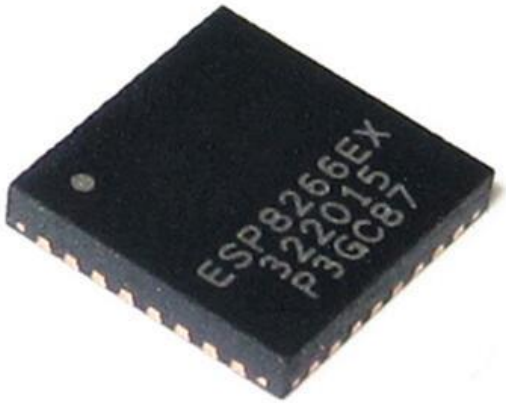
MICROCONTROLADORES

Unidade II - Introdução a ESP8266/ESP32
Aula 1

Prof. Ewerton Salvador

Microcontroladores escolhidos



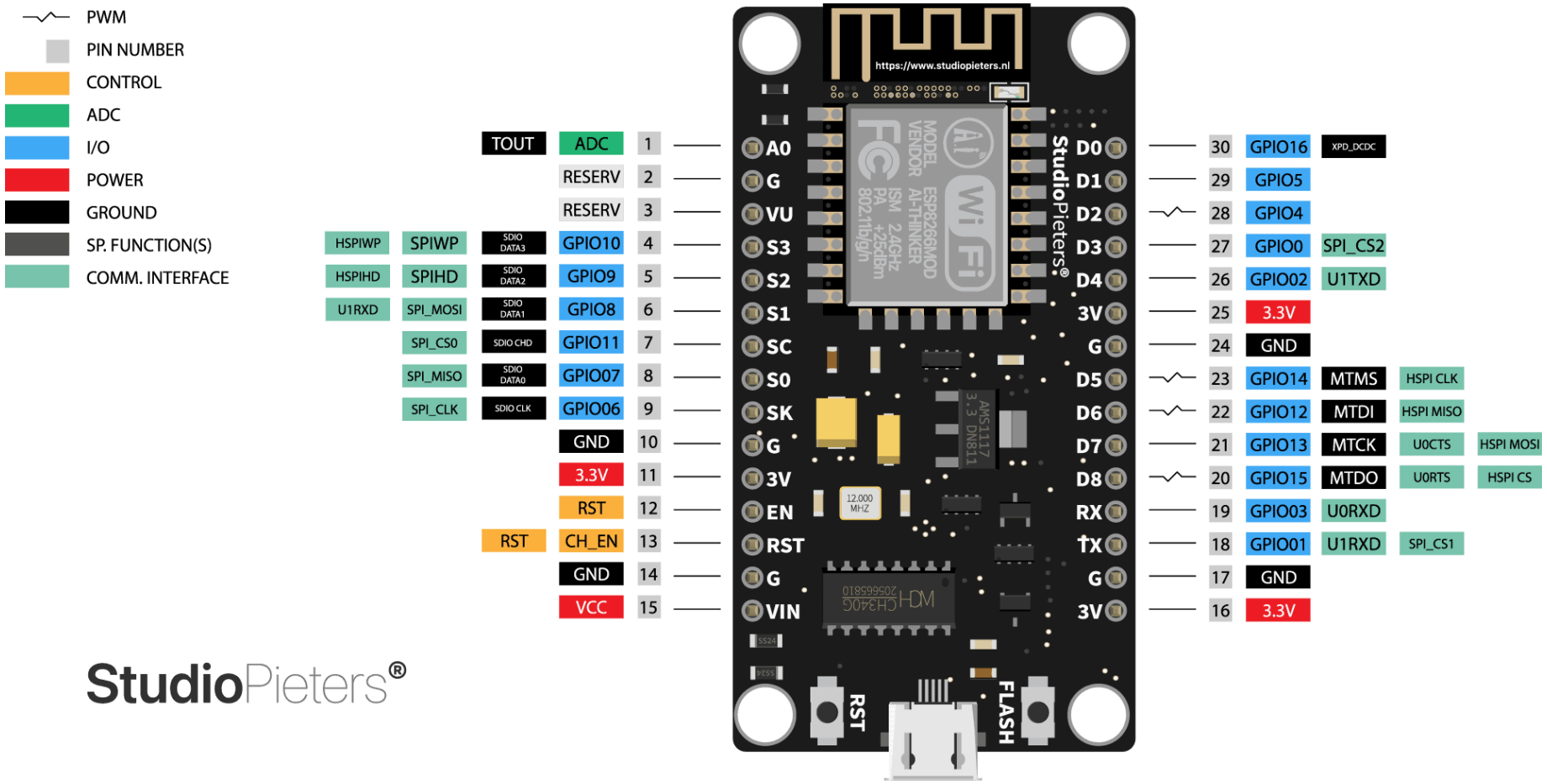
ESP 8266

ESP32

Microcontroladores escolhidos

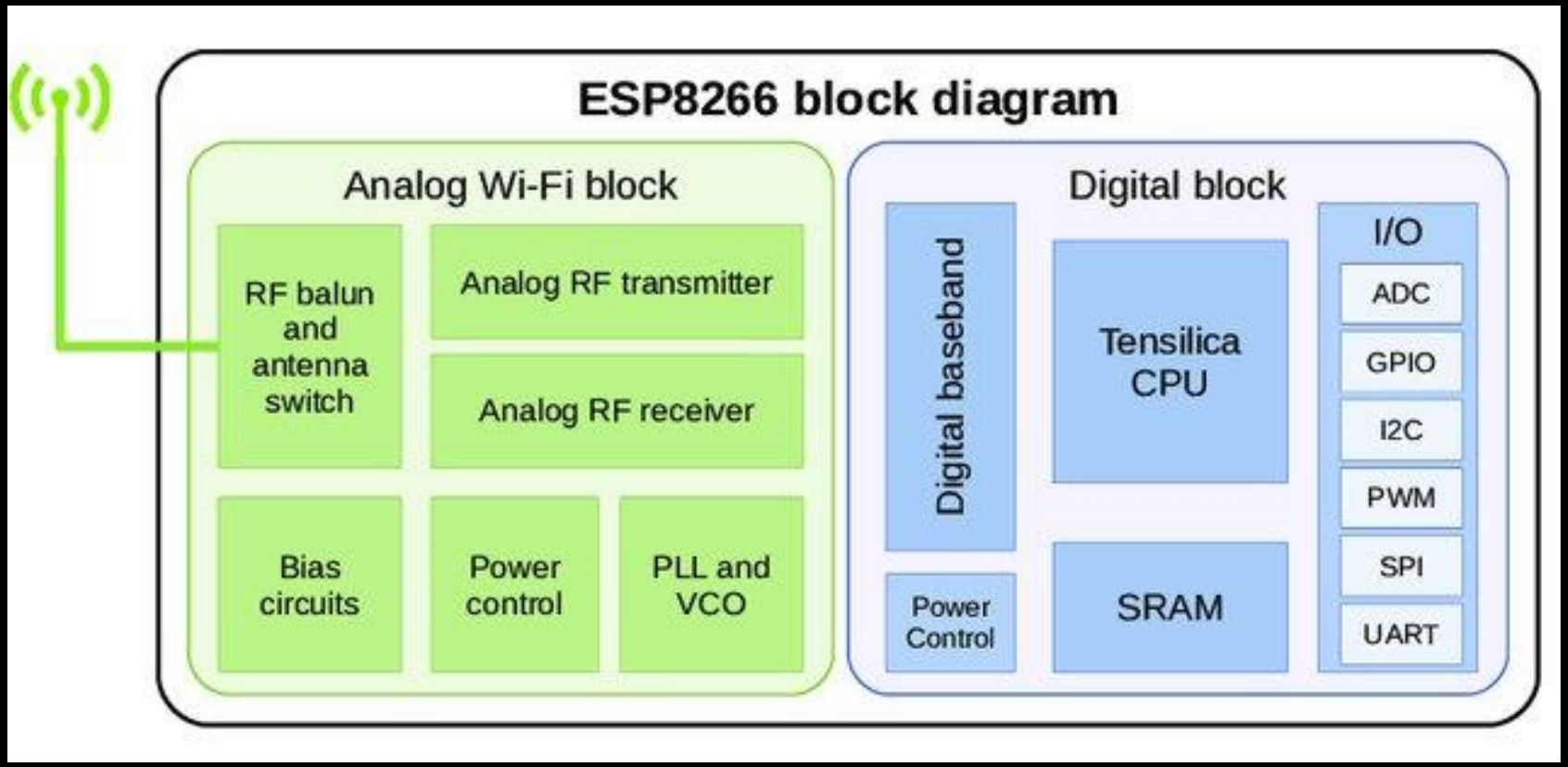
- Fabricados pela chinesa Espressif
- Incluem capacidade de comunicação Wi-Fi
- Arquitetura da CPU é a Cadence Tensilica Xtensa
- Baixo custo com alto poder de processamento (em comparação com soluções similares)
- Disponibilizados em diversos formatos de placas, com possibilidade de programação sem hardware adicional

NodeMCU (ESP8266)



