O ESP8266

Uma introdução à Internet das Coisas

Gabriel M. Melo

EMakers

Núcleo de estudos em Sistemas Embarcados,
Internet das Coisas e Otimização
UFLA

MELO, Gabriel M. de.
ESP8266 - Uma Introdução ao Desenvolvimento em IoT. / Gabriel M. de Melo. – Lavras-MG: Universidade Federal de Lavras, 2017.

Agradecimentos

Este livro é fruto da soma de esforços mútuos do núcleo de estudos *EMakers*[1] em integrar e disseminar o conhecimento na área de Sistemas Embarcados, Otimização e Internet das Coisas para o estudantes e entusiastas brasileiros. Um agradecimento especial ao professor - e coordenador - Bruno de Abreu Silva que, pacientemente, vem nos auxiliando e orientando durante nossos projetos e almejos. Agradeço também ao Departamento de Ciência da Computação da UFLA, que, com portas abertas, sempre nos incentiva e motiva a trabalhar.

1 <emakers.ufla.br>

Lista de ilustrações

Lista de tabelas

Sumário

Introdução

Este livro dá início a uma série de documentos didáticos voltados à eletrônica digital e analógica, programação, otimização e gestão pessoal produzidos pelo grupo EMakers na Universidade Federal de Lavras.

O livro foi estruturado em **5** capítulos, subdivididos em seções que aprofundam uma temática, contendo trecho de códigos *e*/*ou* tabelas.

O primeiro capítulo, denominado **Contextualizando**, é destinado à introduzir o leitor aos principais conceitos necessários para a compreesão básica do restante do livro. São vistos contextos históricos e uma breve revisão sobre eletrônica digital básica.

O capítulo seguinte, **O ESP8266**, apresenta o microcontrolador tema da obra e seus principais aspectos de *hardware*, como *GPIOs*, interfaces de comunicação, protocolos, memória, consumo e demais componentes. São usadas tabelas, figuras e gráficos para apresentação da plataforma.

Em **Firmware**, é discorrido acerca dos *softwares* básicos do microcontrolador. O conjunto de instruções padrão, os firmwares mais utilizados e suas linguagens de programação. É neste capítulo que o leitor será guiado no desenvolvimento e *upload* de seu primeiro programa na plataforma.

No quarto capítulo, é realizada uma abordagem comparativa entre duas disseminadas plataformas de desenvolvimento *IoT*: **ESP8266 VS Arduino**. São ilustradas suas diferenças de *harware*, desempenho, praticidade e consumo entre modelos de custos compatíveis.

O último capítulo é destinado à **Aplicações e Exemplos** de soluções comuns que fazem uso da plataforma ESP8266. Implementações comentadas utilizando *python*, *Lua* e *Arduino* são apresentadas, além dos esquemáticos elétricos utilizados em cada problema.

Algumas notações são padronizadas para referir termos repetitivos no decorrer do livro:

nomeDoPino* ⊥ *númeroDoPino Sempre que citado um pino do microcontrolador, a notação será mantida; onde *nomeDoPino* é o nome impresso na *PCB* referente ao pino; e *númeroDoPino* é o número identificador de cada pino que consta no datasheet. Como um exemplo, o leitor se deparará inúmeras vezes com o *GPIO0 ⊥ 15*, que é um pino de suma importância para o devido funcionamento da plataforma.

Todas as tabelas foram retiradas - *e traduzidas* - do datasheet fornecido pela própria *Expressif Systems*, empresa desenvolvedora da plataforma. As informações referentes à linguagem *Arduino* foram retiradas da documentação oficial.

CAPÍTULO 1

Contextualizando

O MUNDO, bem como suas ferramentas, adapta aos seus mais variados e históricos momentos. Das macros transformações da sociedade às micros, o homem se motivou pela busca do conhecimento do planeta, da vida e de si mesmo, fazendo-o ser humano. Desta forma, com o acúmulo de fatos observados e através da intercomunicação, o ser humano pôde compartilhar experiências, se informar.

A informação e seus meios de produção, armazenamento, transmissão, acesso, segurança e uso foram drasticamente adulterados com o advento do mais poderoso meio de comunicação, a *Internet*. Em menos de 30 anos, a *Internet* impulsionou a aproximação - e a aculturação - de diferentes povos, línguas e estudos. Por uma perspectiva discente, a Internet democratizou, publicizou e agilizou o conhecimento.

1.1 O Termo *Internet of Things*

Um termo, consideravelmente recente, tomou espaço nas discussões de grandes empresas e organizações, a *Internet of Things* (ou, simplesmente, *IoT*). O *IoT* vem gerando uma mudança na percepção do ambiente, onde o conceito **M2M** é uma realidade e os elementos comuns do dia-a-dia comunicam entre si, sendo são capazes de tomar decisões mais completas. Essa vertente vem de encontro com sistemas de automação, embarcados e até otimização do tempo e organização pessoal.

Este livro vem como um introdutório compilado de - *traduzidas* - experiências referentes à uma plataforma didática prática, robusta e ideal para entusiasmados com o *IoT*, baseada em um microcontrolador chinês : o **ESP8266**.

1.2 Microcontroladores

Os microcontroladores (ou *MCUs*) são dispositivos programáveis de baixo processamento que podem monitorar e modificar o ambiente através de pinos digitais *e*/*ou* analógicos de entrada/saída. Um microcontrolador possui, essencialmente, uma unidade central de processamento (*CPU*), memórias de dados e programa, contador interno e periféricos de entrada/saída.

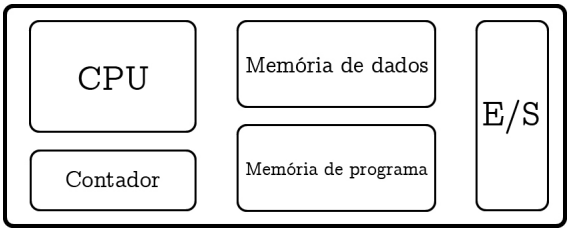

Figura 1 – Microcontrolador

A **CPU**, ou processador, é a responsável por todo o processamento - *sequencial* - das instruções contidas na memória de programa.

