

Edição online

Seminário de Sistemas Embarcados e IoT **2020**



Felipe Neves

Palestra

Crie sua aplicação Bluetooth Low Energy com Zephyr e VSCode

Apoio



Sobre mim:

- Atuo com sistemas embarcados desde 2005, entusiasta da filosofia maker e projetos open source;
- Mestre em Engenharia Elétrica pela USP, Tecnólogo em Eletrônica Automotiva pela FATEC São Paulo, Técnico em Mecatrônica pela ETEC;
- Interessado em tópicos envolvendo: sistemas de tempo real, robótica, controle digital e IoT;
- Atualmente trabalho na Espressif Systems como Engenheiro de Software Sênior, cuidando da pilha de software embarcado dos módulos ESPs.



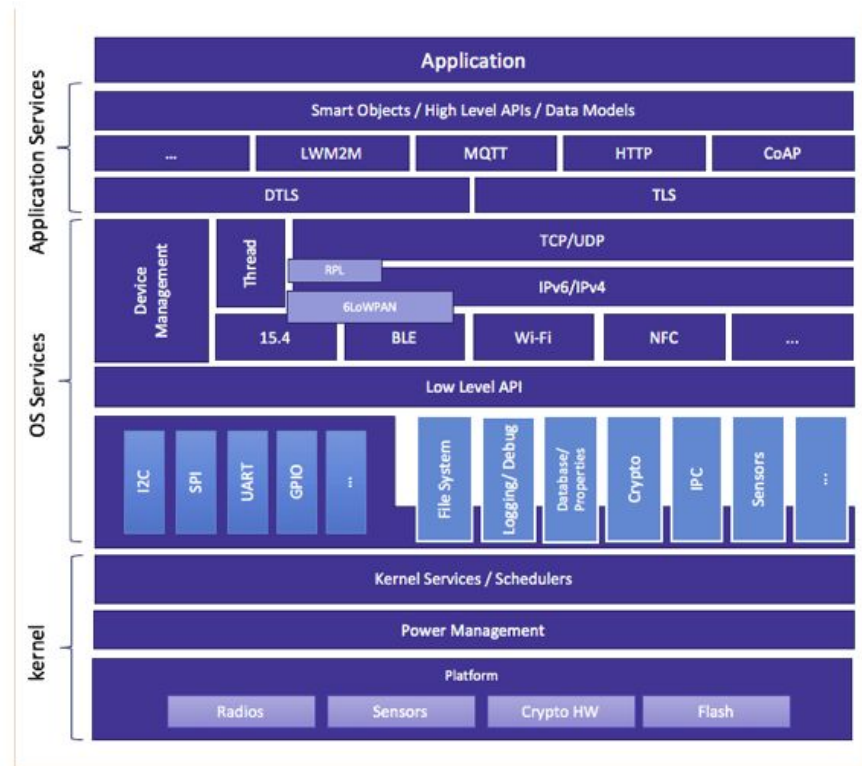
AGENDA

- Revisitando o Zephyr RTOS, as mudanças dos novos LTS;
- Revisitando o protocolo Bluetooth Low Energy - BLE:
 - Central Role e Peripheral Role;
 - Advertising e Scan Cycle;
 - Troca de dados em conexão ativa.
- A Stack BLE do Zephyr:
 - Controller layer e Host layer;
 - Configurando a stack;
 - Descrevendo um GATT service para simular uma UART;
 - Iniciando a stack BLE, registrando o serviço GATT e fazendo sua descoberta no master.
- Colocando tudo junto no VSCode:
 - Detalhes após a instalação do Zephyr;
 - Configurando o Debug com JLink usando o Cortex Debug Plugin;
 - Configurando o Zephyr e realizando o build do app exemplo;
 - Iniciando uma sessão de debug, breakpoints, watch e steps.
- Wrap-up
 - Onde conseguir os fontes dessa demonstração;
 - Onde acessar os guias de instalação do Zephyr;
 - Dúvidas.