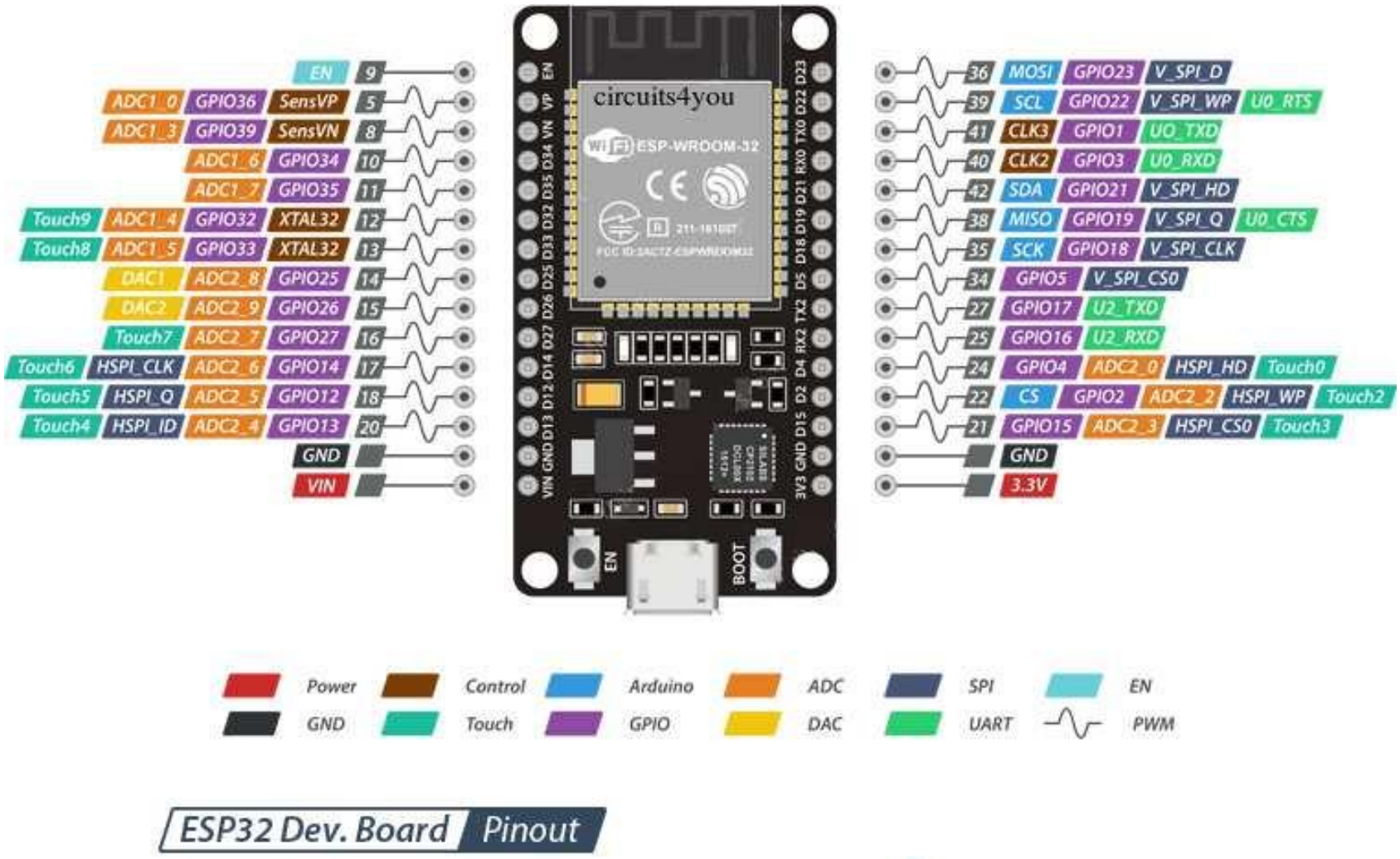
StudioPieters®

ESP8266 – Diagrama de blocos

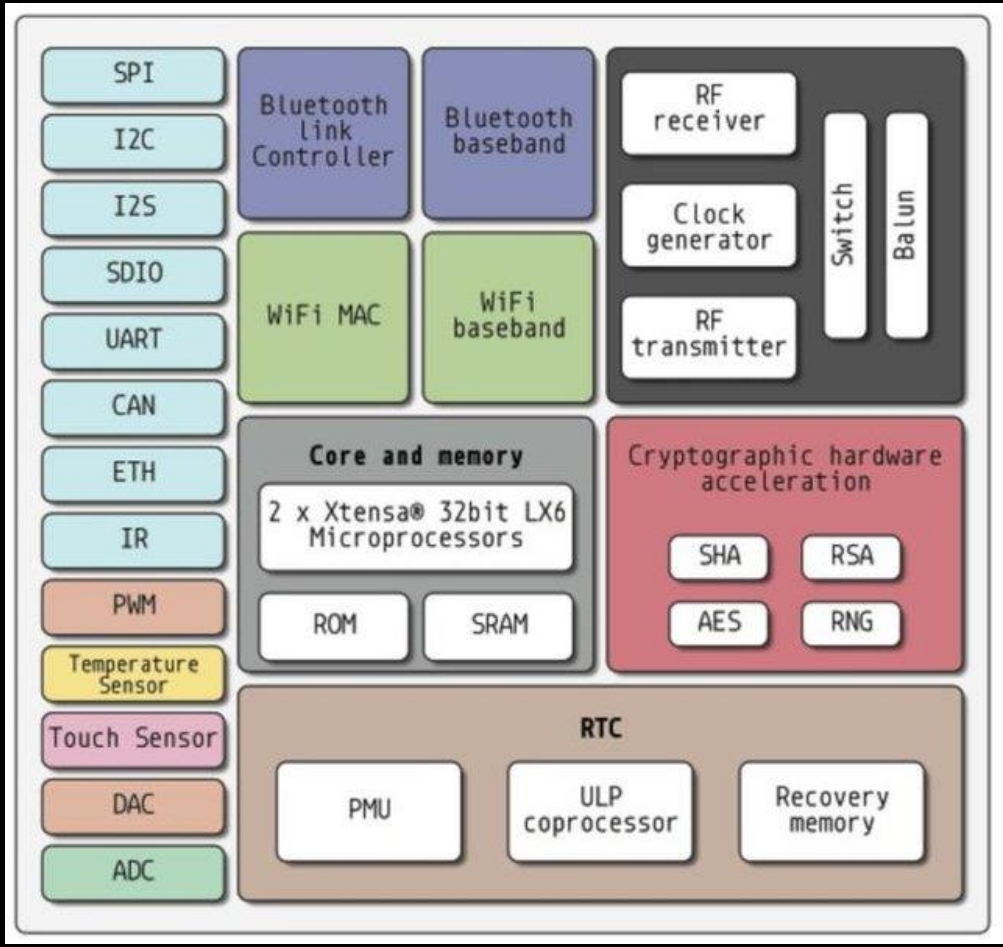


Fonte: García-Orellana, Carlos J., et al. "Low-power and low-cost environmental IoT electronic nose using initial action period measurements." *Sensors* v.19 n.14 (2019)

ESP32


EN 9
ADC1_0 GPIO36 SensVP 5
ADC1_3 GPIO39 SensVN 8
ADC1_6 GPIO34 10
ADC1_7 GPIO35 11
Touch9 ADC1_4 GPIO32 XTAL32 12
Touch8 ADC1_5 GPIO33 XTAL32 13
DAC1 ADC2_8 GPIO25 14
DAC2 ADC2_9 GPIO26 15
Touch7 ADC2_7 GPIO27 16
Touch6 HSPI_CLK ADC2_6 GPIO14 17
Touch5 HSPI_Q ADC2_5 GPIO12 18
Touch4 HSPI_ID ADC2_4 GPIO13 20
GND
VIN
circuits4you
ESP-WROOM-32
EN
BOOT
36 MOSI GPIO23 V_SPI_D
39 SCL GPIO22 V_SPI_WP U0_RTS
41 CLK3 GPIO1 U0_TXD
40 CLK2 GPIO3 U0_RXD
42 SDA GPIO21 V_SPI_HD
38 MISO GPIO19 V_SPI_Q U0_CTS
35 SCK GPIO18 V_SPI_CLK
34 GPIO5 V_SPI_CS0
27 GPIO17 U2_TXD
25 GPIO16 U2_RXD
24 GPIO4 ADC2_0 HSPI_HD Touch0
22 CS GPIO2 ADC2_2 HSPI_WP Touch2
21 GPIO15 ADC2_3 HSPI_CS0 Touch3
GND
3.3V
Power
Control
Arduino
ADC
SPI
EN
GND
Touch
GPIO
DAC
UART
PWM
ESP32 Dev. Board Pinout

ESP32 – Diagrama de blocos



Fonte: Ivkovic, J., and J. L. Ivkovic. "Analysis of the performance of the new generation of 32-bit Microcontrollers for IoT and Big Data Application." *Proceedings of the International Conference on Information Society and Technology (ICIST), Kopaonik, Serbia*. 2017.

