



## Índice

<a href="#">Introdução .....</a>	24
<a href="#">Visão geral .....</a>	25
<a href="#">O ESP8266 .....</a>	26
<a href="#">Maturidade .....</a>	27
<a href="#">A especificação ESP8266 .....</a>	27
<a href="#">Módulos ESP8266 .....</a>	28
<a href="#">ESP-12 .....</a>	28
<a href="#">ESP-1 .....</a>	32
<a href="#">Adafruit HUZZAH .....</a>	38
<a href="#">NodeMCU devKit .....</a>	38
<a href="#">node.IT (também conhecido como ESP-210) .....</a>	40
<a href="#">SparkFun WiFi Shield - ESP8266 .....</a>	40
<a href="#">Espresso Lite .....</a>	41
<a href="#">Wemos D1 .....</a>	41
<a href="#">Oak por digistump .....</a>	41
<a href="#">Conectando ao ESP8266 .....</a>	41
<a href="#">Teoria WiFi .....</a>	42
<a href="#">Programação do Comando AT .....</a>	44
<a href="#">Comandos .....</a>	45
<a href="#">Instalando o processador de comando AT mais recente .....</a>	51
<a href="#">Montando circuitos .....</a>	52
<a href="#">Conversores USB para UART .....</a>	52
<a href="#">Pranchas de ensaio .....</a>	54
<a href="#">Power .....</a>	55
<a href="#">Multímetro / Sonda lógica / Analisador lógico .....</a>	56
<a href="#">Componentes diversos .....</a>	56
<a href="#">Construção física .....</a>	56
<a href="#">Configuração recomendada para programação ESP8266 .....</a>	56
<a href="#">Configuração para atualizar o dispositivo .....</a>	59
<a href="#">Programação .....</a>	60
<a href="#">Modo de inicialização .....</a>	60
<a href="#">ESP8266 - Kit de Desenvolvimento de Software (SDK) .....</a>	61
<a href="#">Incluir diretórios .....</a>	61
<a href="#">Compilando .....</a>	62
<a href="#">Carregando um programa no ESP8266 .....</a>	69
<a href="#">Ambientes de programação .....</a>	73
<a href="#">Ferramentas de compilação .....</a>	73
<a href="#">ar .....</a>	73
<a href="#">esptool.py .....</a>	74
<a href="#">esptool-ck .....</a>	76

<a href="#">gcc .....</a>	78
<a href="#">gen_appbin.py .....</a>	79
<a href="#">make .....</a>	80
<a href="#">nodemcu-pisca-pisca .....</a>	80
<a href="#">nm .....</a>	82
<a href="#">objcopy .....</a>	82

<u>objdump .....</u>	82
<u>ESP8266 Linking .....</u>	83
<u>Piscando no ar - FOTA .....</u>	84
<u>Depuração .....</u>	88
<u>Registro ESP-IDF .....</u>	88
<u>Registrando no UART1 .....</u>	90
<u>Execute um Blinky .....</u>	90
<u>Despejando endereços IP .....</u>	91
<u>Tratamento de exceções .....</u>	91
<u>Depurando e testando conexões TCP e UDP .....</u>	93
<u>Android - protocolo de soquete .....</u>	94
<u>Android - UDP Sender / Receiver .....</u>	94
<u>Windows - Hercules .....</u>	94
<u>Curl .....</u>	94
<u>Eclipse - TCP / MON .....</u>	94
<u>httpbin.org .....</u>	97
<u>Arquitetura ESP8266 .....</u>	97
<u>Programas personalizados .....</u>	97
<u>WiFi na inicialização .....</u>	97
<u>Trabalhando com WiFi - ESP8266 .....</u>	98
<u>Procurando pontos de acesso .....</u>	98
<u>Definindo o modo de operação .....</u>	99
<u>Lidando com eventos WiFi .....</u>	99
<u>Configuração da estação .....</u>	101
<u>Conectando a um ponto de acesso .....</u>	101
<u>Controle e fluxos de dados ao conectar como uma estação .....</u>	102
<u>Sendo um ponto de acesso .....</u>	103
<u>O servidor DHCP .....</u>	104
<u>Endereço IP atual, máscara de rede e gateway .....</u>	105
<u>Configuração protegida de WiFi - WPS .....</u>	105
<u>Trabalhando com TCP / IP .....</u>	106
<u>A arquitetura espconn .....</u>	106
<u>TCP .....</u>	107
<u>Enviando e recebendo dados TCP .....</u>	111
<u>Controle de fluxo .....</u>	113
<u>Tratamento de erros TCP .....</u>	113
<u>UDP .....</u>	114

