

# Relatório de Desenvolvimento do HAL Bluetooth com Zephyr RTOS

Angelo Luigi Bocchi Lovatto

26 de novembro de 2025

## Resumo

Este relatório apresenta um resumo das etapas de estudo e desenvolvimento realizadas ao longo do projeto de criação de um (HAL) com suporte a Bluetooth utilizando o Zephyr RTOS em uma placa baseada no microcontrolador SAMD21, em conjunto com um módulo BLE externo (BTLC1000). São descritos o contexto do projeto, a preparação do ambiente, os principais testes realizados, os erros encontrados, as soluções aplicadas e os aprendizados adquiridos ao longo do processo.

## 1 Introdução

O projeto de HAL com Bluetooth utilizando o Zephyr RTOS surgiu no contexto da disciplina de Sistemas Embarcados, com o objetivo de ir além dos exemplos básicos de GPIO e temporização, explorando um sistema operacional de tempo real moderno, a integração com uma placa de desenvolvimento SAMD21 e a comunicação sem fio via Bluetooth.

Ao longo dos trabalhos, foram realizados diversos experimentos envolvendo:

- instalação e configuração do ambiente Zephyr e da ferramenta `west`;
- primeiros testes com exemplos básicos (`hello_world`, `blinky`);
- uso da placa SAMD21 Xplained Pro como alvo real;
- tentativas de integração do módulo Bluetooth (BTLC1000) utilizando a *Bluetooth stack* do Zephyr;
- organização de um HAL específico para o projeto, integrando camadas de hardware, drivers e aplicação.

Este relatório consolida essa trajetória, com foco tanto na parte técnica quanto nos erros e obstáculos que ajudaram a consolidar o aprendizado.

## 2 Contexto do Projeto

O projeto está inserido em um repositório colaborativo, voltado para o desenvolvimento de aplicações com Zephyr em uma placa SAMD21, incluindo uma camada HAL que facilite a reutilização de código em diferentes exercícios e experimentos.

O objetivo específico da parte **HAL–Bluetooth–Zephyr** é:

- disponibilizar uma camada de abstração para funcionalidades básicas da placa (GPIO, temporizadores, UART, etc.);
- integrar essa camada com a *stack* Bluetooth do Zephyr, permitindo comunicação BLE;
- documentar a trajetória de aprendizagem, evidenciando erros comuns de configuração e compilação e como foram resolvidos;
- deixar uma base de código e de documentação que possa ser usada como referência em trabalhos futuros.

## 3 Preparação do Ambiente e Primeiros Passos

### 3.1 Instalação do Zephyr e `west`

Uma parte significativa do trabalho inicial foi dedicada a configurar corretamente o ambiente de desenvolvimento:

- instalação do Python em versão compatível com o Zephyr;
- instalação do `west`, ferramenta de gerenciamento de projetos do Zephyr;
- execução do `west init` e `west update` para obter o código-fonte do Zephyr;
- configuração da *toolchain* (compilador `arm-none-eabi-gcc`, CMake, Ninja, etc.);
- ajustes de variáveis de ambiente (`PATH`) no Windows para que todas as ferramentas fossem encontradas corretamente.

#### Erros comuns nesta etapa

Durante essa fase, surgiram problemas típicos de ambiente:

- mensagens de erro do CMake por não encontrar o SDK ou o compilador;
- diretórios incorretos ao executar `west build` (por exemplo, tentar compilar fora da pasta `zephyrproject/zephyr`);

- versões conflitantes de Python ou pacotes não instalados (`pip` faltando algum módulo).

A correção desses erros envolveu:

- revisar cuidadosamente a documentação oficial do Zephyr;
- padronizar o uso de um único terminal (PowerShell) e de um único diretório raiz (`C:\Users\angel\zephyrproject`);
- reinstalar ou ajustar paths sempre que necessário.

### 3.2 Primeiros testes: *hello\_world* e *blinky*

Antes de partir para Bluetooth, foi fundamental validar o ambiente compilando e executando exemplos simples:

- `samples/hello_world`;
- `samples/basic/blinky`.

Esses testes foram construídos tanto para a simulação (quando aplicável) quanto para a placa SAMD21 Xplained Pro. Quando o LED da placa finalmente piscou com o exemplo *blinky*, ficou confirmada a cadeia completa: *build*, *flash* e execução no hardware real.