ESP8266 vs. ESP32

Especificações	ESP8266	ESP32
MCU	Xtensa Single-Core 32-bit L106	Xtensa Dual-Core 32-bit LX6 600 DMIPS
802.11 b/g/n Wi-Fi	Sim, HT20	Sim, HT40
Bluetooth	Não	Bluetooth 4.2 e anteriores
Frequência Típica	80 MHz	160 MHz
SRAM	160 kBytes	512 kBytes
Flash	SPI Flash, até 16 Mbytes	SPI Flash, até 16 MBytes
GPIO	17	36
PWM Hardware/Software	Nenhum / 8 Canais	1 / 16 Canais
SPI / I2C / I2S / UART	2 / 1 / 2 / 2	4 / 2 / 2 / 2
ADC	10-bit	12-bit
CAN	Nenhum	1
Interface Ethernet MAC	Nenhum	1
Sensor de Toque	Nenhum	Sim
Sensor de Temperatura	Nenhum	Sim
Temperatura de Operação	-40°C – 125°C	-40°C – 125°C

Fonte: *https://www.cnx-software.com/2016/03/25/esp8266-and-esp32-differences-in-one-single-table/*

Linguagens/Ambientes de programação

- Baixo nível
  - Linguagem de máquina / Assembly
- Alto nível
  - Arduino IDE
  - Javascript
  - Lua
  - MicroPython
  - ESP-IDF SDK
  - Etc.

Linguagem de máquina

```
00000000  fc 31 c0 8e c0 8e d8 8e  d0 bc 00 7c 89 e6 bf 00  |.1.........|....|
00000010  06 b9 00 01 f3 a5 89 fd  b1 08 f3 ab fe 45 f2 e9  |.............E..|
00000020  00 8a f6 46 bb 20 75 08  84 d2 78 07 80 4e bb 40  |...F. u...x..N.@|
00000030  8a 56 ba 88 56 00 e8 fc  00 52 bb c2 07 31 d2 88  |.V..V....R...1..|
00000040  6f fc 0f a3 56 bb 73 19  8a 07 bf 87 07 b1 03 f2  |o...V.s.........|
00000050  ae 74 0e b1 0b f2 ae 83  c7 09 8a 0d 01 cf e8 c5  |.t..............|
00000060  00 42 80 c3 10 73 d8 58  2c 7f 3a 06 75 04 72 05  |.B...s.X,.:.u.r.|
00000070  48 74 0d 30 c0 04 b0 88  46 b8 bf b2 07 e8 a6 00  |Ht.0....F.......|
00000080  be 7b 07 e8 b2 00 8a 56  b9 4e e8 8e 00 eb 05 b0  |.{.....V.N......|
00000090  07 e8 b0 00 30 e4 cd 1a  89 d7 03 7e bc b4 01 cd  |....0......~....|
000000a0  16 75 0d 30 e4 cd 1a 39  fa 72 f2 8a 46 b9 eb 16  |.u.0...9.r..F...|
000000b0  30 e4 cd 16 88 e0 3c 1c  74 f1 2c 3b 3c 04 76 06  |0.....<.t.,;<.v.|
000000c0  2c c7 3c 04 77 c9 98 0f  a3 46 0c 73 c2 88 46 b9  |,.<.w....F.s..F.|
000000d0  be 00 08 8a 14 89 f3 3c  04 9c 74 0a c0 e0 04 05  |.......<..t.....|
000000e0  be 07 93 c6 07 80 53 f6  46 bb 40 75 08 bb 00 06  |......S.F.@u....|
000000f0  b4 03 e8 59 00 5e 9d 75  06 8a 56 b8 80 ea 30 bb  |...Y.^.u..V...0.|
00000100  00 7c b4 02 e8 47 00 72  86 81 bf fe 01 55 aa 0f  |.|...G.r.....U..|
00000110  85 7c ff be 85 07 e8 19  00 ff e3 b0 46 e8 24 00  |.|.........F.$.|
00000120  b0 31 00 d0 eb 17 0f ab  56 0c be 78 07 e8 eb ff  |.1......V..x....|
00000130  89 fe e8 03 00 be 85 07  ac a8 80 75 05 e8 04 00  |...........u....|
00000140  eb f6 24 7f 53 bb 07 00  b4 0e cd 10 5b c3 8a 74  |..$.S.......[..t|
00000150  01 8b 4c 02 b0 01 56 89  e7 f6 46 bb 80 74 13 66  |..L...V...F..t.f|
00000160  6a 00 66 ff 74 08 06 53  6a 01 6a 10 89 e6 48 80  |j.f.t..Sj.j...H.|
00000170  cc 40 cd 13 89 fc 5e c3  20 20 a0 0a 44 65 66 61  |.@....^.  ..Defa|
00000180  75 6c 74 3a a0 0d 8a 00  05 0f 01 06 07 0b 0c 0e  |ult:............|
00000190  83 a5 a6 a9 0d 0c 0b 0a  09 08 0a 0e 11 10 01 3f  |...............?|
000001a0  bf 44 4f d3 4c 69 6e 75  f8 46 72 65 65 42 53 c4  |.DO.Linu.FreeBS.|
000001b0  66 bb 44 72 69 76 65 20  00 00 80 8f b6 00 00 00  |f.Drive ........|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200
```