A **memória de dados**, geralmente uma RAM, é utilizada essencialmente para mapear os registradores, de controle e de dados.

A **memória de programa**, uma ROM, contém as intruções a serem lidas e executadas pela *CPU*.

A **interface de entrada e saída de periféricos** é, de fato, a responsável pela comunicação e interação do MCU com o ambiente externo. É através dela que são conectados sensores, atuadores, displays e memórias secundárias.

Amplamente utilizado em sistemas embarcados, os *MCUs* apresentam um baixo consumo energético e tamanho extremamente reduzido quando comparado com computadores pessoais (*PCs*).

Os microcontroladores tiveram um grande impulso e popularidade com o advento do Arduino, que utiliza microcontroladores da *ATMEL*. Outras empresas que desenvolvem microcontroladores em larga escala são a *Texas Instruments* (MSP), *Microchip* (PIC) e *Expressif Systems* (ESP8266).

Você sabia?

O *Kinetis KL03*, da *Freescale*, é o atual menor microcontrolador de 32-bits do mundo com dimensões de 1.6 x 2.0 mm!

1.3 Sinais Analógicos e Digitais

O mundo eletrônico comunica entre si através de sinais elétricos dos quais podem ser descritos como **digitais** ou **analógicos**.

1.3.1 Sinal Analógico

Sinais digitais representam **valores contínuos** em função do tempo. Geralmente, em circuitos digitais, os sinais analógicos expressam a saída de sensores.

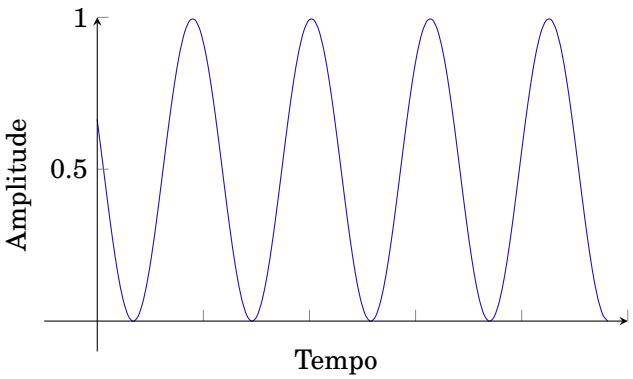

Figura 2 – Sinal Analógico

1.3.2 Sinal Digital

Os sinais digitais representam, ao contrário dos analógicos, **valores discretos**. Na grande maioria dos circuitos digitais, podem assumir apenas dois valores: **0** e **1**.

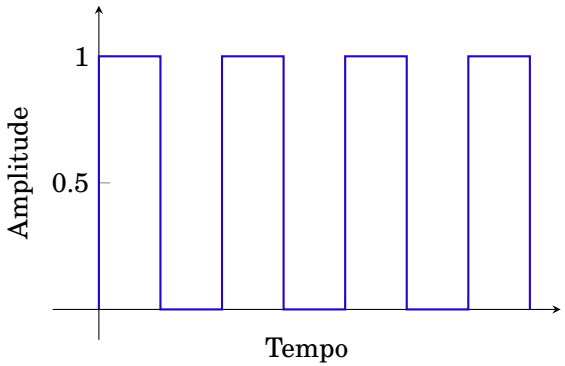

Figura 3 – Sinal Digital

1.3.2.1 Clock

Gerado por um gerador de clock (geralmente um cristal oscilador), o clock é um valor que varia entre nível lógico alto e baixo - *onda quadrada* em uma frequência específica e é usado na coordenação de ações de circuitos digitais.

1.4 *Wi-Fi*

O protocolo de rede sem fio *IEEE 802.11*, mais conhecido como *Wi-Fi*, alterou o perfil de uso da *Internet*, a conectando a cada vez mais dispositivos e pessoas.

Varios padrões do protocolo foram desenvolvidos, com diferentes frequências, largura de banda, velocidades de transferência e alcance. A Tabela 1 ilustra os principais protocolos *Wi-Fi*:

Tabela 1 – Principais protocolos Wi-Fi

Protocolo	**Freq.** (GHz)	**Larg.** de banda (MHz)	**Alcance** *indoor* (m)	**Alcance** *outdoor* (m)
802.11	2.4	22	20	100
802.11b	2.4	22	35	140
802.11g	2.4	20	38	140
802.11n	2.4/5	20/40	70/70	250/250

CAPÍTULO 2

O ESP8266

O ESP8266 é um microcontrolador de 32-bits com *Wi-Fi* integrado desenvolvido pela chinesa *Espressif Systems* e surgido, em meados de 2014, para suprir a contínua demanda por uma plataforma de **baixo consumo** energético, **tamanho reduzido** e **desempenho confiável** na industria de *IoT*.

Assim como a maioria dos modelos Arduino, o ESP possui GPIOs (Pinos de entrada e saída de propósito geral) e suporte a PWM (Modulação por largura de pulso). O upload do firmware é feito também pela UART (RX/TX), porém o ESP8266 conta com upload OTA (over-the-air), que é a gravação através de uma rede.

2.1 CPU

O ESP8266 utiliza o processador Tensilica L106 32-bit, RISC e que possui velocidade de clock de 80MHz.

Na Prática...

Facilmente, podemos alterar - *overclock* - da *CPU* para 160MHz! Para isso, basta adicionarmos o seguinte código no escopo geral do código:

```
extern "C" {
    bool system\_update\_cpu\_freq(int);
}
```

E na função *setup()*:

```
void setup(){
    ...
    system\_update\_cpu\_freq(160);
    ...
}
```

2.2 Modem *Wi-Fi*

Com um completo e autônomo sistema de conexão *Wi-Fi* (suporte a *802.11b/g/n/e/i*), o *ESP8266* pode tanto estabelecer uma rede própria, em modo **AP**, ou um se conectar a um *host*, em modo **STA**. Possui ainda o modo **AP_STA**, que cria uma rede, mas também acessa uma *station*, simultaneamente.