Revisitando Zephyr RTOS:

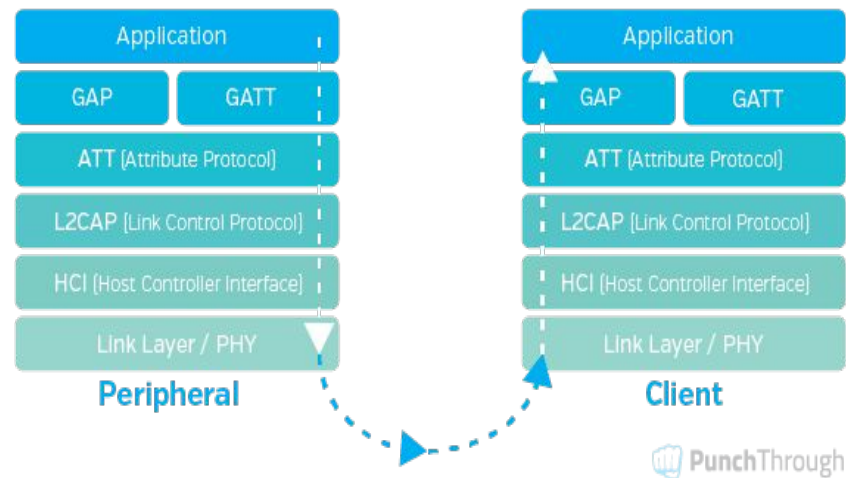
- RTOS escalável, vendor neutral;
- Device model uniforme;
- Suporta várias arquiteturas;
- Atualmente mantido por um comite, o TSC;
- Long Term Support a partir da versão 1.14;
- Stack BLE 5.1 certificada pelo SIG;





Revisitando o protocolo BLE:

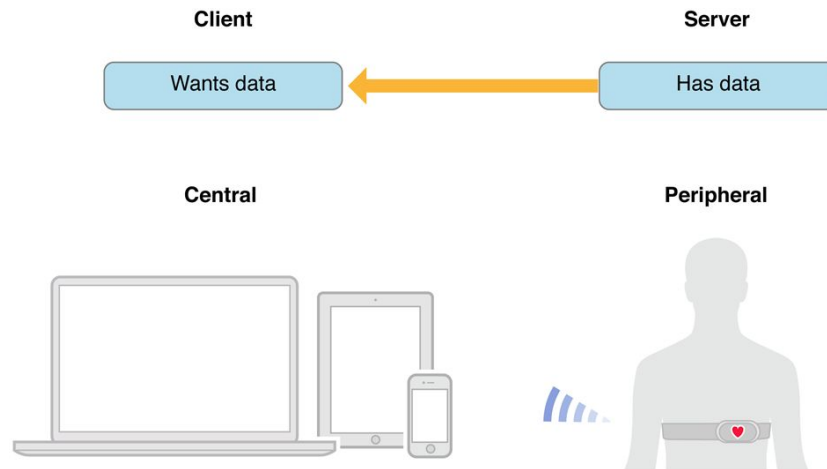
- Variação do Bluetooth clássico, voltado para prover conexão sem fios aliado a baixo consumo de energia;
- Na versão 5.0 em diante a PHY é capaz de prover largura de banda de até 2Mbps;
- Possui padronização consistente mantido pelo grupo Bluetooth SIG, garantindo interoperabilidade.





Central Role E Peripheral Role:

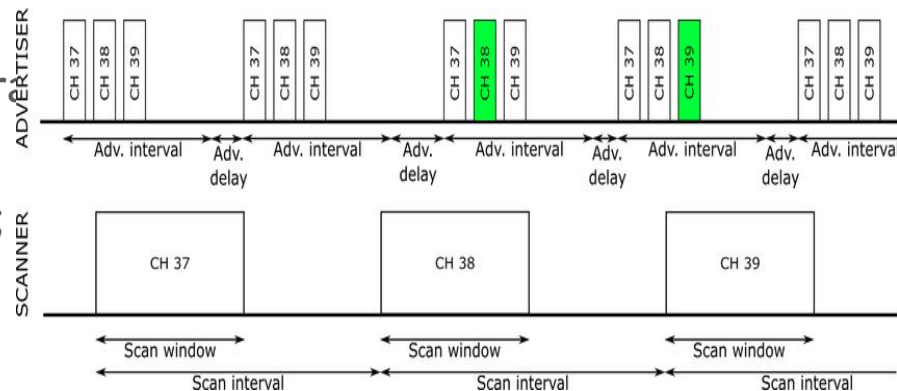
- **Central:** Efetua conexão, procura por Peripherals, pode conectar em mais de um dispositivo;
- **Peripheral:** Geralmente a fonte de dados (sensores), publica sua presença para o Central, apenas 1 conexão por Central.