Linguagem de Máquina

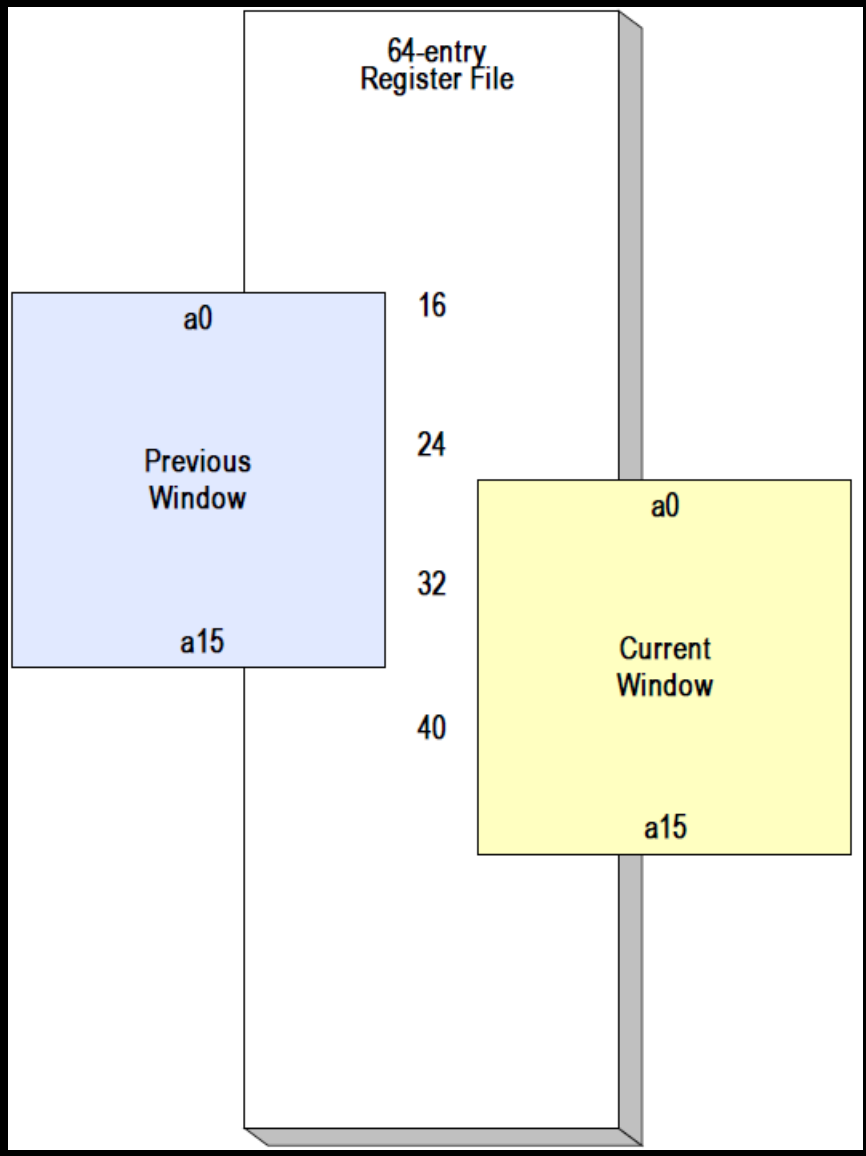
- Arquitetura de 32 bits com instruções de 16 bits e 24 bits (pensando em economia do tamanho do código)
- Arquitetura RISC
- Memória pode ser acessada por byte, meia palavra (16 bits) e palavra (32 bits)

Assembly

```
1   .align 4
2           .global func
3   func:
4           .frame a1, 32
5   .LBB1_func:
6           entry a1,32
7           l32i.n a10,a1,40
8   .LBB2_func:
9           l32i.n a8,a1,32
10          add.n a12,a5,a6
11          add.n a11,a2,a3
12          l32i.n a2,a1,36
13          add.n a11,a4,a11
14          add.n a11,a11,a12
15          add.n a8,a8,a7
16          add.n a2,a2,a10
17          add.n a8,a8,a11
18          add.n a2,a2,a8
19          retw.n
```

Assembly

- Arquivo de registradores de propósito geral chamado AR
  - 32 ou 64 registradores
  - 16 registradores acessados por vez, através de uma janela deslizante
  - Movimentação da janela se dá por meio de uma chamada de função
- Outros registradores:
  - PC, ACCHI e ACCLO (acumuladores), BR registradores booleanos), FR (Registradores de ponto flutuante), SAR (registrador de deslocamento), etc.



Assembly

Instruction Category	Instructions
Load	L8UI, L16SI, L16UI, L32I, L32R
Store	S8I, S16I, S32I
Memory Ordering	MEMW, EXTW
Jump, Call	CALL0, CALLX0, RET J, JX
Conditional Branch	BALL, BNALL, BANY, BNONE BBC, BBCI, BBS, BBSI BEQ, BEQI, BEQZ BNE, BNEI, BNEZ BGE, BGEI, BGEU, BGEUI, BGEZ BLT, BLTI, BLTU, BLTUI, BLTZ
Move	MOVI, MOVEQZ, MOVGEZ, MOVLTZ, MOVNEZ
Arithmetic	ADDI, ADDMI, ADD, ADDX2, ADDX4, ADDX8, SUB, SUBX2, SUBX4, SUBX8, NEG, ABS
Bitwise Logical	AND, OR, XOR
Shift	EXTUI, SRLI, SRAI, SLLI SRC, SLL, SRL, SRA SSL, SSR, SSAI, SSA8B, SSA8L
Processor Control	RSR, WSR, XSR, RUR, WUR, ISYNC, RSYNC, ESYNC, DSYNC

- Algumas instruções possuem terminação “.n”, indicando uma variante “curta” de 16 bits da referida instrução

Assembly

- Uma forma simples de se utilizar Assembly na programação da ESP8266/ESP32 é incorporando o código direto em um programa em C:

```
void app_main(void)
{
    asm("movi.n a1, 100");
    asm("repetir:");
    asm("  addi a1, a1, -1");
    asm("bnez.n a1, repetir");
}
```

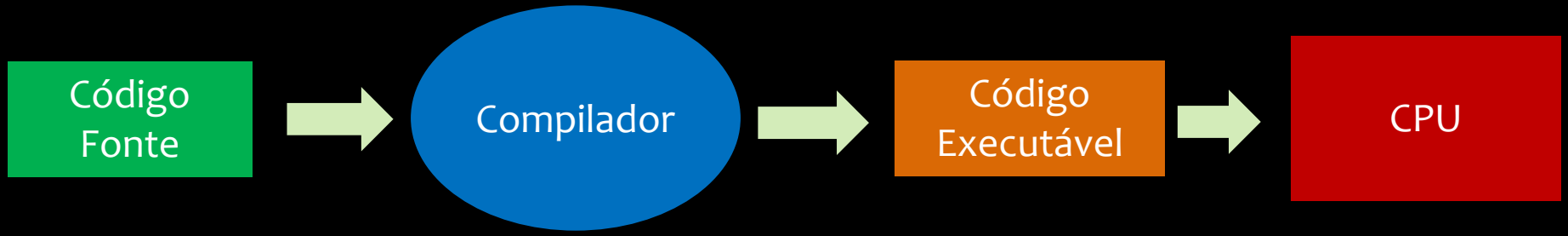
Assembly

- Programação em Assembly para ESP, realisticamente falando, é utilizada em situações muito específicas. Por exemplo:
  - Restrições extremas de desempenho;
  - Acesso a registradores, objetivando um nível ainda mais detalhado de controle do microcontrolador (ex.: registrador CCOUNT, que conta ciclos)