Pode ser implementado como um "módulo *WiFi*"em conjunto com qualquer microcontrolador e comunicá-los através de interfaces **SPI**, **I2C** e **UART** disponíveis na placa.

2.3 Alimentação

A faixa de tensão de operação do *ESP8266* é de **2.5V** à **3.6V**, sendo necessário o fornecimento de, ao menos, **500mA** de corrente.

CUIDADO!

O ESP8266 **não é** tolerante à 5V. Alimentá-lo com essa tensão pode - *e vai* - danificar seu dispositivo.

2.4 Modos de operação

O *ESP8266* possui **três** modos de boot:

Tabela 2 – Modos de boot

UART mode	Modo deve ser selecionado para realizar o upload de um novo *firmware* no MCU. A **GPIO0** deve estar em **GND** / **GPIO2 em VCC**.
Flash mode	Neste modo, o boot é realizado da memória *flash*. O último firmware gravado será executado. A **GPIO0** deve estar em **VCC** / **GPIO2** em **VCC**
SDIO mode	Quando habilitado, o MCU realiza o boot não da memória *flash*, mas sim de um cartão SD, usando protocolo *SPI*. A **GPIO15** deve estar em **VCC**.

2.5 GPIOS

O *ESP8266* possui 17 pinos GPIO que podem assumir varias funções através da devida programação de seus registradores.

Cada um dos pinos pode ser configurado com **pull-up** ou **pull-down** interno e também alta impedância. Possui um pino **ADC**, TOUT ⊥ 9, de resolução de 1024 bits, porém sua alimentação deve ser limitada de 0 a 1V.

2.6 *I2C*

I2C (*Inter-integrated Circuit*) é um protocolo de comunicação *master*/*slave* entre dispositivos que se baseia em um barramento de apenas duas vias: **SDA** (Serial Data) por onde são transmitidos e recebidos os dados e **SCL** (Serial Clock) que dita a temporização do tráfego das informações. O grande diferencial do protocolo é que são permitidos, teoricamente, até 127 dispositivos distintos comunicando através de um mesmo barramento.

O ESP8266 possui suporte a interface *I2C*, tanto como Master quanto Slave, para comunicação com outros microcontroladores e outros equipamentos periféricos, como sensores. A pinagem do barramento é vista pela Tabela 3 abaixo:

Tabela 3 – Pinagem I2C

Pino	Função	Descrição
GPIO02 ⊥ *14*	SDA	Pino de dados
MTMS ⊥ *9*	SCL	Pino de Clock

2.7 UART

O *ESP8266* possui **duas** interfaces *UART* (Universal Asynchronous Receiver/Transmitter) que podem alcançar **4.5 Mbps** de velocidade de transferência: a **UART0**, que pode ser usada para comunicação bidirecional; e a **UART1** que apenas envia dados, por seu pino *TX* (geralmente usado como *debugger*).

Tabela 4 – Pinagem UART

Pino	Função	Descrição
GPIO1 ⊥ 26	TX0	Transmissão Serial
GPIO3 ⊥ 25	RX0	Recepção Serial
GPIO2 ⊥ 23	TX1	Transmissão Serial

2.8 Watchdog

Watchdog timer é um circuito contador independente do clock principal e que serve como um recurso de segurança adicional.

Uma das maiores diferenças entre o *ESP8266* e um MCU utilizado em alguns modelos *Arduino*, é o fato de que o microcontrolador da *Espressif* executa varias tarefas em segundo plano: mantém a conexão do modem Wi-Fi, gerencia a pilha do protocolo TCP/IP, etc. Para manter a execução destas tarefas, ele possui dois watchdog timers: **SW_WDT**, que é via software; e **HW_WDT**, via Hardware. Caso uma função gere um loop longo (mais de 3 segundos) onde não haja alimentação do watchdog, o microcontrolador irá reiniciar, justamente para manter a execução das tarefas de segundo plano.

Consulte a subseção 3.3.3 para melhor entendimento de como o software controla o *timer* do *Watchdog*. Este tópico tratará da função *yield()*, que realiza a alimentação do **SW_WDT**.

2.9 PWM

PWM (ou *Modulação por Largura de Pulso*) se refere a um tipo de sinal digital que permite a variação do tempo, de forma analógica, em que um sinal digital assume nível lógico alto. Desta forma, é possível modular a largura do pulso (duração do nível alto) e gerar sinais de tensões intermediárias.

Na Prática...

Para usarmos um pino como saída de PWM:

```
analogWrite(pino, valor);
```

Alterar a frequência do sinal:

```
analogWriteFreq(novaFrequencia); //100 Hz and 1
    kHz
```

Alterar o *range* do sinal:

```
analogWriteRange(novoRange); // 1024 por padr o
    -> 2^10
```

Você sabia?

Uma clássica experiência do modelo pode ser realizada ao chavear, rapidamente, o interruptor da iluminação do seu quarto, por exemplo. A sensação será a de uma iluminação intermediária entre lâmpada desligada e normalmente acesa.

O ESP8266 possui 4 pinos de interface *PWM*, mas podem ser estendidos pelo usuário. A definição padrão dos pinos é dada pela Tabela 5 abaixo.

Tabela 5 – Pinagem PWM

Pino	**Função**
MTDI ⊥ *10*	PWM0
MTDO ⊥ *13*	PWM1
MTMS ⊥ *9*	PWM2
GPIO4 ⊥ *16*	PWM4

2.10 Interrupção externa

Interrupções são eventos ou condições que levam o microcontrolador a pausar a execução de uma tarefa em andamento, executar outra temporariamente e, então, retornar para a tarefa inicial.