O Scanner:

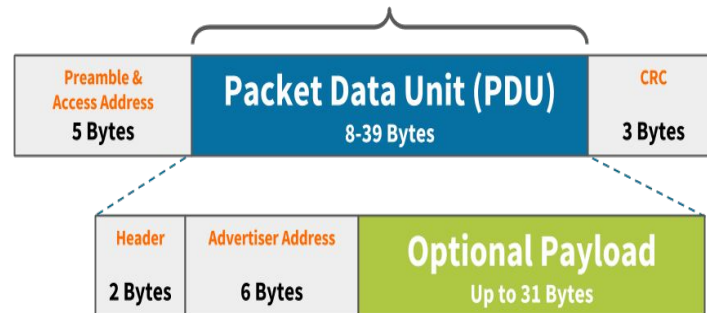
- Realizado pelo Central, procura por dispositivos conectáveis;
- Passado o intervalo de scan, o Central poderá conectar nos dispositivos descobertos durante esse intervalo;
- Esse ciclo pode ser utilizado para troca de dados sem conexão.





O Advertisement:

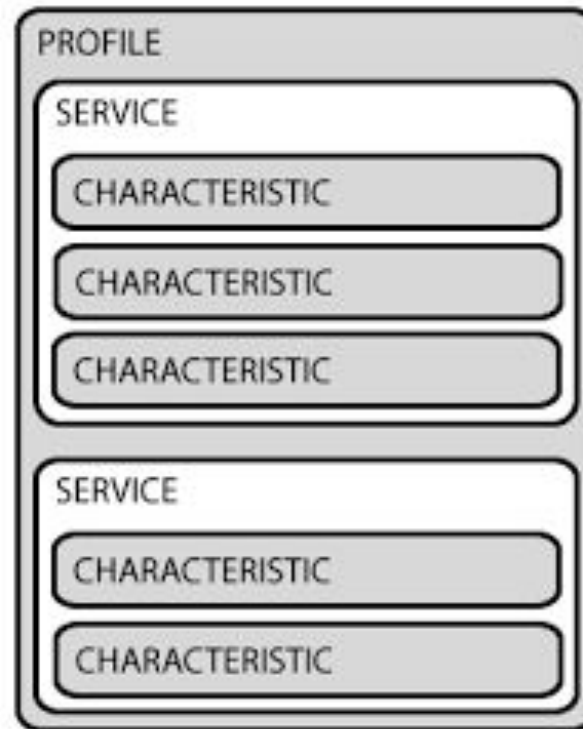
- Realizado pelo Peripheral, que envia periodicamente um pacote de dados para que uma Central próxima saiba de sua presença;
- O pacote de Advertisement pode conter dados específicos do Peripheral, usado em Beacons para troca de dados sem conexão





Trocando dados em conexão:

- Uma vez conectado, o Central realiza a descoberta dos serviços do Peripheral;
- Os serviços contêm características e descritores que determinam a direção, conteúdo e permissões de acesso aos dados;
- As propriedades mais comuns são: WRITE_WITHOUT_RESPONSE, WRITE, NOTIFY e READ;
- As permissões mais comuns são PERM_WRITE e PERM_READ.

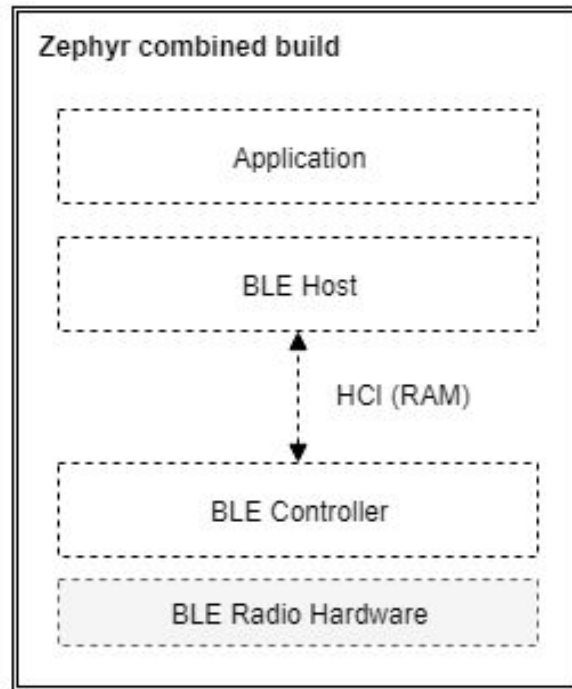




A Stack BLE do Zephyr

- Stack BLE 5.1 certificada pelo Bluetooth SIG.
- Suporte a Controller+Host, Host-Only ou Controller-Only, dependendo da plataforma;
- Muito configurável;
- Código aberto.

