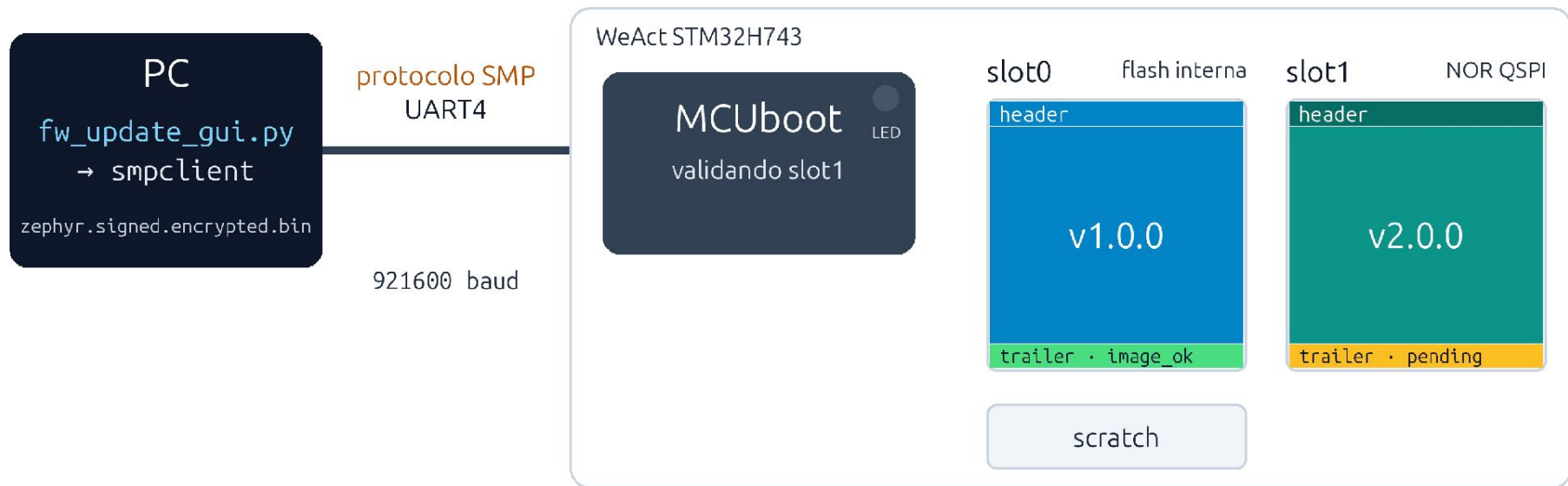
```
hash: fa6f8f0e3a516810...
```

7 image test <hash>: grava o trailer do slot1 → pending



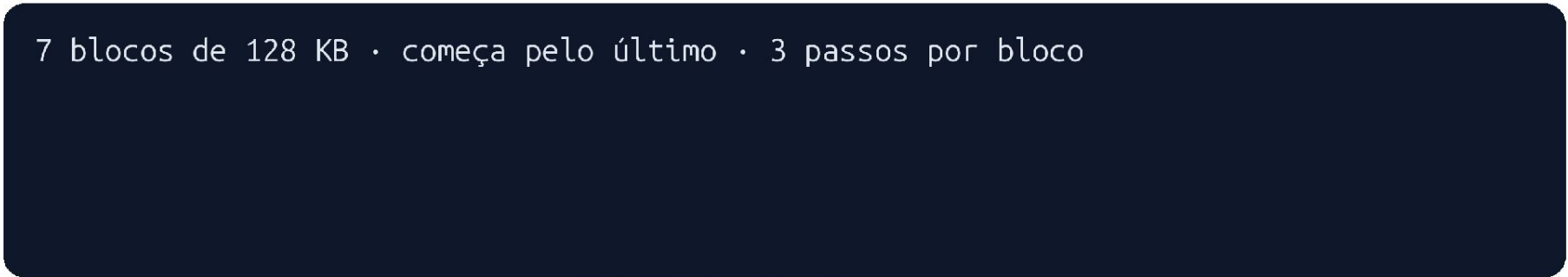
```
PC → image test fa6f8f0e...
  slot=0  1.0.0  flags: confirmed
  slot=1  2.0.0  flags: pending
0 trailer diz ao MCUboot: "troque no próximo boot, em modo teste".
```