Com exceção do pino *GPIO16* $\perp$ *8*, todas as demais GPIOs do ESP8266 possuem funcionalidade de interrupção externa.

Na Prática...

Podemos utilizar uma interrupção que será deflagrada ao estado do sinal do pino apresentar uma curva de subida - *rising* - e chamará a função *resolveInterrupcao()* da seguinte forma:

```
void setup() {
    ...
    pinMode(pinoDeInterrupcao, INPUT);
    attachInterrupt(pinoDeInterrupcao,
        resolveInterrupcao, "RISING");
    // Alem de "RISING", pode assumir "CHANGE" ou
        "FALLING"
    ...
}

void resolveInterrupcao() {
    ...
}
```

2.11 Consumo de Energia

O ESP8266 já possui modos de consumo definidos, que gerenciam o funcionamento de algumas funcionalidades visando economia energética de acordo com cada tipo de aplicação.

A Tabela 2.11 apresenta estes modos, suas funções ativas e consumo médio.

Tabela 6 – Modos de Consumo Energético

modem-sleep	Modo usado em aplicações que requerem a CPU funcionando, como em aplicações com PWM e I2S. "Desliga"o circuito do Modem Wi-Fi enquanto mantiver uma conexão Wi-Fi sem transmissão de dados. *Consumo médio: 15mA*.
light-sleep	Durante o modo, a CPU pode ser suspendida em aplicações como uma interruptor Wi-Fi. Sem haver transmissão de dados, a o circuito do Modem Wi-Fi pode ser desligado e a CPU suspendida para economia de consumo de energia. *Consumo médio: 0.9mA*.
deep-light	Durante o modo, o modem Wi-Fi é totalmente desligado. Para aplicações onde existam longos intervalos de tempo sem transmissão de dados. Por exemplo, o monitoramento da temperatura ambiente, lendo dados durante um período, dormindo por outro período e acordando para reconectar a um ponto de acesso. *Consumo médio: 20μA*.

CAPÍTULO 3

Firmware

O FIRMWARE traz, por padrão, o conjunto de instruções AT (PDF), porém é possível realizar o upload de outros firmwares para programação em linguagens mais comuns como o *nodeMCU* (LUA) e o *micropython* (Python). Ambos os firmwares baseam em um sistema interno de arquivos, com um arquivo "main"executado no boot e também possuem sua própria biblioteca que é atualizada por suas comunidades. Entretanto, a Arduino IDE possui, atualmente, suporte ao ESP8266, tornando possível a programação em Arduino (C++), utilizando, inclusive, algumas bibliotecas do mesmo sem realizar quaisquer alterações.

3.1 Conversores *USB* para *UART*

O ESP8266 não possui, nativamente, interface para comunicação USB, porém existem conversores que facilitam a comunicação entre o microcontrolador e um PC para realização do upload do programa, por exemplo.

É possível ainda utilizar de dispositivos que já possuam um conversor interno para realizar o upload, como o Arduino UNO. Interligando os pinos de UART dos dois dispositivos (TX(ESP) → RX(Arduino) e RX(ESP) → TX(Arduino)) é possível realizar a comunicação do *ESP8266* com um PC através do conversor USB do Arduino.

ATENÇÃO!

A faixa de tensão de operação do ESP8266 é de 2.5 a 3.6V, sendo necessário, então, um divisor de tensão do *TX* (Arduino), que possui 5V, para o *RX* (ESP8266).

3.2 *IDE Arduino*

Implementada em *Java*, a *IDE Arduino* possui uma interface **simples** e **objetiva**.

Apesar de não apresentar algumas funções e macros presentes em outros editores de texto, como *auto-complete* e snippets, a *IDE* possui recursos interessantes ao usúario. como o **Monitor Serial**.

A figura 4 exibe um *screenshot* do layout do programa.

3.2.1 Monitor Serial

É uma interface de interação com o usuário e depuração do código compilado fornecida pela *IDE Arduino*, onde é possível visualizar dados enviados, via comunicação serial, do MCU para um PC. É possível usá-lo, também, para envio de dados em tempo real do

computador para o microcontrolador. Todos os projetos exemplificados neste livro utilizam o **Monitor Serial** para identificarmos as etapas de execução do programa no *ESP8266*.

Na Prática...

Antes de utilizarmos o **Monitor Serial**, devemos nos certificar de ter inicializado a comunicação serial com seu *baudrate* (tava de transferência de dados), via software:

```
void setup(){
        ...
    Serial.begin(115200);
        ...
}
```

Para escrevermos alguma mensagem no monitor, usamos a função *print()*:

```
Serial.print("MENSAGEM PARA MONITOR");
```

3.3 A Linguagem *Arduino*

Baseada em **C++**, a linguagem de domínio específico *Arduino* traz facilidades ao programador de microcontroladores com funções de leitura, escrita e depuração de pinos de I/O implementadas. Além das funções estruturais *loop()* e *setup()*, que guiam o desenvolvimento do software.

A DSL Arduino possui suporte à orientação a abjetos, alocação dinâmica e manipulação de ponteiros, bem como os tipos de dados aceitos por sua linguagem "mãe".