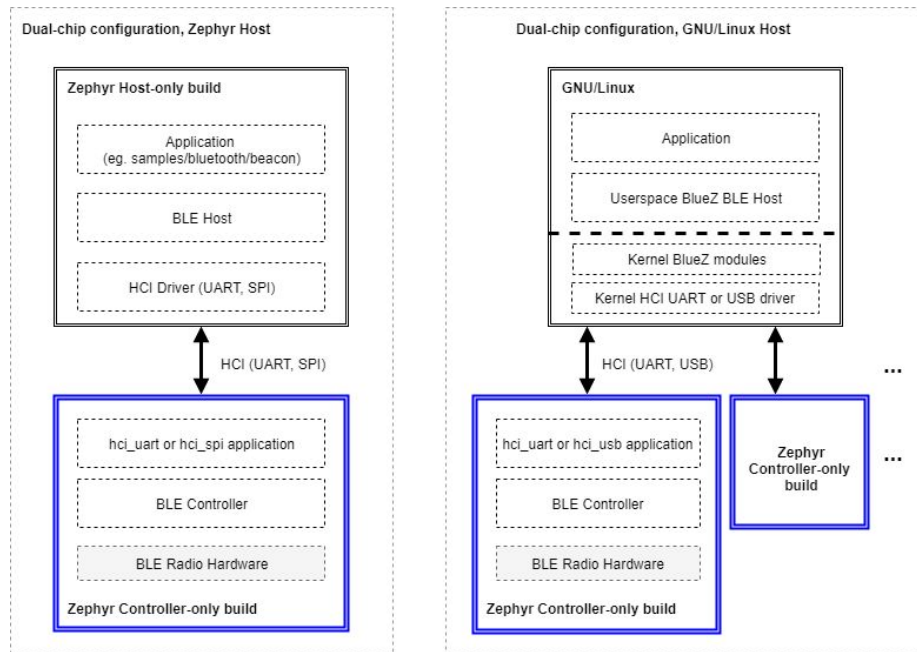
Single-chip configuration





Controller layer e Host Layer

- A stack do zephyr é flexível para suportar diferentes tipos de build.





A stack BLE do Zephyr, continuação

(Top) → Bluetooth

Zephyr Kernel Configuration

Bluetooth Stack Selection (HCI-based) --->

```
[ ] RAW HCI access
[ ] RAW HCI H:4 transport
[ ] RAW HCI Command Extension
[*] Peripheral Role support
[ ] Central Role support
[ ] Extended Advertising and Scanning support [EXPERIMENTAL]
  Observer --->
  GATT Services --->
(1) Maximum number of simultaneous connections
[ ] Controller to Host ACL flow control support
[ ] Enable fetching of remote version
```

