ESP-BIST

Simplificando o desenvolvimento de aplicações seguras

Lucas Tamborrino

05/08/2025

Agenda

- Introdução
- Normas de Segurança Internacionais
- Como ocorre a certificação?
- Auditoria de Software
- Testes Funcionais
- ESP-BIST
- Próximos Passos
- Q&A

Safety vs Security

- Safety (Segurança funcional): Detectar falhas não intencionais do sistema (ex: bug de software, defeito de hardware). Foco em evitar danos físicos a pessoas/meio ambiente.
- Security (Segurança cibernética): Proteger o sistema contra ataques maliciosos (ex: invasões, malware). Foco em proteger dados/integridade do sistema.

Segurança Funcional

Garantir que, quando falhas ocorrerem (e elas vão!), o sistema reaja de maneira previsível e segura.

1. Gestão de Falhas (não apenas prevenção): Assume que falhas vão acontecer (em hardware/software) e foca em:
   - Detecção (ex.: testes de RAM com March C)
   - Contenção (ex.: desligar um motor com defeito)
   - Mitigação (ex.: redundância em sistemas críticos)
2. Comportamento Determinístico em Caso de Falha

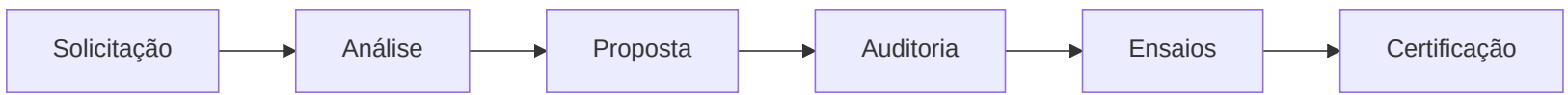
   Sistemas seguros devem falhar de modo seguro (fail-safe ou fail-silent).
3. Normas definem o "quão seguro" um sistema deve ser.

Normas de Segurança Internacionais

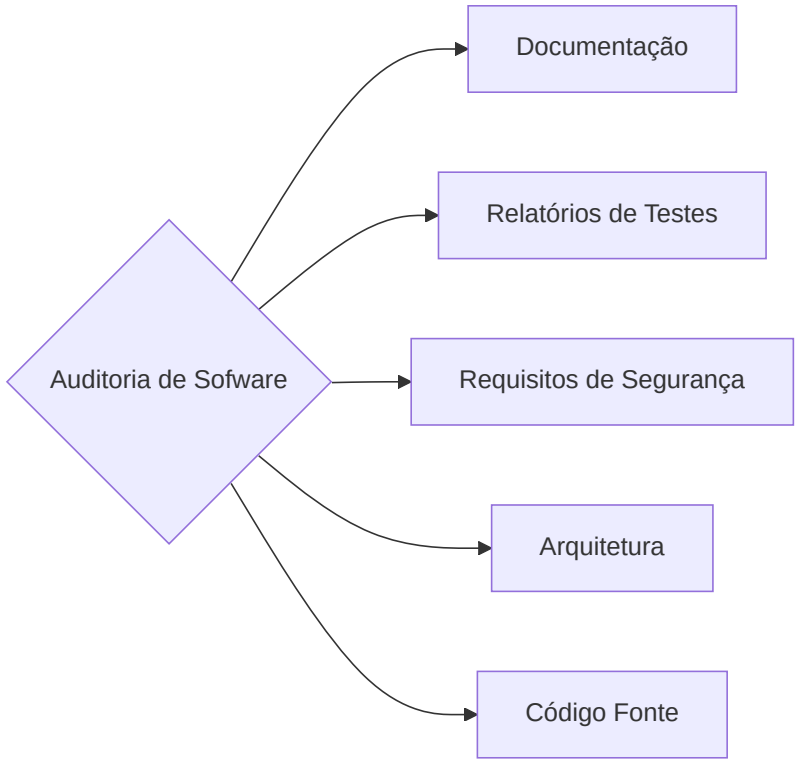
Norma	Âmbito	Níveis
IEC 61508	Sistemas industriais	SIL 1-4
ISO 26262	Automotivo	ASIL A-D
ISO 13849	Máquinas industriais	PL a-e
IEC 62304	Dispositivos médicos	Classes A-C
DO-178C	Aeronáutica	Níveis A-E
IEC 60730	**Eletrodomésticos**	Classes A, **B**, C

- Classe A: Não crítico (falha não perigosa)
- **Classe B: Risco limitado (ex.: dano material)**
- Classe C: Risco elevado (ex.: incêndio, choque elétrico)

Como ocorre a certificação?