---

## Página 4

<u>Transmitir com UDP .....</u>	116
<u>Solicitação de ping .....</u>	117
<u>Serviço de nomes .....</u>	117
<u>Sistemas de nomes de domínio multicast .....</u>	118
<u>Instalando Bonjour .....</u>	119
<u>Trabalhando com SNTP .....</u>	122
<u>ESP-NOW .....</u>	123
<u>GPIOs .....</u>	124
<u>Configurações de pullup e pull down .....</u>	130
<u>Manipulação de interrupção GPIO .....</u>	0,130
<u>Expanding o número de GPIOs disponíveis .....</u>	132
<u>Biblioteca C ESP_PCF8574 .....</u>	136
<u>Biblioteca JavaScript PCF8574 .....</u>	137
<u>Trabalhando com I2C .....</u>	137
<u>Trabalhando com SPI - Interface Periférica Serial .....</u>	139
<u>Hardware SPI .....</u>	141

<u>O MetalPhreak / ESP8266 SPI Driver .....</u>	143
<u>Trabalhando com serial .....</u>	144
<u>ESP8266 Gerenciamento de tarefas .....</u>	146
<u>Temporizadores e tempo .....</u>	147
<u>Trabalhando com memória .....</u>	148
<u>Trabalhando com memória flash .....</u>	0,152
<u>Modulação por largura de pulso - PWM .....</u>	153
<u>Conversão analógica para digital .....</u>	0,154
<u>Modos de suspensão .....</u>	156
<u>Watchdog timer .....</u>	157
<u>Controle de sessão .....</u>	158
<u>Segurança .....</u>	159
<u>Mapeamento do Arduino .....</u>	159
<u>Sistema de arquivos Spiffs .....</u>	160
<u>APIs TCP / IP de parceiros .....</u>	161
<u>Soquetes TCP / IP .....</u>	162
<u>Manipulando erros .....</u>	165
<u>Sockets - aceitar () .....</u>	168
<u>Sockets - bind () .....</u>	169
<u>Sockets - close () .....</u>	169
<u>Sockets - closesocket () .....</u>	169
<u>Soquetes - conectar () .....</u>	169
<u>Sockets - fcntl () .....</u>	170
<u>Sockets - freeaddrinfo () .....</u>	170
<u>Sockets - getaddrinfo () .....</u>	170
<u>Sockets - gethostname () .....</u>	170
<u>Sockets - getpeername () .....</u>	0,170
<u>Sockets - getsockname () .....</u>	170

---

## Página 5

<u>Sockets - getsockopt () .....</u>	170
<u>Sockets - htonl () .....</u>	171
<u>Sockets - htons () .....</u>	171
<u>Sockets - inet_ntop () .....</u>	171
<u>Sockets - inet_nton () .....</u>	171
<u>Sockets - ioctlsocket () .....</u>	171
<u>Sockets - listen () .....</u>	171
<u>Sockets - read () .....</u>	171
<u>Sockets - recv () .....</u>	172
<u>Sockets - recvfrom () .....</u>	172
<u>Sockets - select () .....</u>	173
<u>Sockets - send () .....</u>	173
<u>Sockets - sendto () .....</u>	173
<u>Sockets - setsockopt () .....</u>	173
<u>Sockets - shutdown () .....</u>	174
<u>Sockets - socket () .....</u>	174
<u>Sockets - write () .....</u>	174
<u>Estruturas de dados de soquete .....</u>	175
<u>Sockets - struct sockaddr .....</u>	175
<u>Sockets - struct sockaddr_in .....</u>	175
<u>Java Sockets .....</u>	175
<u>WebSockets .....</u>	178
<u>Um aplicativo de navegador WebSocket .....</u>	178
<u>FreeRTOS WebSocket .....</u>	0,180
<u>Mongoose WebSocket .....</u>	.180
<u>Servidores Web .....</u>	181