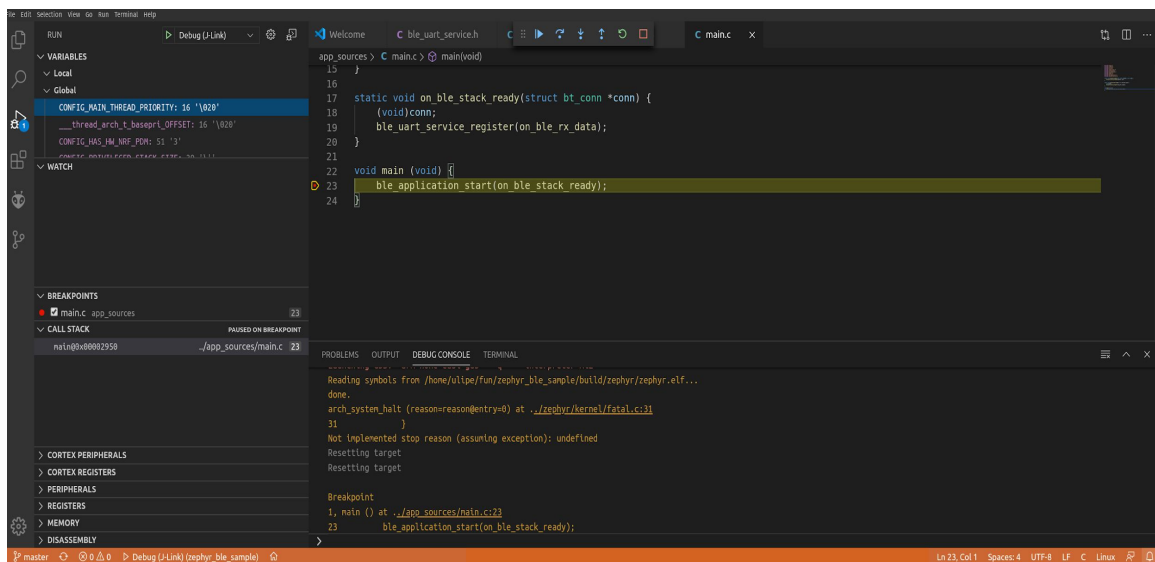
↓↓↓↓↓↓↓↓↓↓↓↓

```
[Space/Enter] Toggle/enter  [ESC] Leave menu      [S] Save
[O] Load          [?] Symbol info    [/] Jump to symbol
[F] Toggle show-help mode  [C] Toggle show-name mode  [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```



Colocando tudo junto no VSCode:

- Apesar do suporte em linha de comando, é possível usar uma IDE para desenvolver com o Zephyr;
- O VSCode oferece um IDE leve, poderoso e com plugins interessantes, incluindo sessões de debug.





Detalhes após instalar o Zephyr

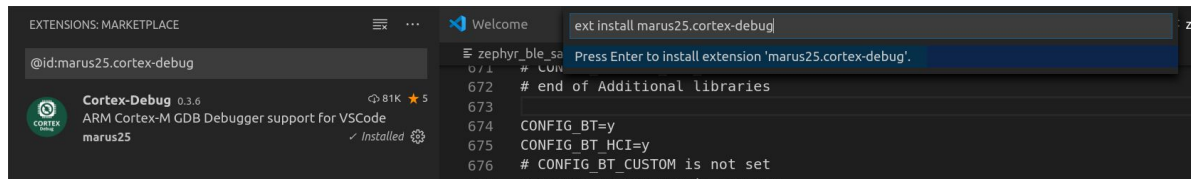
- Colocar o código de aplicação em modo out-of-tree;
- Inicialize o west no diretório de sua aplicação;
- Crie um arquivo para preparar o ambiente para o Zephyr;
- Configure o kernel e salve a configuração para uso futuro.

```
▼ ZEPHYR_BLE_SAMPLE
> .vscode
▼ .west
  ≡ config
▼ app_sources
  C ble_application.c
  C ble_application.h
  C ble_uart_service.c
  C ble_uart_service.h
  C main.c
> bootloader
> build
> modules
> tools
> zephyr
◆ .gitignore
M CMakeLists.txt
i README.md
≡ setup.sh
≡ zephyr_ble_sample_config
```



O Ambiente de Debug com JLink

- Instalando o cortex debug usando o VSCode;
- Adicione uma pasta .vscode na sua aplicação;
- Crie o Arquivo de configuração de debug launch.json;
- Aponte nesse arquivo local do executável, serviço de debug e demais opções, template ao lado.



```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "cortex-debug",
      "request": "launch",
      "servertype": "jlink",
      "cwd": "${workspaceRoot}",
      "executable": "build/zephyr/zephyr.elf",
      "name": "Debug (J-Link)",
      "device": "nRF52832_xxAB",
      "interface": "swd",
    }
  ]
}
```

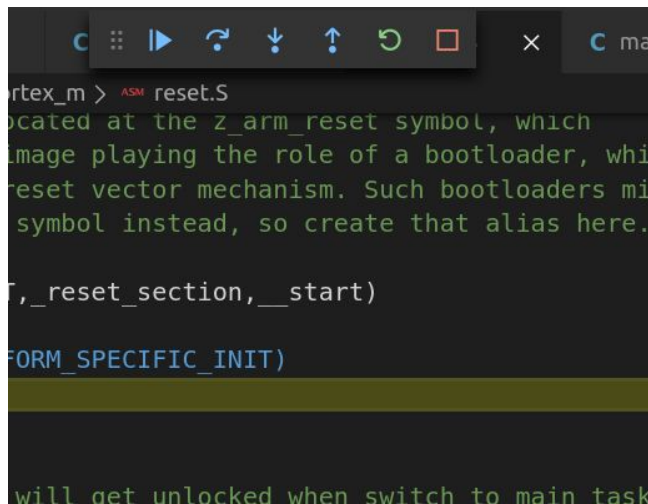


Build de aplicação, flash e debug:

```
ulipe@pop-os:~/fun/zephyr_ble_sample/build$ cmake -GNinja -DBOARD=nrf52dk_nrf52832 ..
```

```
ulipe@pop-os:~/fun/zephyr_ble_sample/build$ ninja build
```

```
ulipe@pop-os:~/fun/zephyr_ble_sample/build$ ninja menuconfig
```



```
ortex_m > ASM reset.S
located at the z_arm_reset symbol, which
image playing the role of a bootloader, whi
reset vector mechanism. Such bootloaders mi
symbol instead, so create that alias here.

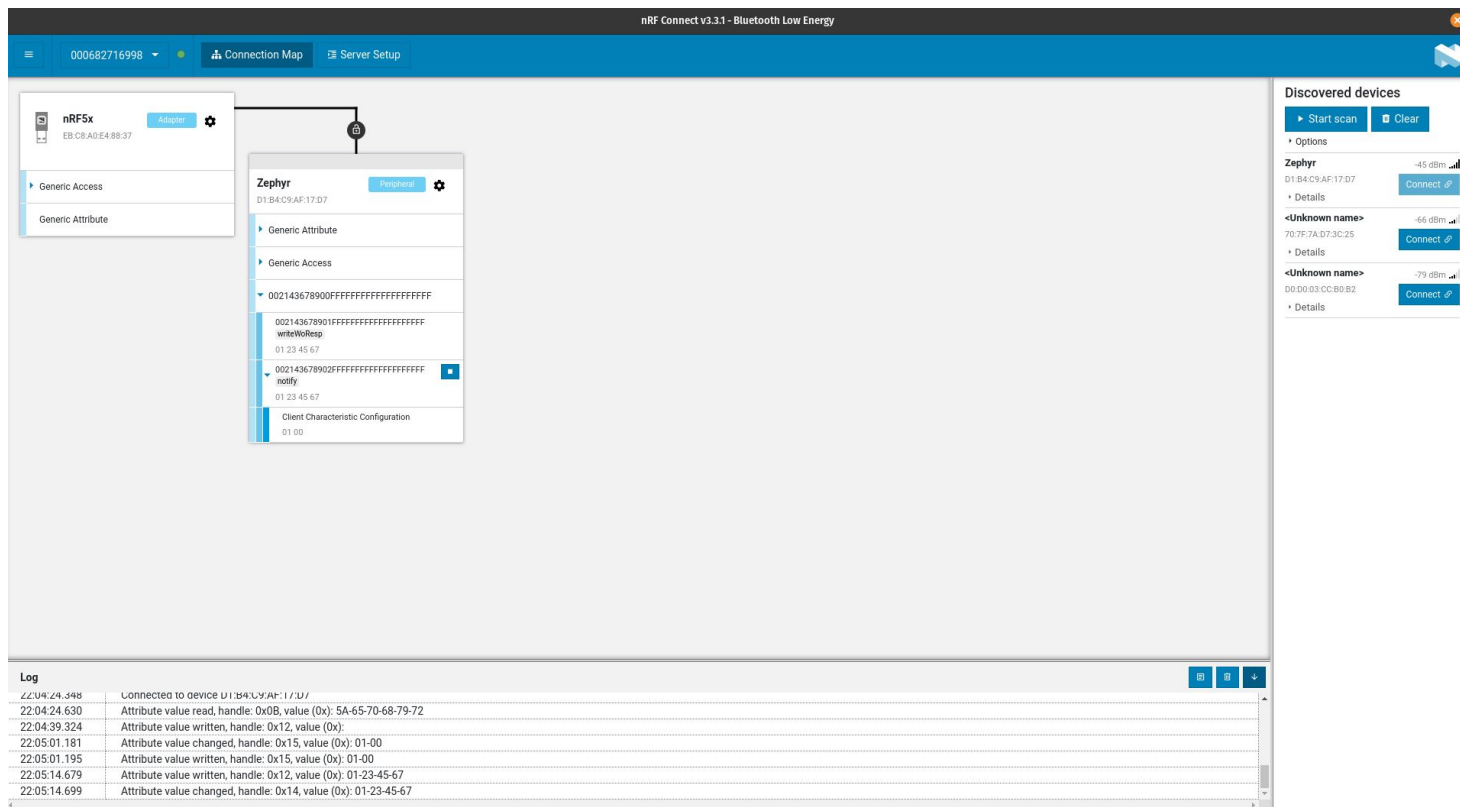
T, _reset_section, __start)