3.3.1 Funções Estruturais

Um código em Arduino possui duas funções essenciais: a *loop()* e a setup():

Tabela 7 – Funções básicas

void setup()	Função executada apenas uma vez após o MCU dar o boot ou reset. Nela são inicializadas variáveis e indicadas classes, setados modos de entrada ou saída dos pinos, etc.
void loop()	Função chamada após executar a void *setup()*, que rege a execução do código, repetindo infinitas vezes seu conteúdo. Ao final de um ciclo de execução da função, é chamada a função *yield()* para dar feed ao *SW WDT*.

```
void setup() {
    ...
}

void loop() {
    ...
}
```

Figura 4 – IDE Arduino

3.3.2 Timer: *delay() VS millis()*

A linguagem possui duas funções principais para contagem de tempo durante execução do programa:

Tabela 8 – Funções de contagem de tempo

delay(*tempo*)	Função que recebe em *tempo* um valor, em millisegundos, em que o MCU irá pausar a maioria das tasks e retornar a execução normal após este período. A função, porém não desabilita interrupções e mantém os valores da porta serial (recebidos em *RX*), PWM e estados dos pinos. *A função delay() chama, internamente, a função* **yield()**.
millis()	Função retorna o tempo, em milisegundos, desde o início do programa atual. O valor retornado tem tipo *unsigned long* e sofrerá overflow em cerca de 50 dias de execução contínua do MCU. *A função millis()* ***não*** *chama, internamente, a função* **yield()**.

3.3.3 Função *yield()*

Uma das situações mais comuns aos novos usuários do ESP8266, é o reset inesperado - *e misterioso* - do MCU. Como visto anteriormente, o 8266 possui 2 circuitos de *Watchdog* sendo um via software (SW_WDT) e outro via Hardware (HW_WDT). O SW_ WDT é alimentado justamente pela função *yield()*, que faz a contagem de seu timer (cerca de 3 segundos) reiniciar.

Na Prática...

No trecho de código abaixo podemos analisar o uso do *yield()* para evitar o reset do microcontrolador:

```
void loop() {
    ...
    piscaLed(led,10000); // recebe o tempo em
        milissegundos
    ...
}

int piscaLed(int led,int tempo) {
    int tempoInicial = millis();
    int contador = tempoInicial;
    while (contador - tempoInicial < tempo) {
        digitalWrite(led, HIGH);
        delayMicroseconds(500000);
        digitalWrite(led, LOW);
        delayMicroseconds(500000);
        contador = millis();
        yield();
    }
}
```

Como a função *piscaLed()* possui um loop interno que dura, no caso, 10 segundos, caso comentássemos a chamada da *yield()*, o *SW_WDT* não seria alimentado e, em cerca de 3 segundos, o microcontrolador se reiniciaria. Podemos, analogamente, utilizar as seguintes funções que chamam internamente a *yield()* para evitar o *reset*:

```
delay();
```

ou

```
ESP.wdtFeed(); // Esta, porem, alimenta tambem o
    HW_WDT
```

ATENÇÃO!

As funções *delayMicroseconds()* e *millis()* **não** chamam, internamente, a função *yield()*.

3.4 SPIFFS

Um dos problemas enfrentados pelos programadores de microcontroladores é a limitação física da memória, de forma mais crítica, na RAM. Pensando nisso, o *ESP8266* traz, nativamente, um sistema interno de arquivos baseado em um sistema *SPI NOR Flash*. Não suporta criação e manipulação de diretórios, porém é possível nomear um arquivo utilizando "/", tornando possível, ao menos, fantasiara um diretório.

Na Prática...

Inicia o SPIFFS

```
SPIFFS.begin();
```

Abre um arquivo

```
SPIFFS.open(path, mode); // mode recebe "r", "w",
     "a", "r+", "w+" ou "a+".
```

Remove um arquivo

```
SPIFFS.remove(path)
```

Fecha um arquivo

```
file.close()
```

3.5 PROGMEM

Ainda pesando na limitação de hardware da memória, o ESP importa da biblioteca do Arduino uma função que possibilita a alocação de strings e variáveis na memória flash (que é bem maior) ao invés da RAM: a **PROGMEM**.

Na Prática...

Existem duas formas de declarar uma variável para ser armazenada em **flash**:

```
const tipoDado nomeVariavel[] PROGMEM = {};
```

ou

```
const PROGMEM tipoDado nomeVariavel[] = {};
```

Em ambas, após a alocação na memória, para acessar o seu conteúdo são necessários funções especiais de leitura. Por exemplo, para recuperar o valor da variável *mensagem* declarada como:

```
const char mensagem[] PROGMEM = {"ESTE LIVRO EH
     UMA INTRODUCAO AO IOT"};
```

devemos fazê-la da seguinte forma:

```
char mensagemAux;
for (k = 0; k < strlen_P(mensagem); k++)
 {
   mensagemAux =  pgm_read_byte_near(mensagem + k
        );
   Serial.print(mensagemAux);
 }
```

Ao utilizar o **Monitor Serial** como debugger ou uma interface de numerosas mensagens, estamos propícios a, facilmente, consumir muita memória RAM durante as chamadas funções de impressão como:

```
    Serial.print("ESTE LIVRO EH UMA INTRODUCAO AO IOT");
```

Nestas chamadas, todas os dados são armazenados, por padrão, na RAM do dispositivo. Para contornar isso, podemos modificá-las adicionando a macro *F()*, que altera este armazenamento para a flash:

```
    Serial.print(F("ESTE LIVRO EH UMA INTRODUCAO AO IOT")
        );
```

ATENÇÃO!

Todas as variáveis devem ser definidas globalmente ou definidas como estáticas, **caso contrário** o **PROGMEM** não funcionará.

CAPÍTULO 4

Exemplos e Aplicações

TODAS implementações do capítulo utilizam a linguagem *Arduino* e a *IDE Arduino* como framework para desenvolvimento no ESP8266. Foram selecionados exemplos básicos, como o clássico **Blink**; que utilizem o acesso à Internet para obtenção de dados, como o relógio usando **NTP**; de comunicação entre dispositivos sem fio através de protocolos **TCP** e **MQTT**; leitura e processamento de dados de **Sensores**; um **Web Server** e, por fim, exemplos de construção de um **Datalogger** usando uma nuvem ou a própria memória flash do dispositivo.