<a href="#">Mongoose .....</a>	181
<a href="#">Programando usando Eclipse .....</a>	182
<a href="#">Instalando o Terminal Serial Eclipse .....</a>	186
<a href="#">Desenvolvimento Web usando Eclipse .....</a>	192
<a href="#">Programando usando o Arduino IDE .....</a>	193
<a href="#">Implicações do suporte IDE Arduino .....</a>	194
<a href="#">Instalando o IDE Arduino com suporte ESP8266 .....</a>	195
<a href="#">Dicas para trabalhar no ambiente Arduino .....</a>	201
<a href="#">Inicilize classes globais em setup () .....</a>	201
<a href="#">Invocando Espressif SDK API de um esboço .....</a>	201
<a href="#">Tratamento de exceções .....</a>	202
<a href="#">O sistema de arquivos SPIFFS .....</a>	202
<a href="#">O comando mkspiffs .....</a>	202
<a href="#">A arquitetura do suporte IDE Arduino .....</a>	203
<a href="#">Construindo aplicativos ESP Arduino usando o IDE Eclipse .....</a>	211
<a href="#">Razões para considerar o uso do Eclipse em vez do Arduino IDE .....</a>	225
<a href="#">Notas sobre o uso do pacote Eclipse Arduino .....</a>	226
<a href="#">Bibliotecas Arduino ESP .....</a>	227

---

## Página 6

<a href="#">A biblioteca WiFi .....</a>	227
<a href="#">WiFi.begin .....</a>	227
<a href="#">WiFi.beingSmartConfig .....</a>	228
<a href="#">WiFi.beginWPSConfig .....</a>	228
<a href="#">WiFi.BSSID .....</a>	228
<a href="#">WiFi.BSSIDstr .....</a>	228
<a href="#">Canal WiFi .....</a>	228
<a href="#">WiFi.config .....</a>	229
<a href="#">WiFi.disconnect .....</a>	229
<a href="#">WiFi.encryptionType .....</a>	229
<a href="#">WiFi.gatewayIP .....</a>	229
<a href="#">WiFi.getNetworkInfo .....</a>	229
<a href="#">WiFi.hostByName .....</a>	230
<a href="#">WiFi.hostname .....</a>	230
<a href="#">WiFi.isHidden .....</a>	230
<a href="#">WiFi.localIP .....</a>	230
<a href="#">WiFi.macAddress .....</a>	230
<a href="#">WiFi.mode .....</a>	231
<a href="#">WiFi.printDiag .....</a>	231
<a href="#">WiFi.RSSI .....</a>	231
<a href="#">WiFi.scanComplete .....</a>	231
<a href="#">WiFi.scanDelete .....</a>	232
<a href="#">WiFi.scanNetworks .....</a>	232
<a href="#">WiFi.smartConfigDone .....</a>	232
<a href="#">WiFi.softAP .....</a>	232
<a href="#">WiFi.softAPConfig .....</a>	233
<a href="#">WiFi.softAPDisconnect .....</a>	233
<a href="#">WiFi.softAPmacAddress .....</a>	233
<a href="#">WiFi.softAPIP .....</a>	233
<a href="#">WiFi.SSID .....</a>	233
<a href="#">WiFi.status .....</a>	233
<a href="#">WiFi.stopSmartConfig .....</a>	234
<a href="#">WiFi.subnetMask .....</a>	234
<a href="#">WiFi.waitForConnectResult .....</a>	234
<a href="#">WiFiClient .....</a>	234
<a href="#">WiFiClient .....</a>	234

<a href="#">WiFiClient.available</a>	.....	234
<a href="#">WiFiClient.connect</a>	.....	235
<a href="#">WiFiClient.connected</a>	.....	235
<a href="#">WiFiClient.flush</a>	.....	235
<a href="#">WiFiClient.getNoDelay</a>	.....	235
<a href="#">WiFiClient.peek</a>	.....	235
<a href="#">WiFiClient.read</a>	.....	235
<a href="#">WiFiClient.remoteIP</a>	.....	235