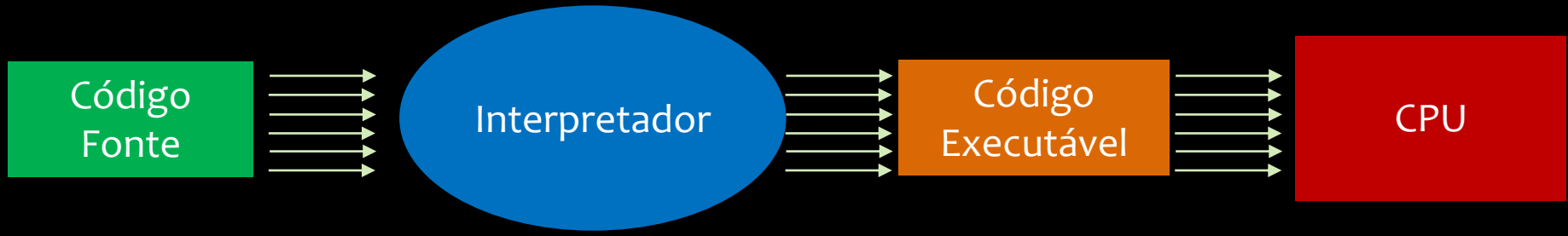
- Cadence aparentemente exige negociação e assinatura de NDA (*Non Disclosure Agreement*) para liberar documentação atualizada sobre ISA

Programação em Alto Nível

- Linguagens de alto nível precisam ser traduzidas para linguagem de máquina
- Compilação:



- Interpretação:



Arduino IDE

- Arduino IDE é anterior aos microcontroladores ESP8266/ESP32
- Foco em facilidade de desenvolvimento
- Foram desenvolvidas bibliotecas do Arduino para fornecer suporte a ESP8266/ESP32
  - Essas bibliotecas invocam funções da SDK nativa da ESP8266/ESP32

Arduino IDE

- Pontos positivos:
  - Ambiente de desenvolvimento comum para vários microcontroladores diferentes
  - Grande diversidade de bibliotecas de funções desenvolvidas pela comunidade

- Pontos negativos:
  - Por ter sido pensada originalmente para o Arduino, não necessariamente se adapta com facilidade a outros tipos de microcontroladores
  - Programadores dependem de bibliotecas para outros microcontroladores estarem atualizadas (também ocorre em **qualquer** ambiente de programação que não seja nativo)

Javascript

- Linguagem interpretada amplamente utilizada em desenvolvimento web
- Fracamente tipada, orientada a objetos, suporte a arquitetura orientada a eventos
- O projeto open source **Espruino** provê um interpretador Javascript para microcontroladores
  - Já foi implementado para microcontroladores ARM Cortex M3/M4

Lua

- Linguagem popular de scripting
- Pode ser utilizada em ESP8266 através do firmware NodeMCU Lua
  - Construída sobre a SDK da Espressif (sem RTOS)
- No caso da ESP32, existe um componente da ESP-IDF chamado Lua-RTOS

MicroPython

- Linguagem de programação compacta, derivada do Python 3, voltada para microcontroladores
- Inclui diversas subfunções da biblioteca padrão Python
- Necessário instalar o firmware interpretador na ESP8266/ESP32

ESP8266 SDK / ESP-IDF

- Espressif oferece kits oficiais de desenvolvimento nativo para os microcontroladores ESP8266 e ESP32
- Utiliza linguagem de programação C e C++
- Open source
- Fornece um maior nível de controle das funcionalidades do microcontrolador em comparação com outras abordagens de programação em alto nível

ESP8266 SDK / ESP-IDF

- ESP8266
  - ESP8266 RTOS SDK
  - ESP8266 NONOS SDK
- ESP32
  - ESP-IDF (*IoT Development Framewok*) – RTOS SDK
- ESP8266 foi refatorado para apresentar o mesmo estilo de programação da ESP-IDF

ESP8266 SDK / ESP-IDF

- Framework passou por transformações nos últimos anos
  - **ESP-IDF**
    - Versão 1.0: 02/12/2016
    - Versão 2.0: 06/04/2017
    - Versão 3.0: 24/04/2018
    - Versão 4.0: 10/02/2020
    - Atualmente na versão 4.3.3 (13/06/2022)
  - **ESP8266-RTOS-SDK**
    - Versão 2.0: 02/04/2018
    - Versão 3.0: 11/09/2018 (adaptação para o estilo da ESP-IDF)
    - Atualmente na versão 3.4 (08/04/2021)

ESP8266 SDK / ESP-IDF

- Quantidade de informação disponível sobre os SDKs atualmente é limitada
  - Mudanças no SDK fazem com que parte da documentação existente se torne obsoleta
  - Barreira linguística chinês -> inglês

- Provavelmente as melhores fontes de informações atualizadas para programação sejam as abaixo:
  - ESP8266-RTOS-SDK: https://docs.espressif.com/projects/esp8266-rtos-sdk/en/latest/get-started/index.html
  - ESP-IDF: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html
  - Exemplos contidos nos próprios SDKs

Apresentação do FreeRTOS

- Tanto o ESP-IDF quanto uma das duas versões do ESP8266-SDK incluem o uso de uma versão customizada do FreeRTOS
  - As páginas dos SDKs da Espressif possuem uma descrição das principais diferenças entre o FreeRTOS *vanilla* e o FreeRTOS customizado