8 Reset: o MCUboot verifica o slot1 antes de trocar

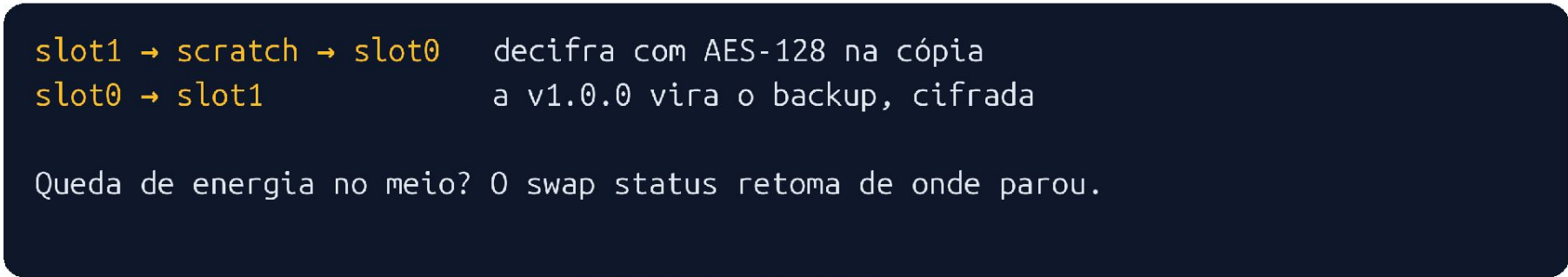


```
lendo a imagem do slot1...
```

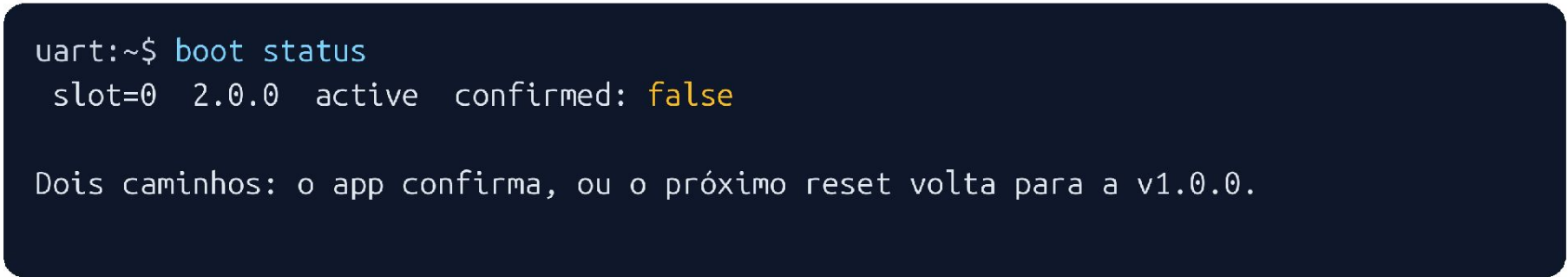
9



10



11



12



921600 baud

MCUboot
app v2.0.0 confirmado

Diagram illustrating the layout of a 10MB file:

- Header (blue bar): 1MB
- Trailer (green bar): 1MB
- Image Area (orange bar): 8MB
 - Image OK (green bar): 1MB
 - Image Error (orange bar): 7MB