4.1 *Blink*

```
#define LED 5

void setup() {
    pinMode(LED, OUTPUT);
}

void loop() {
    digitalWrite(LED,HIGH);
    delay(1000);
    digitalWrite(LED,LOW);
    delay(1000);
}
```

4.2 Relógio - *NTP*

```
#include <NTPClient.h>
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

/*
CONFIGURACAO WIFI
*/
const char *ssid     = "ssid_rede";
const char *password = "senha_rede";

WiFiUDP ntpUDP;

//UTC -3:00 Brasil
int16_t utc = -3;
uint32_t atualMillis = 0;
uint32_t anteriorMillis = 0;

NTPClient timeClient(ntpUDP, "a.st1.ntp.br", utc*3600,
     60000);

void setup(){
  Serial.begin(115200);

  WiFi.begin(ssid, password);

  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
    Serial.print ( "." );
  }

  timeClient.begin();
  timeClient.update();
}

void forcaUpdate(void) {
  timeClient.forceUpdate();
}
```

```
void checkOST(void) {
  atualMillis = millis();//Tempo atual em ms
  //Logica de verificacao do tempo
  if (atualMillis - anteriorMillis > 1000) {
    anteriorMillis = atualMillis;    // Salva o tempo
        atual
    Serial.println(timeClient.getFormattedTime());
  }
}

void loop() {
  //Chama a verificacao de tempo
  checkOST();
}
```

4.3 Comunicação entre *ESP8266* e *Arduino* via *I2C*

4.3.1 *Master - ESP8266*

```
#include <Wire.h>


void setup() {
  Wire.begin();
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  Wire.beginTransmission(2);
  Wire.write("Oi, Arduino");
  Wire.endTransmission();

  int data_available = Wire.requestFrom(2, 1);
  if(data_available) {
    String slaveResp = Wire.read();
    if(slaveResp=="Oi, ESP"){
      digitalWrite(LED_BUILTIN, HIGH);
      Wire.write(led_status);
    } else{
      digitalWrite(LED_BUILTIN, LOW);
    }
    Serial.println(slaveResp);
  }
  delay(100);
}
```

4.3.2 *Slave - Arduino*

```
#include <Wire.h>


String rec\_value = "";

void setup() {
  Wire.begin(2);
  Wire.onReceive(receiveCallback);
  Wire.onRequest(requestCallback);
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(115200);
}


void loop() {
}

void receiveCallback(int bytes)
{
  if(Wire.available() != 0)
  {
    for(int i = 0; i< bytes; i++)
    {
      rec_value = Wire.read();
      Serial.print("Recebido: ");
      Serial.println(rec_value);
    }
    if(rec_value=="Oi, Arduino"){
      digitalWrite(LED_BUILTIN, LOW);
    } else {
      digitalWrite(LED_BUILTIN, HIGH);
    }
  }
}

void requestCallback(int bytes) {
```

```
  Wire.write("Oi, Esp");
}
```

4.4 Controle remoto de aparelhos via *MQTT*

```
#include <IRremoteESP8266.h>
#include <IRsend.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Valores lidos pelo controle remoto
#define SamsungPower        0xE0E040BF
#define SamsungSource        0xE0E0807F
#define SamsungExit        0xE0E0B44B
#define SamsungUp        0xE0E006F9
#define SamsungDown        0xE0E08679
#define SamsungRight        0xE0E046B9
#define SamsungLeft        0xE0E0A659
#define SamsungSelect        0xE0E016E9
#define SamsungUpVol        0xE0E0E01F
#define SamsungDownVol        0xE0E0D02F
#define SamsungUpCh        0xE0E048B7
#define SamsungDownCh        0xE0E008F7
#define SamsungMute        0xE0E0F00F

// Inicia o pino 12 para saida de IR
IRsend irsend(12);

/*
 CONFIGURACAO WIFI
*/
const char* ssid = "ssid_rede";
const char* password =  "senha_rede";
// Endereco do servidor mqtt
const char* mqttServer = "m13.cloudmqtt.com";
// Porta do Broker
const int mqttPort = 11111;
// Nome de usuario fornecido pelo broker
const char* mqttUser = "userMqtt";
const char* mqttPassword = "passMqtt";
```

```
WiFiClient espClient;
PubSubClient client(espClient);

unsigned long initTime;
unsigned long posTime;

void mqtt_callback(char* topic, byte* payload, unsigned
     int length);

void setup() {
  irsend.begin();
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }

  client.setServer(mqttServer, mqttPort);
  client.setCallback(callback);

  // publica("t pico", "mensagem")
  client.publish("/node03","conecao_estabelecida");
  // Inscreve em um topico
  client.subscribe("/node03/tv");

}

void callback(char* topic, byte* payload, unsigned int
     length) {

    String mensagem;
    //obtem a string do payload recebido
    for(int i = 0; i < length; i++)
    {
       char c = (char)payload[i];
       mensagem += c;
    }

    if (mensagem.equals("power"))
    {
```

```
      initTime = millis();
      posTime = initTime;
      while(posTime<= initTime+50){
       irsend.sendSAMSUNG(SamsungPower, SAMSUNG_BITS, 0);
              // hex value, 16 bits, no repeat
       irsend.sendSAMSUNG(SamsungPower, SAMSUNG_BITS, 1);
              // hex value, 16 bits, repeat
       posTime = millis();
      }
    }

    else if (mensagem.equals("source"))
    {
      initTime = millis();
      posTime = initTime;
      while(posTime<= initTime+50){
       irsend.sendSAMSUNG(SamsungSource, SAMSUNG_BITS, 0)
            ;  // hex value, 16 bits, no repeat
       irsend.sendSAMSUNG(SamsungSource, SAMSUNG_BITS, 1)
            ;  // hex value, 16 bits, repeat
       posTime = millis();
      }
    }

    else if (mensagem.equals("select"))
    {
       irsend.sendSAMSUNG(SamsungSelect, SAMSUNG_BITS, 0)
            ;  // hex value, 16 bits, no repeat
    }

  else if (mensagem.equals("exit"))
    {
       irsend.sendSAMSUNG(SamsungExit, SAMSUNG_BITS, 0);
            // hex value, 16 bits, no repeat
    }

    else if (mensagem.equals("up"))
    {
       irsend.sendSAMSUNG(SamsungUp, SAMSUNG_BITS, 0);
            // hex value, 16 bits, no repeat
```

```
    }

    else if (mensagem.equals("down"))
    {
       irsend.sendSAMSUNG(SamsungDown, SAMSUNG_BITS, 0);
             // hex value, 16 bits, no repeat
    }

    else if (mensagem.equals("right"))
    {
      irsend.sendSAMSUNG(SamsungRight, SAMSUNG_BITS, 0);
            // hex value, 16 bits, no repeat
    }

    else if (mensagem.equals("left"))
    {
      irsend.sendSAMSUNG(SamsungLeft, SAMSUNG_BITS, 0);
           // hex value, 16 bits, no repeat
    }

    else if (mensagem.equals("volup"))
    {
      irsend.sendSAMSUNG(SamsungUpVol, SAMSUNG_BITS, 0);
            // hex value, 16 bits, no repeat
    }

    else if (mensagem.equals("voldown"))
    {
       irsend.sendSAMSUNG(SamsungDownVol, SAMSUNG_BITS,
            0);  // hex value, 16 bits, no repeat
    }

    else if (mensagem.equals("chup"))
    {
       irsend.sendSAMSUNG(SamsungUpCh, SAMSUNG_BITS, 0);
             // hex value, 16 bits, no repeat
    }

    else if (mensagem.equals("chdown"))
    {
```

```
        irsend.sendSAMSUNG(SamsungDownCh, SAMSUNG_BITS, 0)
             ;  // hex value, 16 bits, no repeat
    }

    else if (mensagem.equals("mute"))
    {
        irsend.sendSAMSUNG(SamsungMute, SAMSUNG_BITS, 0);
              // hex value, 16 bits, no repeat
    }
}

void loop() {
  client.loop();
}
```