- É necessário se compreender ao menos alguns conceitos básicos do FreeRTOS para se programar com os SDKs nativos de ESP8266 e ESP32

Real Time OS

- Sistemas Operacionais fornecem uma camada de software para permitir um ambiente de programação que abstrai a complexidade do hardware e facilita o desenvolvimento e manutenção de programas

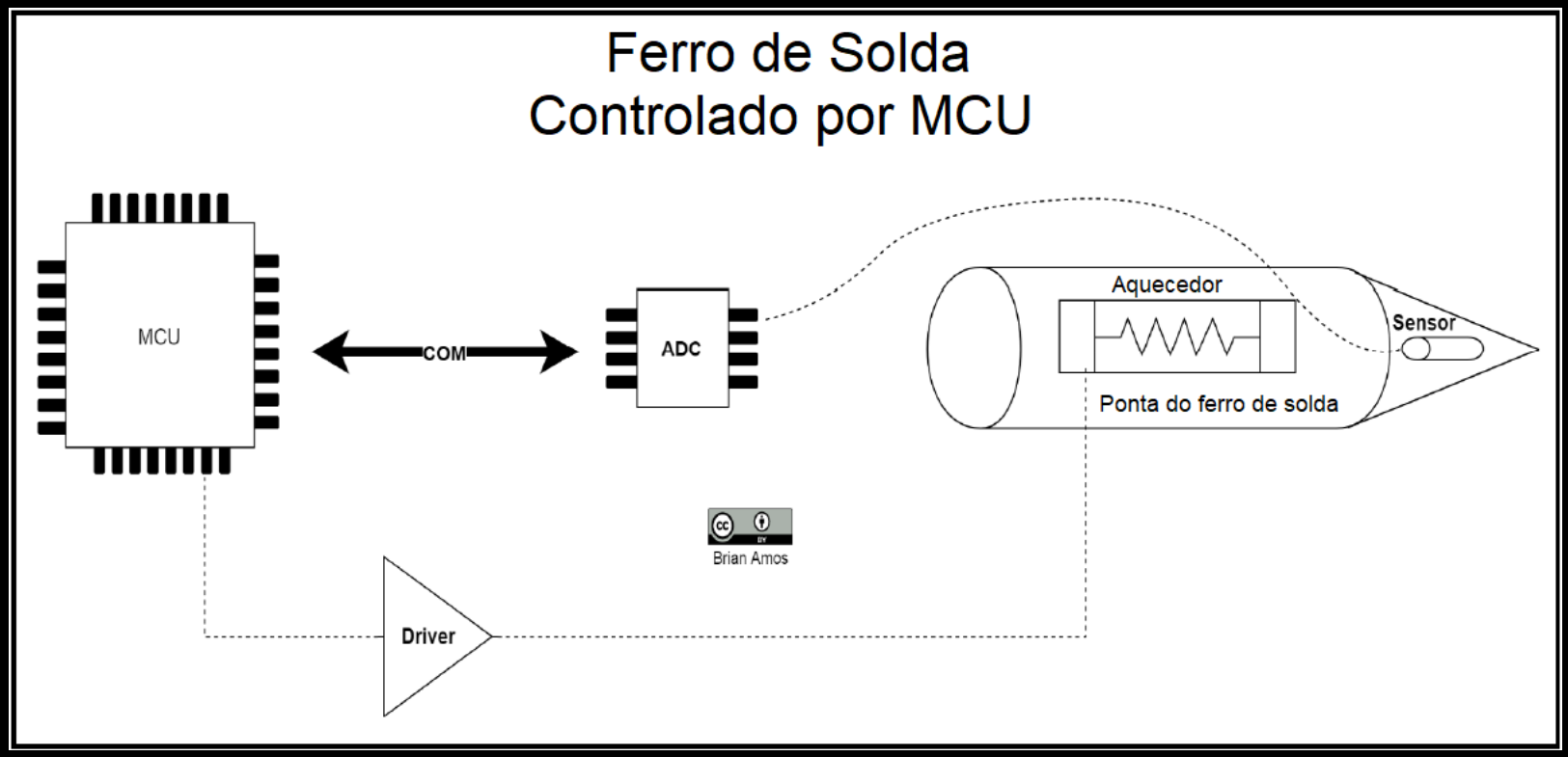
  - Escalonamento de processos/tarefas
  - Gerenciamento de Memória
  - Gerenciamento de Dispositivos de Entrada/Saída
  - Sistemas de Arquivos
  - Etc.

Real Time OS

- Quando um SO fornece uma forma determinística para execução de um código ele pode ser considerado um SO de tempo real (RTOS)

Real Time OS

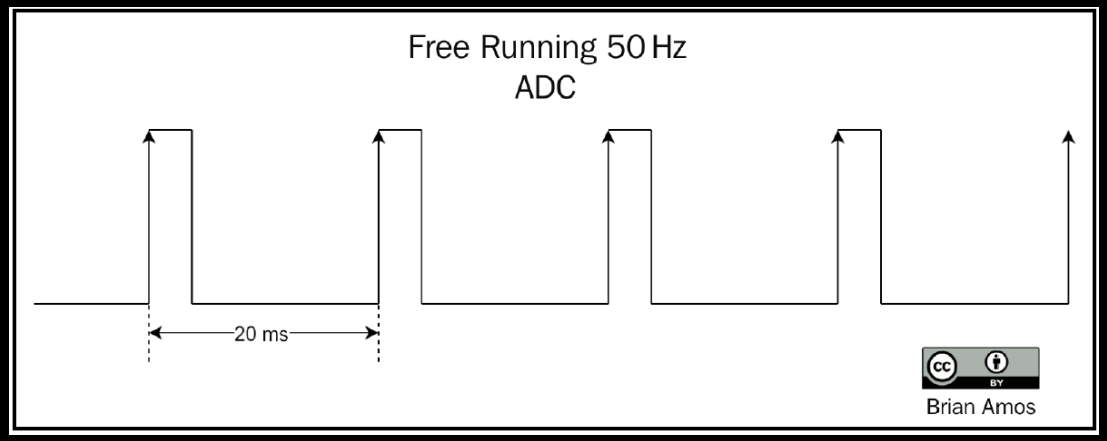
- Consideremos o sistema abaixo para exemplificar o que seria um requisito de tempo real:



Adaptado de: AMOS, Brian. **Hands-On RTOS with Microcontrollers: Building Real-time Embedded Systems Using FreeRTOS, STM32 MCUs, and SEGGER Debug Tools**. Packt Publishing, 2020.