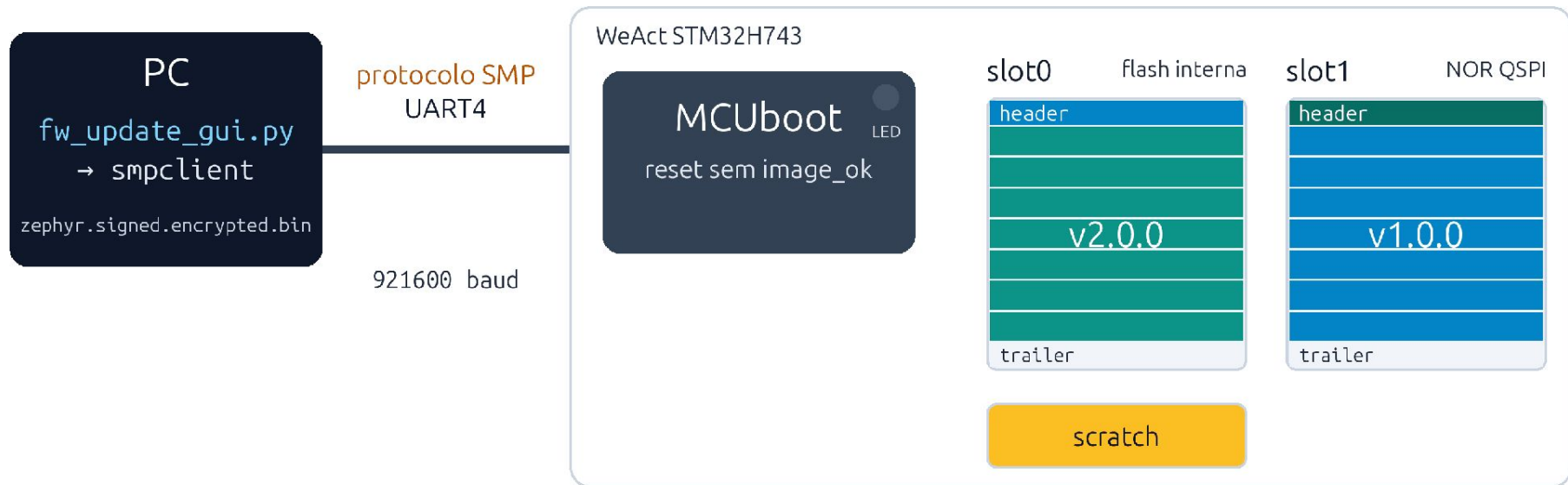
The version number **v2.0.0** is displayed in the center of the image area.

scratch

The diagram illustrates a packet structure. It consists of a vertical stack of colored rectangles. The top rectangle is green and labeled 'header'. Below it are five blue rectangles; the middle one is labeled 'v1.0.0'. The bottom rectangle is light blue and labeled 'trailer'.

```
uart:~$ boot confirm
Firmware image confirmed successfully.
boot_write_img_confirmed() grava image_ok no trailer do slot0
Nos próximos resets, fica na v2.0.0.
```

13 Não confirmou: o MCUboot volta para a v1.0.0



reset ou watchdog (20 s) sem `image_ok` → o MCUboot desfaz a troca

Em produto: confirme só depois de um autoteste (sensor, display, comunicação).

07 • Robot Framework

Robot Framework

- Framework open source de testes, em **Python**
- Teste escrito com **palavras-chave** legíveis
- Fala com o produto: **serial**, HTTP, SSH
- **report.html** vira evidência de cada versão

ONDE É USADO testa o produto pronto, como caixa-preta

aceitação

testes de sistema na indústria

Twister

harness: robot

Renode

os testes do Renode são em Robot

```
# tests/robot/suites/acceptance.robot
*** Test Cases ***
04 An update that is not confirmed rolls back
    ${r}= Update Firmware ${IMG_V200}
    Should Be Equal As Integers ${r.rc} 0
    Open Board Shell
    Running Version Should Be 2.0.0 test
    Reboot Board
    Running Version Should Be 1.0.0 confirmed
```

```
# resources/board.resource
Library SerialLibrary # shell USB
Library Process       # fw_update.py
```