4.5 *Web Server* para envio de mensagens para um *channel* do *Slack*

```
#include <ESP8266WiFi.h>

/*
 CONFIGURACAO WIFI
 */
const char* ssid = "ssid_rede";
const char* password = "senha_rede";
WiFiServer server(80);

/*
 CONFIGURACAO SLACK
 */
// url do webhooks gerado no slack
const String slack_hook_url = "https://hooks.slack.
    com/services/******/******/******";
// url do icone usado pelo usuario criado pelo hooks
const String slack_icon_url = "https://pbs.twimg.com/
    profile_images/1462227900/
    cda288d94c3e99d0ccc4e8d1c61d7073_normal.jpg";
const String slack_message = "#GoEMakers - mensagem
    enviada pelo ESP8266 by Arduino Firmware";
const String slack_username = "meu_esp";

void setup()
{
  Serial.begin(115200);

  WiFi.begin(ssid, password);

  Serial.print("Conectando a ");
  Serial.print(ssid);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
```

```
    }
    Serial.println();

    Serial.println("Connectado!");
    Serial.print("Endereco IP: ");
    Serial.println(WiFi.localIP());

    server.begin();
  }

  void consultaServidor(){
    WiFiClient client = server.available();
    if (!client) {
      return;
    }
    String leitura = client.readStringUntil('\r');
    Serial.println(leitura);
    client.flush();

    String buf = "";

    buf += "HTTP/1.1 200 OK\r\nContent-Type: text/html\
        r\n\r\n<!DOCTYPE HTML>\r\n";
    buf += "<html lang=\"en\"><head><meta charset=\"UTF
        -8\" name=\"viewport\" content=\"width=device-
        width, initial-scale=1, user-scalable=no\"/>\r\
        n";
    buf += "<title>EMakers - Curso ESP8266</title>";
    buf += "<style>.c{text-align: center;} .titulo{
        color: white;} div,input{padding:5px;font-size
        :1em;background-color:#1fa3ec;} input{width
        :30\%;} body{text-align: center;font-family:
        verdana;} button{border:0;border-radius:0.3rem;
        background-color:#1fa3ec;color:#fff;line-height
        :2.4rem;font-size:1.2rem;width:20\%;} </style>"
        ;
    buf += "</head>";
    buf += "<script> function mudaCor(el){el.style.
        backgroundColor = '#000000';}</script>";
    buf += "<div><h1 class=\"titulo\">EMakers - Curso
```

```
        ESP8266</h1></div>";
    buf += "<h3>Bot o de envio de mensagens Slack</h3>
        ";
    buf += "<p>Mensagem_1: <a href=\"?function=
        mensagem1\"><button onclick=\"mudaCor(this)\">
        ENVIAR</button></a></p>";
    buf += "<p>Mensagem_2: <a href=\"?function=
        mensagem2\"><button onclick=\"mudaCor(this)\">
        ENVIAR</button></a></p>";
    buf += "<div class=\"titulo\"> <p>Desenvolvido por
        Gabriel Melo</p>";
    buf += "</html>\n";

    client.print(buf);
    client.flush();

    if (leitura.indexOf("mensagem1") != -1){
      mensagemParaSlack("oi, slack");
      Serial.println("Mensagem Enviada!");
    }
    else if (leitura.indexOf("mensagem2") != -1){
      mensagemParaSlack("tchau, slack");
      Serial.println("Mensagem Enviada!");
    }
    else {
      Serial.println("Requisicao invalida");
      client.stop();
    }
  }

  bool mensagemParaSlack(String msg)
  {
    const char* host = "hooks.slack.com";
    Serial.print("Conectando a  ");
    Serial.println(host);

    // Criar conexao TCP
    WiFiClientSecure client;
    const int httpsPort = 443;
    if (!client.connect(host, httpsPort)) {
```

```
      Serial.println("Falha na conexao :-(");
      return false;
    }

    //Criacao da URI para requisicao

    Serial.print("Postando em: ");
    Serial.println(slack_hook_url);

    String postData="payload={\"link_names\": 1, \"
        icon_url\": \"" + slack_icon_url + "\", \"
        username\": \"" + slack_username + "\", \"text
        \": \"" + msg + "\"}";

    //Envio da requisicao para o servidor
    client.print(String("POST ") + slack_hook_url + "
        HTTP/1.1\r\n" +
                 "Host: " + host + "\r\n" +
                 "Content-Type: application/x-www-form-
                     urlencoded\r\n" +
                 "Connection: close" + "\r\n" +
                 "Content-Length:" + postData.length()
                     + "\r\n" +
                 "\r\n" + postData);
    Serial.println("Requisicao enviada");
    String line = client.readStringUntil('\n');
    Serial.printf("Resposta: ");
    Serial.println(line);
    if (line.startsWith("HTTP/1.1 200 OK")) {
      return true;
    } else {
      return false;
    }
  }

  void loop(){
    consultaServidor();
  }
```