Auditoria de Software



Ciclo de vida do Software

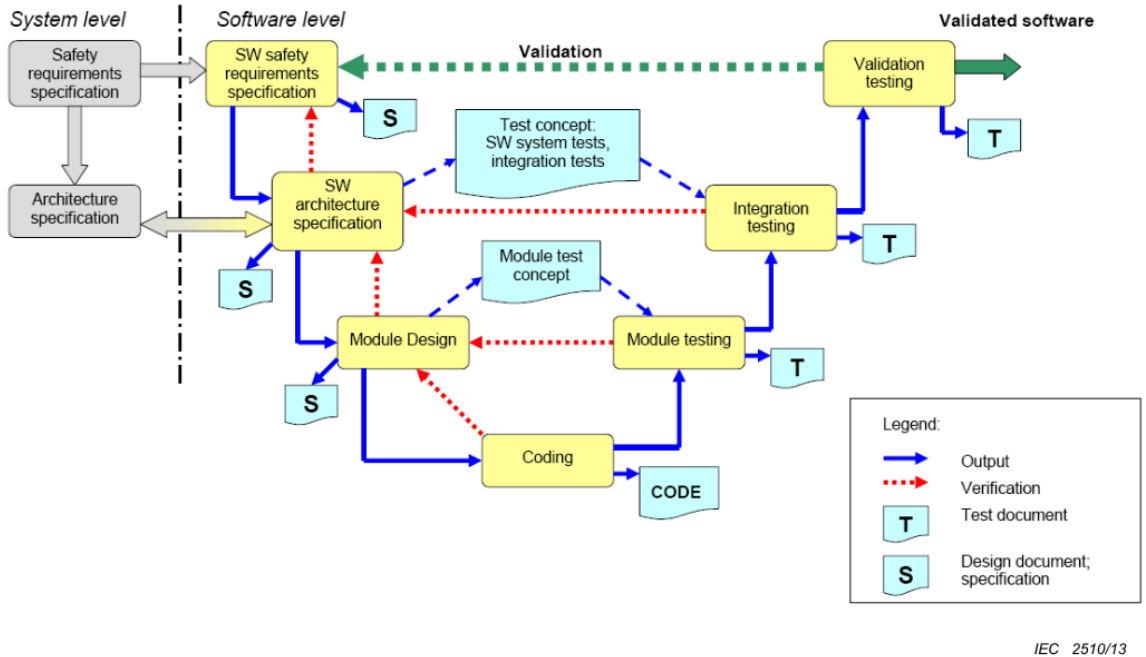


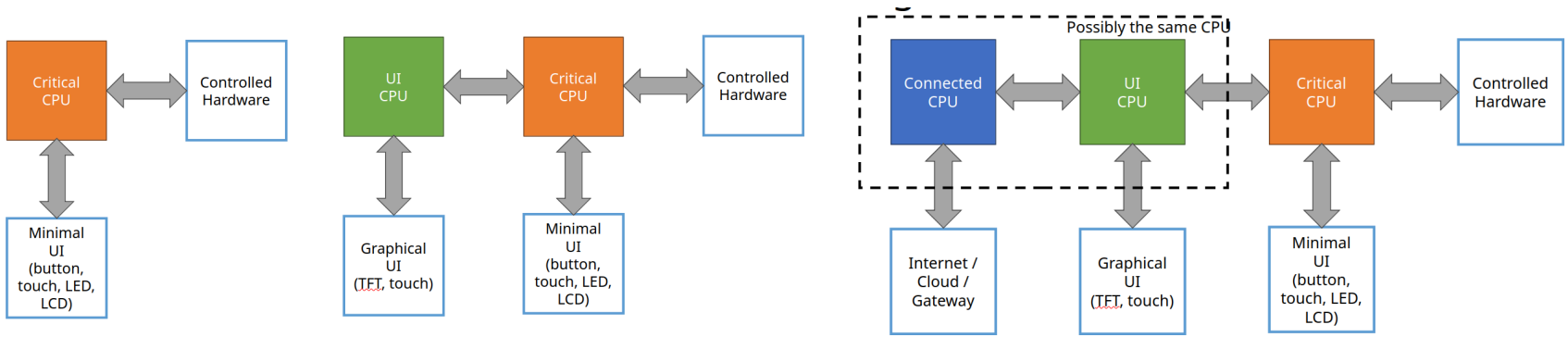
Figure H.1 – V-Model for the software life cycle

Testes

ID	Componente	Falha/Erro	Teste
1.1	CPU - Registers	Stuck at	Functional test
1.3	CPU - Program Counter	Stuck at	Functional test & time-slot monitoring
2.	Interrupt handling and execution	No interrupt or too frequent interrupt	Functional test
3.	Clock	Wrong frequency	Frequency monitoring with independent fixed frequency
4.1	Invariable Memory	All single bit faults	Checksum
4.2	Variable Memory	DC fault	March A & March X
7.1	Digital I/O	Fault conditions specified in H.27	Plausibility check
7.2.1	Analog I/O	Fault conditions specified in H.27	Plausibility check

Isolamento de Software Crítico

- As normas determinam que o software de segurança crítica deve ser executado em ambiente isolado, sem coexistência com software de finalidade não crítica.
- A execução de software crítico não deve ocorrer em conjunto com protocolos de rede, acesso à internet ou quaisquer serviços que introduzam comportamento não determinístico ao sistema.



ESP-BIST

Espressif’s Built-In Self-Test

Objetivos

- Simplificar e acelerar o desenvolvimento de software para aplicações seguras utilizando os SoCs Espressif, em conformidade com as normas e requisitos de certificação.
- Proporcionar um framework para desenvolvimento de aplicações seguras.
- Fornecer uma biblioteca de testes de hardware e software.

ESP-BIST

- Standalone
  - Cmake framework
  - Bare Metal
  - IDF HAL
  - RISCV SOCs (ESP32-C3, ESP32-C6, ESP32-H2, ESP32-P4, etc.)
- Módulo Zephyr & Componente IDF
  - Cmake library
  - Foco em LP-CPU
  - AMP (Assimetric Multi-Processing)
  - HP Core (RTOS, Network, UI, etc) - LP Core (Bare Metal, Critical App + BIST)
  - RISCV LP Core SOCs (ESP32-S3, ESP32-C6, ESP32-P4, etc)