## 4 Migração para a Placa SAMD21 e Organização do HAL

### 4.1 Uso da placa SAMD21 Xplained Pro

Após validar os exemplos básicos, o foco passou a ser o alvo real do projeto: a placa SAMD21 Xplained Pro. Isso exigiu atenção especial para:

- selecionar corretamente o *board* no comando `west build -b samd21_xpro`;
- entender o mapeamento de pinos da placa (LEDs, botões, interfaces de comunicação);
- configurar o método de gravação (por exemplo, `west flash` aproveitando o debugger on-board).

## 4.2 Estruturação do HAL

Com o ambiente funcional, iniciou-se a organização do **HAL** no repositório:

- criação de diretórios específicos para o projeto HAL;
- padronização de nomes de pastas e arquivos, facilitando a navegação e a compreensão pelo professor e pelos colegas;
- definição de uma camada de abstração para funções de baixo nível, por exemplo:
  - inicialização de GPIO e controle de LEDs;
  - leitura de botões físicos;
  - inicialização de UART para *debug*;
  - , futuramente, interface com o módulo Bluetooth.

Também foi feita a integração com o repositório GitHub da disciplina, utilizando o GitHub Desktop para organizar *commits* por semana, o que ajudou a documentar a evolução do trabalho.

## 5 Integração com Bluetooth e Zephyr

### 5.1 Escolha dos exemplos de Bluetooth

Para introduzir o Bluetooth, foram escolhidos exemplos já prontos do próprio Zephyr, como:

- `samples/bluetooth/peripheral_hr` (exemplo de *heart rate*).

A ideia era:

1. entender como o Zephyr organiza a *Bluetooth stack* (camadas de *host*, *controller*, serviços BLE, etc.);
2. adaptar o exemplo para rodar na placa SAMD21 com o módulo BTLC1000;
3. por fim, integrar as funções relevantes ao HAL do projeto.

### 5.2 Principais dificuldades ao habilitar Bluetooth

Ao tentar compilar exemplos de Bluetooth para o `samd21_xpro`, diversos problemas apareceram, por exemplo:

- erros de configuração, como falta de definições `CONFIG_BT` ou ajustes de `Kconfig`;

- limitações do próprio *board* no Zephyr, uma vez que o SAMD21 não possui Bluetooth integrado – é necessário um módulo externo (BTLC1000);
- necessidade de ajustar o *device tree* (`.dts`) para descrever corretamente a conexão entre o SAMD21 e o módulo Bluetooth (SPI, UART ou outra interface, *pins*, interrupções, etc.);
- mensagens de erro de *build* indicando ausência de suporte Bluetooth padrão para aquela placa.

Nem todos esses pontos foram totalmente resolvidos, mas a investigação por si só trouxe aprendizados importantes:

- compreensão de como Zephyr separa *board*, *SoC*, *drivers* e *device tree*;
- entendimento de que nem todo exemplo Bluetooth é imediatamente portável para qualquer placa;
- necessidade de leitura de documentação não apenas do Zephyr, mas também do fabricante (Microchip) para o módulo BTLC1000.

### 5.3 Relação com o HAL

Mesmo sem concluir toda a pilha Bluetooth no hardware real, foi possível:

- planejar interfaces no HAL pensando em funções genéricas de envio/recebimento de dados sem fio;
- organizar o código de forma que, uma vez estabilizada a integração com o módulo BLE, o restante da aplicação não precise mudar;
- estruturar o projeto para suportar futuras extensões (por exemplo, adicionar sensores e enviar leituras via BLE).

## 6 Principais Erros e Como Foram Corrigidos

Nesta seção estão condensados alguns dos tipos de erros que apareceram recorrentemente ao longo do projeto.

### 6.1 Erros de caminho e diretório

Um erro clássico foi executar `west build` no diretório errado, ou apontar o `-s` (*source*) e o `-b` (*board*) incorretamente. Exemplos:

- tentar compilar um exemplo do Zephyr dentro de uma pasta que não continha o CMakeLists adequado;
- confundir o diretório raiz do projeto Zephyr (`zephyrproject/zephyr`) com o diretório onde estava o repositório específico da disciplina.

#### Solução:

- padronizar a navegação: sempre iniciar no diretório `zephyrproject`;
- usar comandos claros, como:

```
cd C:\Users\angel\zephyrproject\zephyr
py -m west build -b samd21_xpro samples/hello_world -p
```

- anotar, no próprio relatório e em arquivos README, o passo a passo correto.