Robot: uma biblioteca por interface

Pronta no PyPI, embutida ou feita em Python

Serial / shell	SerialLibrary	pip
CLI e scripts	Process	embutida
SSH / SFTP	SSHLibrary	pip
Telnet	Telnet	embutida
HTTP / REST	RequestsLibrary	pip
MQTT	MQTTLibrary	pip
Interface web	Browser	pip
Modbus RTU / TCP	pymodbus	wrapper
CAN / SocketCAN	python-can	wrapper
TCP / UDP	socket	wrapper
Bancada (SCPI)	pyvisa	wrapper
Placa simulada	renode-test	Renode

pip robotframework-* no PyPI · wrapper classe Python, ao lado

Não tem pronta? Uma classe Python vira keyword

```
# CanLibrary.py
import can

class CanLibrary:
    ROBOT_LIBRARY_SCOPE = "GLOBAL"

    def __init__(self, channel="can0", interface="socketcan"):
        self.bus = can.Bus(channel=channel, interface=interface)

    def send_can_frame(self, can_id, data):
        self.bus.send(can.Message(arbitration_id=int(can_id, 16),
                                   data=bytes.fromhex(data), is_extended_id=False))

    def receive_can_frame(self, timeout=1.0):
        msg = self.bus.recv(float(timeout))
        assert msg is not None, "no CAN frame"
        return f"[msg.arbitration_id:X]#{msg.data.hex().upper()}"
```

```
*** Settings ***
Library    CanLibrary.py    can0
*** Test Cases ***
Product answers the status request
    Send Can Frame    123    01
    ${frame}=    Receive Can Frame
    Should Start With    ${frame}    124#
```

testado: Robot 7.5 + python-can 4.6.1 (barramento virtual)

Nossa suíte de aceitação na placa

01 Sobe na versão de fábrica login do admin ok

02 Cria um usuário que loga user 1001 · login ok

03 Bloqueado não loga login failed (-13)

04 Apaga um usuário login failed (-2)

05 Recusa o usuário 51 add failed (-28)

06 Não assinada é recusada segue na v1.0.0

07 Sem confirmar, volta sozinho reboot → v1.0.0

08 Update mantém os usuários v2.0.0 · 50 usuários

```
$ UPDATE_PORT=<UART4> SHELL_PORT=<USB> \
  tests/robot/run.sh --exclude speed
```

```
01 Boots the factory version      PASS
02 Creates a user who can log in  PASS
03 A blocked user cannot log in   PASS
04 Deletes a user                 PASS
05 The store refuses user 51      PASS
06 An unsigned image is refused   PASS
07 An update that is not confirmed PASS
08 A confirmed update keeps the users PASS
```

8 tests, 8 passed, 0 failed
Report: tests/robot/results/report.html

azul shell USB · roxo shell + atualizador pela UART4

placa real · shell USB + UART4 · ~8 min

08 • Rodando a demo

O hardware desta demo



WeAct MiniSTM32H743 · imagem: WeAct Studio

STM32H743VIT6

Cortex-M7 até 480 MHz · 2 MB de flash · 1 MB de RAM
placa: [boards/witte/weact_stm32h743](#)

NA PLACA

USB-C	PA11 · PA12	console e shell (CDC ACM)
Display 0.96" ST7735R	SPI4 · PE11 PE12 PE13 PE14	tela LVGL 160×80
W25Q64 QSPI · 8 MB	QUADSPI · PB2 PB6 ...	slot1 do MCUboot
W25Q64 SPI · 8 MB	SPI1 · PB3 PB4 PD7	usuários (LittleFS)
LED	PE3	aceso em recovery

LIGADO POR FORA

Adaptador USB-serial	UART4 · PA0 PA1 · 921600	update e testes na placa
ST-Link / J-Link	SWD	gravação e debug

LINKEDIN

github.com/JorgeGzm/ForumIoTEmbarcados2026



Obrigado!

Jorge Guzman · GZM Embedded Systems

LinkedIn [linkedin.com/in/eng-jorge-guzman](https://www.linkedin.com/in/eng-jorge-guzman)

Web gzm-emb.com

GitHub github.com/JorgeGzm/ForumIoTEmbarcados2026