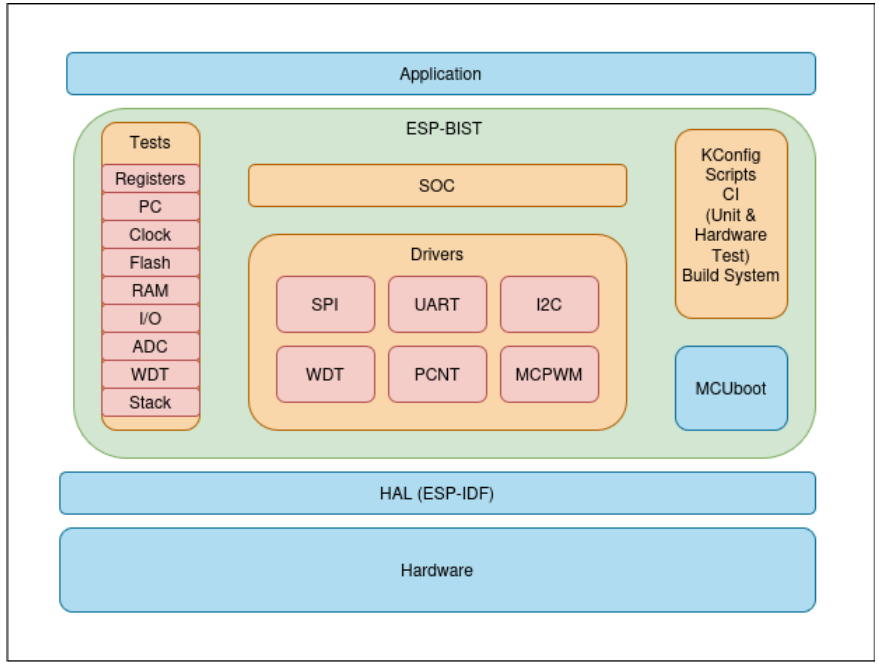
Adicionais

- Stack High Watermark (stack overflow)
- WDT Tests
- WDT para Crystal de referência (32,768 kHz)

ESP-BIST

Standalone

- Framework para desenvolvimento de aplicações seguras
- Bare Metal
- Offline
- CMake
- Kconfig
- MCUboot
- Devcontainer
  - ESP-IDF - v5.1.4 (HAL)
  - Qemu - v8.2.0
  - esptool - v4.7.0
  - MCUboot - v2.2.0
  - pytest - v8.4.1



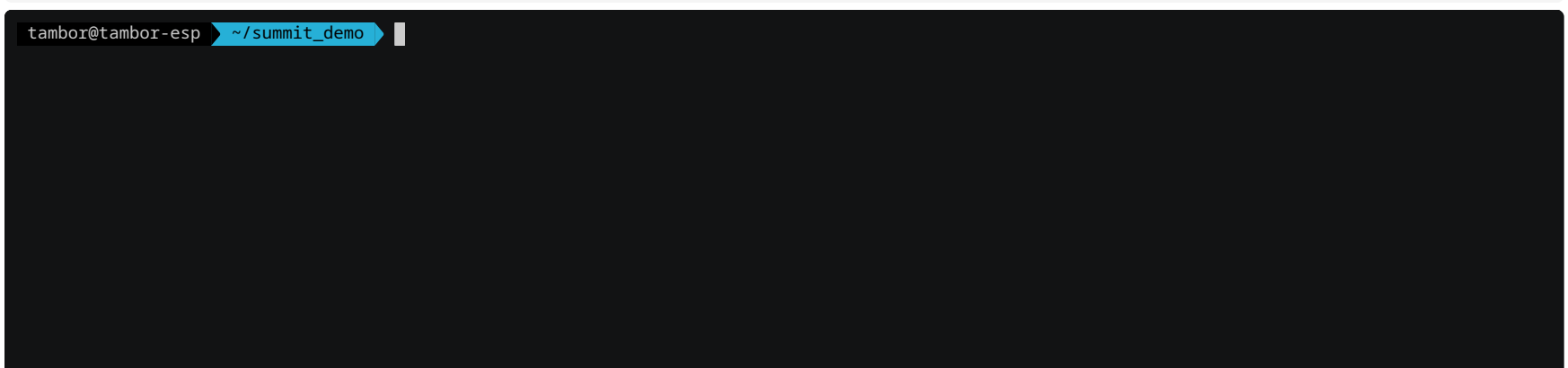
Workflow - Standalone

Download

```
git clone https://github.com/espressif/esp-bist.git
```

Dev Container

```
cd esp-bist
devcontainer build --workspace-folder .
devcontainer up --workspace-folder .
devcontainer exec --workspace-folder . bash
```



Workflow - Standalone

Build Bootloader

```
cd $MCUBOOT_PATH/boot/espressif
cmake -DCMAKE_TOOLCHAIN_FILE=tools/toolchain-esp32c3.cmake -DMCUBOOT_TARGET=esp32c3 -DESP_HAL_PATH=$IDF_PATH -B build -
GNinja
ninja -C build
```