Real Time OS

- 50 amostragens do ADC por segundo (50Hz)

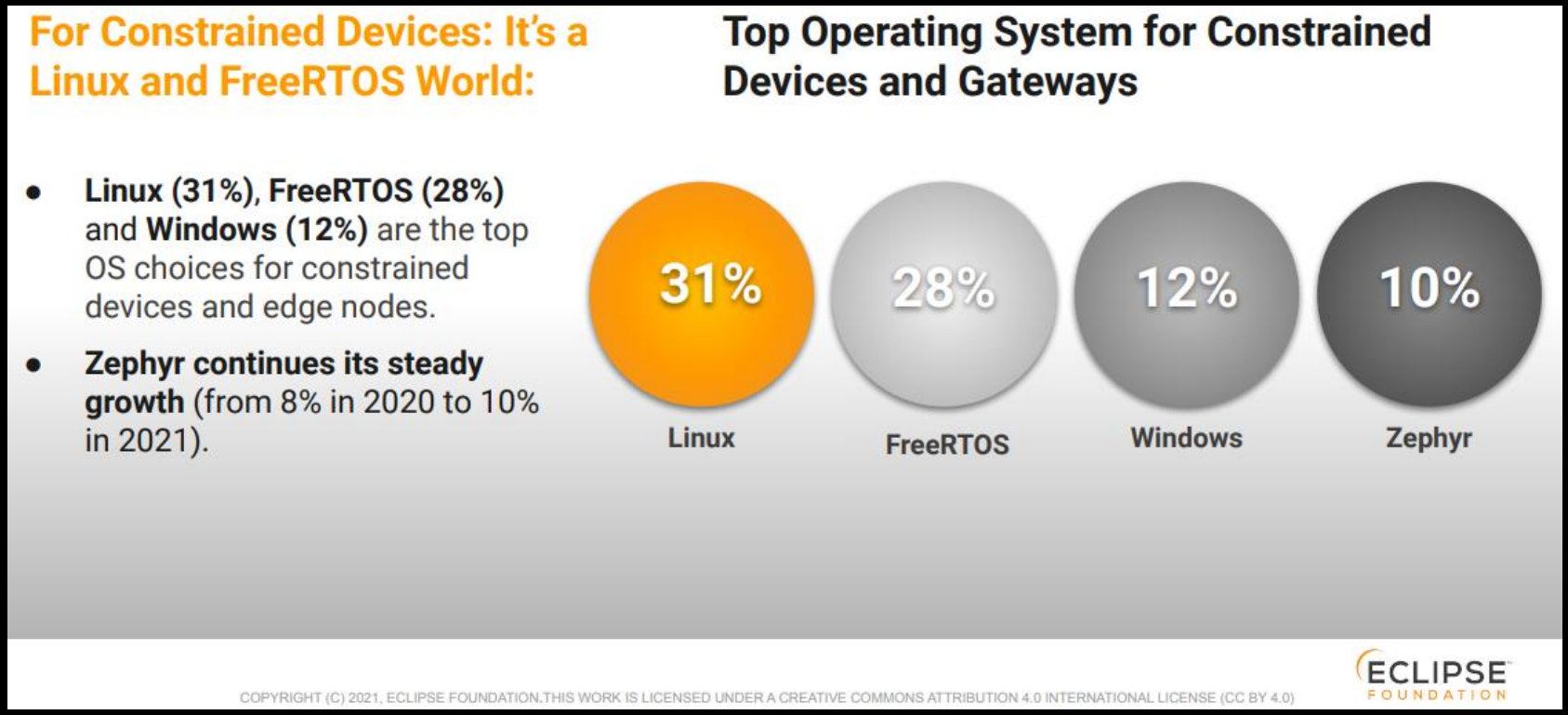


Adaptado de: AMOS, Brian. **Hands-On RTOS with Microcontrollers: Building Real-time Embedded Systems Using FreeRTOS, STM32 MCUs, and SEGGER Debug Tools**. Packt Publishing, 2020.

- Controle do aquecedor funciona a 5Hz
- ADC ativa uma linha de hardware para avisar que uma conversão está pronta
  - MCU tem até 20ms para transferir os dados do ADC para sua memória interna
- MCU tem 200ms para calcular os valores atualizados para o aquecedor

FreeRTOS

- FreeRTOS é uma das implementações de RTOS mais populares para MCUs



FreeRTOS

- Lançado em 2003
- Inicialmente desenvolvido por Richard Barry, que posteriormente fundou a Real Time Engineers Ltd.
- Open Source
- Escrito na linguagem C
- Atualmente disponível para 35 plataformas de microcontroladores
- Adquirido pela Amazon (AWS) em 2017

Quando utilizar um RTOS?

- RTOS não é a única maneira de se obter comportamento de tempo real!
  - É uma solução possível, mas não a única solução
- Por exemplo, problemas extremamente simples podem ser resolvidos com hardware (algumas portas lógicas)
- Problemas intermediários podem ser inicialmente resolvidos com hardware programável (FPGA ou CPLD), podendo evoluir para ASIC

Quando utilizar um RTOS?

- RTOS costumam ser uma excelente solução para casos mais complexos, envolvendo várias tarefas, comunicação em rede (Wi-Fi, CAN, etc.)

- Níveis ainda maiores de complexidades podem demandar a utilização de um sistema operacional de propósito mais geral, ao invés de um RTOS

- O mais importante: não existe solução única! É necessário mente aberta e escolha de soluções coerentes com requisitos, habilidades, etc.
  - Por isso que Eng. de Computação exige o aprendizado de várias abordagens para o desenvolvimento de hardware/software