Página 6

---

**Página 7**

<a href="#">WiFiClient.remotePort</a>	.....	236
<a href="#">WiFiClient.setLocalPortStart</a>	.....	236
<a href="#">WiFiClient.setNoDelay</a>	.....	236
<a href="#">WiFiClient.status</a>	.....	236
<a href="#">WiFiClient.stop</a>	.....	236
<a href="#">WiFiClient.stopAll</a>	.....	236
<a href="#">WiFiClient.write</a>	.....	236
<a href="#">WiFiServer</a>	.....	237
<a href="#">WiFiServer</a>	.....	237
<a href="#">WiFiServer.available</a>	.....	237
<a href="#">WiFiServer.begin</a>	.....	237
<a href="#">WiFiServer.getNoDelay</a>	.....	237
<a href="#">WiFiServer.hasClient</a>	.....	0,237
<a href="#">WiFiServer.setNoDelay</a>	.....	238
<a href="#">WiFiServer.status</a>	.....	238
<a href="#">WiFiServer.write</a>	.....	238
<a href="#">Endereço IP</a>	.....	238
<a href="#">ESP8266WebServer</a>	.....	238
<a href="#">ESP8266WebServer</a>	.....	241
<a href="#">ESP8266WebServer.arg</a>	.....	241
<a href="#">ESP8266WebServer.argName</a>	.....	241
<a href="#">ESP8266WebServer.args</a>	.....	241
<a href="#">ESP8266WebServer.begin</a>	.....	241
<a href="#">ESP8266WebServer.client</a>	.....	241
<a href="#">ESP8266WebServer.handleClient</a>	.....	242
<a href="#">ESP8266WebServer.hasArg</a>	.....	242
<a href="#">ESP8266WebServer.method</a>	.....	242
<a href="#">ESP8266WebServer.on</a>	.....	242
<a href="#">ESP8266WebServer.onFileUpload</a>	.....	243
<a href="#">ESP8266WebServer.onNotFound</a>	.....	243
<a href="#">ESP8266WebServer.send</a>	.....	243
<a href="#">ESP8266WebServer.sendContent</a>	.....	243
<a href="#">ESP8266WebServer.sendHeader</a>	.....	243
<a href="#">ESP8266WebServer.setContentLength</a>	.....	243
<a href="#">ESP8266WebServer.streamFile</a>	.....	244
<a href="#">ESP8266WebServer.upload</a>	.....	244
<a href="#">ESP8266WebServer.uri</a>	.....	244
<a href="#">Biblioteca ESP8266mDNS</a>	.....	0,244
<a href="#">MDNS.addService</a>	.....	244
<a href="#">MDNS.begin</a>	.....	244
<a href="#">MDNS.update</a>	.....	244
<a href="#">I2C - Fio</a>	.....	245
<a href="#">Wire.available</a>	.....	245
<a href="#">Wire.begin</a>	.....	245

---

Página 8

<a href="#">Wire.beginTransmission .....</a>	246
<a href="#">Wire.endTransmission .....</a>	246
<a href="#">Wire.flush ....</a>	246
<a href="#">Wire.onReceive .....</a>	247
<a href="#">Wire.onReceiveService .....</a>	247
<a href="#">Wire.onRequest .....</a>	247
<a href="#">Wire.onRequestService .....</a>	247
<a href="#">Wire.peek .....</a>	247
<a href="#">Wire.pins .....</a>	247
<a href="#">Wire.read .....</a>	248
<a href="#">Wire.requestFrom .....</a>	248
<a href="#">Wire.setClock .....</a>	248
<a href="#">Wire.write .....</a>	249
<a href="#">Biblioteca de ticker .....</a>	249
<a href="#">Ticker ...</a>	249
<a href="#">anexar ...</a>	249
<a href="#">attach_ms ...</a>	250
<a href="#">separar ...</a>	250
<a href="#">uma vez ...</a>	250
<a href="#">once_ms ...</a>	250
<a href="#">Biblioteca EEPROM .....</a>	250
<a href="#">EEPROM.begin .....</a>	251
<a href="#">EEPROM.commit .....</a>	251
<a href="#">EEPROM.end .....</a>	251
<a href="#">EEPROM.get .....</a>	251
<a href="#">EEPROM.getDataPtr .....</a>	0,251
<a href="#">EEPROM.put .....</a>	251
<a href="#">EEPROM.read .....</a>	251
<a href="#">EEPROM.write .....</a>	251
<a href="#">SPIFFS .....</a>	252
<a href="#">SPIFFS.begin .....</a>	252
<a href="#">SPIFFS.open .....</a>	252
<a href="#">SPIFFS.openDir .....</a>	252
<a href="#">SPIFFS.remove .....</a>	252
<a href="#">SPIFFS.rename .....</a>	253
<a href="#">Arquivo.disponível .....</a>	253
<a href="#">File.close ....</a>	253
<a href="#">File.flush ....</a>	253
<a href="#">File.name .....</a>	253
<a href="#">File.peek .....</a>	253
<a href="#">Posição do arquivo .....</a>	253
<a href="#">File.read .....</a>	253
<a href="#">File.seek .....</a>	254
<a href="#">File.size ...</a>	254