Workflow - Standalone

Build Project

```
cd project/directory
cmake -DSOC_TARGET=esp32c3 -B build -GNinja
ninja -C build
```

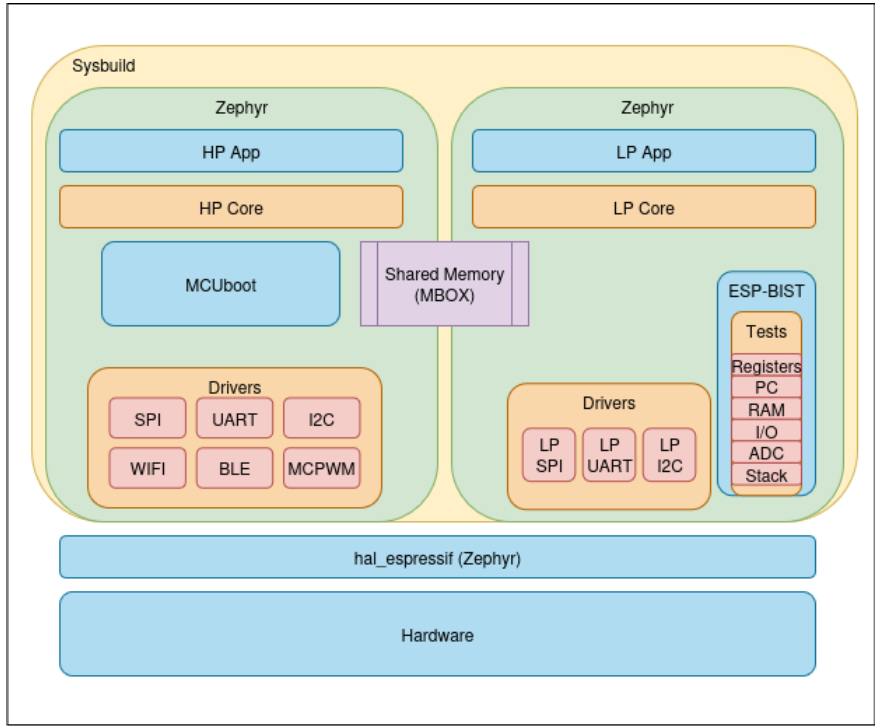
Flash && Monitor

```
ninja -C build flash
ninja -C build monitor
```

ESP-BIST

Zephyr Module

- Cmake library
- Kconfig
- AMP (Asymmetric Multi-Processing)
- Zephyr Build System (Sysbuild)
- Device tree
- IPC (Mailbox)
- Sample Project