## 6.2 Erros de configuração (`Kconfig` e `prj.conf`)

Outro grupo de erros estava relacionado a `Kconfig`:

- opções de Bluetooth não habilitadas;
- dependências internas do Zephyr não satisfeitas;
- conflitos entre configurações de `prj.conf` e capacidades reais da placa.

#### Solução parcial:

- estudar as opções de `menuconfig` e a documentação das *samples* Bluetooth;
- comparar arquivos `prj.conf` dos exemplos que funcionam com *boards* suportados;
- ir ajustando e recompilando, observando o que muda em termos de erros.

## 6.3 Erros de *drivers* e *device tree*

Ao tentar aproximar o exemplo de Bluetooth do hardware real (SAMD21 + BTLC1000), surgiram:

- mensagens de erro indicando que determinados *nodes* do *device tree* não existiam;
- necessidade de especificar *buses* (SPI, UART) e pinos corretos para a comunicação com o módulo BLE;
- dúvidas sobre qual *driver* do Zephyr poderia ser reutilizado ou adaptado.

### Solução parcial:

- leitura da documentação de *device tree* do Zephyr;
- comparação com o *device tree* de *boards* que já possuem Bluetooth funcional;
- planejamento futuro de criação de um *overlay* específico para o módulo BTLC1000, ainda em desenvolvimento.

## 6.4 Problemas de gravação e *debug*

Ao longo do desenvolvimento, também apareceu:

- dificuldade de *flash* em alguns momentos, por causa de drivers USB ou conflitos entre ferramentas de gravação;
- necessidade de escolher uma *toolchain* consistente entre os diversos testes (Renode, placa real, etc.).

### Solução:

- focar em um único fluxo: PowerShell + `west flash`, usando sempre o mesmo conjunto de ferramentas;
- evitar misturar demais ferramentas paralelas (por exemplo, evitar usar simultaneamente o ambiente da Microchip e o fluxo completo do Zephyr para o mesmo projeto).

## 7 Aprendizados Principais

Ao final dessa etapa do projeto, mesmo com algumas partes ainda em construção (como o suporte completo ao módulo BTLC1000), os principais aprendizados foram:

- **Domínio básico do Zephyr RTOS:** compreender a estrutura de um projeto Zephyr, uso do `west`, conceitos de `Kconfig`, *device tree* e `prj.conf`.
- **Importância da organização do repositório:** separar exemplos, código de produção e HAL em pastas coerentes, facilitando a correção e o reuso.
- **Depuração de erros de *build*:** transformar mensagens de erro em pistas de investigação, em vez de apenas repetir comandos sem entender o problema.
- **Integração hardware–software:** perceber que, para habilitar Bluetooth, não basta “ligar uma opção”; é necessário que hardware, *device tree*, drivers e configuração conversem entre si.

- **Uso de ferramentas de controle de versão:** criação de *commits* com mensagens significativas, estruturando o progresso em semanas (“*HAL Progresso semana X*”).
- **Postura de projeto:** em vez de apenas rodar exemplos prontos, pensar em camadas (HAL, aplicação, drivers) e em como a solução pode ser estendida no futuro.

## 8 Conclusões e Próximos Passos

O trabalho com o HAL–Bluetooth–Zephyr ainda está em evolução, mas já proporcionou uma base sólida em:

- desenvolvimento embarcado com RTOS;
- integração de hardware real (SAMD21) com um ecossistema moderno como o Zephyr;
- estudo preliminar da *stack* Bluetooth e dos requisitos de configuração para habilitá-la.

Como próximos passos, destacam-se:

- finalizar a descrição do módulo BTLC1000 no *device tree*, com todas as conexões corretas;
- validar, na prática, um exemplo BLE simples (por exemplo, serviço de *heart rate* ou um serviço customizado para envio de dados);
- integrar as funções Bluetooth de maneira limpa na camada HAL, para que a aplicação possa usar essas funções de forma transparente;

Em resumo, o conjunto de erros, correções e experimentos descritos aqui representa uma trajetória realista de aprendizado em sistemas embarcados modernos, indo desde a configuração básica do ambiente até o planejamento de uma solução completa com suporte a Bluetooth, estruturada em camadas de abstração de hardware.