---

Página 9

<a href="#">File.write ....</a>	254
---------------------------------	-----

<a href="#">Dir.fileName .....</a>	254
<a href="#">Dir.next .....</a>	254
<a href="#">Dir.open .....</a>	254
<a href="#">Dir.openDir .....</a>	254
<a href="#">Dir.remove .....</a>	255
<a href="#">Dir.rename .....</a>	255
<a href="#"><b>Biblioteca ESP .....</b></a>	<b>255</b>
<a href="#">ESP.deepSleep .....</a>	255
<a href="#">ESP.eraseConfig .....</a>	255
<a href="#">ESP.getBootMode .....</a>	255
<a href="#">ESP.getBootVersion .....</a>	255
<a href="#">ESP.getChipId .....</a>	255
<a href="#">ESP.getCpuFreqMHz .....</a>	255
<a href="#">ESP.getCycleCount .....</a>	255
<a href="#">ESP.getFlashChipId .....</a>	255
<a href="#">ESP.getFlashChipMode .....</a>	255
<a href="#">ESP.getFlashChipRealSize .....</a>	256
<a href="#">ESP.getFlashChipSize .....</a>	256
<a href="#">ESP.getFlashChipSizeByChipId .....</a>	256
<a href="#">ESP.getFlashChipSpeed .....</a>	256
<a href="#">ESP.getFreeHeap .....</a>	256
<a href="#">ESP.getFreeSketchSpace .....</a>	256
<a href="#">ESP.getResetInfo .....</a>	256
<a href="#">ESP.getResetInfoPtr .....</a>	256
<a href="#">ESP.getSdkVersion .....</a>	256
<a href="#">ESP.getSketchSize .....</a>	256
<a href="#">ESP.getVec .....</a>	257
<a href="#">ESP.reset .....</a>	257
<a href="#">ESP.restart .....</a>	257
<a href="#">ESP.updateSketch .....</a>	257
<a href="#">ESP.wdtDisable .....</a>	257
<a href="#">ESP.wdtEnable .....</a>	257
<a href="#">ESP.wdtFeed .....</a>	257
<a href="#"><b>Biblioteca de strings .....</b></a>	<b>258</b>
<a href="#">Construtor .....</a>	258
<a href="#">String.c_str .....</a>	258
<a href="#">String.reserve .....</a>	258
<a href="#">String.length .....</a>	258
<a href="#">String.concat .....</a>	258
<a href="#">String.equalsIgnoreCase .....</a>	258
<a href="#">String.startsWith .....</a>	258
<a href="#">String.endsWith .....</a>	259
<a href="#">String.charAt .....</a>	259

<a href="#">String.setCharAt .....</a>	259
<a href="#">String.getBytes .....</a>	259
<a href="#">String.toCharArray .....</a>	259
<a href="#">String.indexOf .....</a>	259
<a href="#">String.lastIndexOf .....</a>	259
<a href="#">String.substring .....</a>	259
<a href="#">String.replace .....</a>	259
<a href="#">String.remove .....</a>	259
<a href="#">String.toLowerCase .....</a>	259
<a href="#">String.toUpperCase .....</a>	259
<a href="#">String.trim .....</a>	260

<a href="#">String.toInt .....</a>	260
<a href="#">String.toFloat .....</a>	260
<a href="#">Programação com JavaScript .....</a>	0,260
<a href="#">Smart.js .....</a>	261
<a href="#">Smart.js GPIO .....</a>	262
<a href="#">Configurando um servidor HTTP .....</a>	263
<a href="#">Depuração .....</a>	264
<a href="#">Espruino .....</a>	265
<a href="#">Editando e implantando código .....</a>	265
<a href="#">Trabalhando com variáveis .....</a>	266
<a href="#">Inicializando o Espruino .....</a>	266
<a href="#">Acesso WiFi .....</a>	266
<a href="#">Escrevendo aplicativos de soquete de rede usando Espruino .....</a>	267
<a href="#">Escrevendo um cliente REST usando Espruino .....</a>	268
<a href="#">Escrevendo um servidor Web usando Espruino .....</a>	269
<a href="#">Trabalhando com GPIO .....</a>	271
<a href="#">Trabalhando com I2C e JavaScript .....</a>	271
<a href="#">Depurando JavaScript .....</a>	272
<a href="#">Editando JavaScript .....</a>	272
<a href="#">Bibliotecas Espruino ESP8266 .....</a>	273
<a href="#">Recursos principais de JavaScript .....</a>	274
<a href="#">Executando código em intervalos .....</a>	274
<a href="#">Trabalhando com GPIO .....</a>	275
<a href="#">SPI .....</a>	275
<a href="#">Principais diferenças do JavaScript .....</a>	276
<a href="#">Construindo Espruino .....</a>	276
<a href="#">Programando com Lua .....</a>	277
<a href="#">ESPlorer IDE .....</a>	277
<a href="#">GPIO com Lua .....</a>	277
<a href="#">WiFi com Lua .....</a>	278
<a href="#">Rede com Lua .....</a>	278
<a href="#">Programação com Basic .....</a>	278
<a href="#">Integração com Web Apps .....</a>	278