ESP-BIST - Zephyr Module

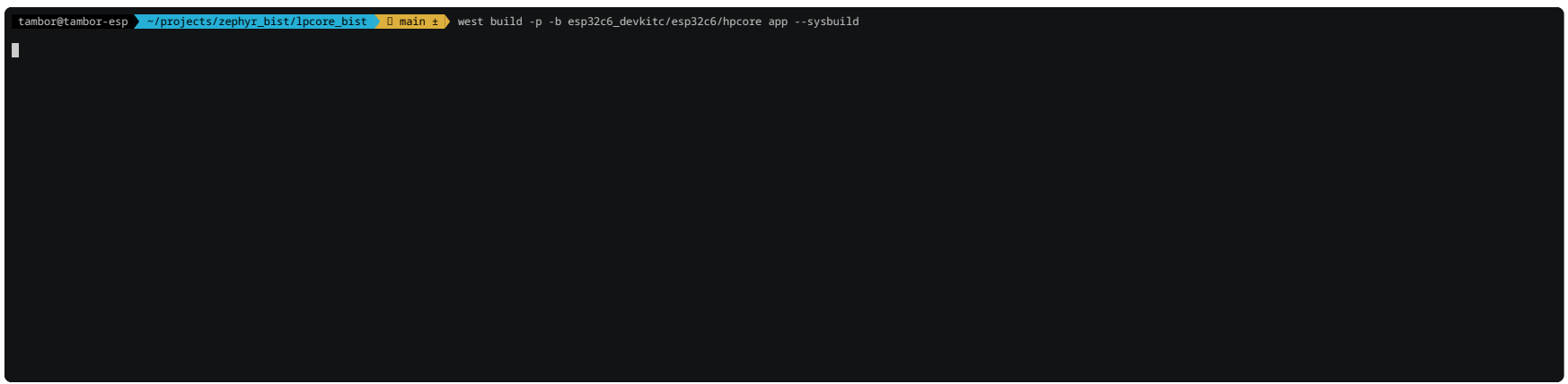
```
zephyr/
├── boards/
│   ├── esp32c6_devkitc_hpcore.conf
│   └── esp32c6_devkitc_hpcore.overlay
├── remote/
│   ├── boards/
│   │   └── esp32c6_devkitc_lpcore.overlay
│   ├── src/
│   │   └── main.c
│   ├── CMakeLists.txt
│   ├── Kconfig
│   └── prj.conf
├── src/
│   └── main.c
├── CMakeLists.txt
├── Kconfig
├── Kconfig.sysbuild
├── prj.conf
├── README.md
└── sysbuild.cmake
```

```c
// main.c
static void runtime_tests(void)
{
        bist_esp_err_t test_err = BIST_ESP_OK;

        // Disable interupts
        ulp_lp_core_intr_disable();

        test_err = bist_cpu_regs_test();
        if (test_err == BIST_ESP_CPU_TEST_ERR) {
                printf("CPU register test failed\n");
                fail_safe_exit();
        }

        test_err = bist_cpu_csr_regs_test();
        if (test_err == BIST_ESP_CPU_CSR_TEST_ERR) {
                printf("CPU CSR register test failed\n");
                fail_safe_exit();
        }

        // Enable interrupts
        ulp_lp_core_intr_enable();

        printf("All RUNTIME tests passed!\n");
}
```

Workflow - Zephyr Module

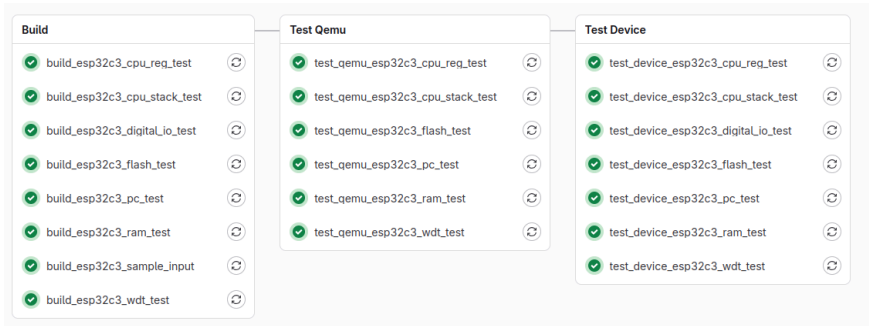
Build Project

```
# Inside Zephyr workspace
west build -p -b esp32c6_devkitc/esp32c6/hpcore <path/to/esp-bist/samples/zephyr/> -D ZEPHYR_EXTRA_MODULES=<path/to/esp-bist/> --sysbuild
```



Who watches the Watchmen?

- CI
- Test Framework
  - Pytest
  - Unity
  - Qemu
  - GDB Scripts
- Fault Injection
- Hardware Tests



Resumo

- ESP-BIST é um framework e biblioteca para desenvolvimento de aplicações seguras com SoCs Espressif.
- Testes de hardware e software para garantir a segurança funcional.
- Ferramentas de desenvolvimento.
- Standalone Bare Metal
- Integração com Zephyr
- Integração com ESP-IDF

Próximos Passos

- Certificação do ESP-BIST
- Integração com ESP-AMP
- Time triggered scheduler
- Suporte a mais SoCs:
  - ESP32-S3
  - ESP32-C2
  - ESP32-H2
  - ESP32-P4
  - ESP32-C5
  - ESP32-H4
  - ESP32-C61
- Artigos e tutoriais

Q&A

Perguntas, sugestões, feedbacks?