FORM_SPECIFIC_INIT)

will get unlocked when switch to main task
```




Interagindo com o BLE usando o NRF Connect (ou similar):



The screenshot displays the nRF Connect v3.3.1 - Bluetooth Low Energy interface. The main window shows a connection to a device named "Zephyr" (D1:B4:C9:AF:17:D7). The interface is divided into several sections:

- Left Panel:** Contains a sidebar with "nRF5x" (EB:C8:A0:E4:88:37) and a "Generic Access" section with a "Generic Attribute" entry.
- Center Panel:** Displays the "Zephyr" device details, including its MAC address and a list of services and characteristics. The "Generic Attribute" service is expanded, showing a "writeWithResp" characteristic with a value of "01 23 45 67".
- Right Panel:** Lists "Discovered devices" with their names, MAC addresses, and signal strengths. The "Zephyr" device is listed with a signal strength of -45 dBm.
- Bottom Panel:** A "Log" section showing a series of events, including connection status and attribute value changes.

Log:

Time	Event
22:04:24.348	Connected to device D1:B4:C9:AF:17:D7
22:04:24.630	Attribute value read, handle: 0x0B, value (0x): 5A-65-70-68-79-72
22:04:39.324	Attribute value written, handle: 0x12, value (0x):
22:05:01.181	Attribute value changed, handle: 0x15, value (0x): 01-00
22:05:01.195	Attribute value written, handle: 0x15, value (0x): 01-00
22:05:14.679	Attribute value written, handle: 0x12, value (0x): 01-23-45-67
22:05:14.699	Attribute value changed, handle: 0x14, value (0x): 01-23-45-67



Wrap-up:

- O fato do Zephyr ser um projeto complexo, não torna sua utilização necessariamente complexa;
- O Bluetooth Low Energy é muito mais que apenas Beacon;
- O VSCode com suas extensões pode ser uma alternativa ao Eclipse;



Wrap-up, continuação:

- O projeto exemplo dessa apresentação: https://github.com/uLipe/zephyr_ble_sample ;
- Cortex Debug plugin para o VSCode:
<https://marketplace.visualstudio.com/items?itemName=marus25.cortex-debug>;
- Guia completo de instalação para o Zephyr:
https://docs.zephyrproject.org/latest/getting_started/index.html ;
- Pacote de software Segger para uso com JLink (EDU ou PRO):
<https://www.segger.com/downloads/jlink/?p=805#J-LinkSoftwareAndDocumentationPack>
- NRF Connect, ferramenta Desktop para teste de dispositivos Bluetooth:
<https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Connect-for-desktop>
- CySmart, alternativa ao NRF Connect, somente Windows:
<https://www.cypress.com/documentation/software-and-drivers/cysmart-bluetooth-le-test-and-debug-tool>
- BLEFruit, Ferramenta de teste BLE em Python: https://github.com/adafruit/Adafruit_Python_BluefruitLE
- GATTLib, Lib para construção de aplicação BLE Central Side (*Nix apenas):<https://github.com/labapart/gattlib>



Dúvidas?



Contatos:

- Meu github (Aparece coisa útil de vez em quando): <https://github.com/uLipe>
- LinkedIn (Não, não to procurando trampo):
<https://www.linkedin.com/in/felipe-neves-a8092619/>
- Meu e-mail (Profissional, ou pessoal mesmo): ryukokki.felipe@gmail.com

Seminário de Sistemas Embarcados e IoT 2020

OBRIGADO!



Apoio