4.6 Controle de ponto usando RFID

```
#include <ESP8266WiFi.h>
#include <MFRC522.h>
#include <SPI.h>

#define SS_PIN 4
#define RST_PIN 5
MFRC522 mfrc522(SS_PIN, RST_PIN);

String msg = "";
char st[20];

/*
CONFIGURACAO WIFI
*/
const char *ssid     = "ssid_rede";
const char *password = "senha_rede";

/*
 CONFIGURACAO SLACK
 */
// url do webhooks gerado no slack
const String slack_hook_url = "https://hooks.slack.
     com/services/******/******/******";
// url do icone usado pelo usuario criado pelo hooks
const String slack_icon_url = "https://pbs.twimg.com/
     profile_images/1462227900/
     cda288d94c3e99d0ccc4e8d1c61d7073_normal.jpg";
const String slack_message = "#GoEMakers - mensagem
     enviada pelo ESP8266 by Arduino Firmware";
const String slack_username = "meu_esp";

void setup()
{
  Serial.begin(9600); // Inicia a serial
  Serial.println();
    WiFi.begin(ssid, password);
```

```
    Serial.print("Connecting");
    while (WiFi.status() != WL_CONNECTED)
    {
      delay(500);
      Serial.print(".");
    }
    Serial.println();

    Serial.print("Connected, IP address: ");
    Serial.println(WiFi.localIP());

    SPI.begin();    // Inicia  SPI bus
    mfrc522.PCD_Init(); // Inicia MFRC522
    Serial.println("Aproxime o seu cartao do leitor..."
        );

}


bool mensagemParaSlack(String msg) {
  const char* host = "hooks.slack.com";
  Serial.print("Conectando a  ");
  Serial.println(host);

  // Criar conexao TCP
  WiFiClientSecure client;
  const int httpsPort = 443;
  if (!client.connect(host, httpsPort)) {
    Serial.println("Falha na conexao :-(");
    return false;
  }

  //Criacao da URI para requisicao

  Serial.print("Postando em: ");
  Serial.println(slack_hook_url);

  String postData="payload={\"link_names\": 1, \"
      icon_url\": \"" + slack_icon_url + "\", \"
      username\": \"" + slack_username + "\", \"text
```

```
          \": \"" + msg + "\"}";

    //Envio da requisicao para o servidor
    client.print(String("POST ") + slack_hook_url + "
        HTTP/1.1\r\n" +
                "Host: " + host + "\r\n" +
                "Content-Type: application/x-www-form-
                    urlencoded\r\n" +
                "Connection: close" + "\r\n" +
                "Content-Length:" + postData.length()
                    + "\r\n" +
                "\r\n" + postData);
    Serial.println("Requisicao enviada");
    String line = client.readStringUntil('\n');
    Serial.printf("Resposta: ");
    Serial.println(line);
    if (line.startsWith("HTTP/1.1 200 OK")) {
      return true;
    } else {
      return false;
    }
  }

  void loop()
  {
    //Procura novos cards
    if ( ! mfrc522.PICC_IsNewCardPresent())
    {
      return;
    }
    //Seleciona um card
    if ( ! mfrc522.PICC_ReadCardSerial())
    {
      return;
    }
    //Mostra UID na serial
    Serial.print("UID da tag :");
    String conteudo= "";
    byte letra;
    for (byte i = 0; i < mfrc522.uid.size; i++)
```

```
    {
       Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0
            " : " ");
       Serial.print(mfrc522.uid.uidByte[i], HEX);
       conteudo.concat(String(mfrc522.uid.uidByte[i] <
            0x10 ? " 0" : " "));
       conteudo.concat(String(mfrc522.uid.uidByte[i],
            HEX));
    }
    Serial.println();
    Serial.print("Mensagem : ");
    conteudo.toUpperCase();

    if (conteudo.substring(1) == "64 96 35 24") //UID 2
         - Cartao
    {
      msg = "Gabriel Marques de Melo - Chegou na sala!"
           ;
      mensagemParaSlack(msg);
      Serial.println(msg);
      Serial.println();
      delay(2000);
    }
  }
```

4.7 Monitor de temperatura ambiente com *DHT11*

```
#include "DHT.h"

#define PINO_DHT 4
#define TIPO_DHT DHT11

DHT dht(PINO_DHT, TIPO_DHT);

void setup() {
    Serial.begin(115200);
    dht.begin();

}

void tempUmi() {
  float u = dht.readHumidity();
  float t = dht.readTemperature();
  Serial.print("Temperatura: ");
  Serial.println(String(t));
  Serial.print("Umidade: ");
  Serial.println(String(u));

}

void loop() {
    tempUmi();
    delay(500);
}
```

4.8 Comunicação *TCP* entre ESP8266

4.9 *Datalogger* com *SPIFFS*

4.10 *Datalogger* com *Google Sheets*


EMAKERS