Página 10

---

## Página 11

<a href="#">Servicos REST .....</a>	278
<a href="#">Protocolo REST .....</a>	279
<a href="#">ESP8266 como um cliente REST .....</a>	279
<a href="#">Fazendo uma solicitação REST usando Mongoose .....</a>	279
<a href="#">ESP8266 como um provedor de serviços REST .....</a>	280
<a href="#">Tasker .....</a>	280
<a href="#">AutoRemote .....</a>	280
<a href="#">DuckDNS .....</a>	282
<a href="#">Aplicativos móveis .....</a>	283
<a href="#">Blynk .....</a>	283
<a href="#">Snippets de amostra .....</a>	283
<a href="#">Formando uma conexão TCP .....</a>	283
<a href="#">Aplicativos de amostra .....</a>	284
<a href="#">Amostra - Acenda um LED com base na chegada de um datagrama UDP .....</a>	284
<a href="#">Amostra - Medição de distância ultrassônica .....</a>	286
<a href="#">Amostra - Scanner WiFi .....</a>	289
<a href="#">Amostra - Trabalhando com cartões micro SD .....</a>	289
<a href="#">Amostra - Reproduzindo áudio de um evento .....</a>	289
<a href="#">Amostra - Uma luz de humor mutável .....</a>	289
<a href="#">Amostra - Rede de inicialização .....</a>	294
<a href="#">Bibliotecas de amostra .....</a>	294

<a href="#">Lista de funções .....</a>	294
<a href="#">authModeToString .....</a>	294
<a href="#">checkError .....</a>	295
<a href="#">delayMilliseconds .....</a>	295
<a href="#">dumpBSSINFO .....</a>	295
<a href="#">dumpEspConn .....</a>	295
<a href="#">dumpRestart .....</a>	295
<a href="#">dumpState .....</a>	295
<a href="#">errorToString .....</a>	296
<a href="#">eventLogger .....</a>	296
<a href="#">eventReasonToString .....</a>	296
<a href="#">flashSizeAndMapToString .....</a>	296
<a href="#">setAsGpio .....</a>	296
<a href="#">setupBlink .....</a>	296
<a href="#">toHex .....</a>	297
<a href="#">Usando FreeRTOS .....</a>	297
<a href="#">A arquitetura de uma tarefa no FreeRTOS .....</a>	298
<a href="#">Bloqueio e sincronização dentro do RTOS .....</a>	300
<a href="#">Listas no RTOS .....</a>	300
<a href="#">ESP8266 - Criação de aplicativos para RTOS .....</a>	300
<a href="#">Consoles com RTOS .....</a>	302
<a href="#">Dicas de depuração .....</a>	303
<a href="#">Desenvolvendo soluções em Linux .....</a>	0,304

---

## Página 12

<a href="#">Construindo um ambiente Linux .....</a>	304
<a href="#">Referência da API .....</a>	313
<a href="#">Referência da API FreeRTOS .....</a>	313
<a href="#">cTaskGetState .....</a>	313
<a href="#">pcTaskGetName .....</a>	313
<a href="#">xEventGroupClear .....</a>	314
<a href="#">xEventGroupCreate .....</a>	314
<a href="#">xEventGroupSetBits .....</a>	314
<a href="#">xEventGroupWaitBits .....</a>	314
<a href="#">xTaskCreate .....</a>	315
<a href="#">vTaskDelay .....</a>	316
<a href="#">vTaskDelayUntil .....</a>	316
<a href="#">vTaskDelete .....</a>	316
<a href="#">xTaskGetCurrentTaskHandle .....</a>	317
<a href="#">xTaskGetTickCount .....</a>	317
<a href="#">vEventGroupDelete .....</a>	317
<a href="#">vTaskList .....</a>	317
<a href="#">vTaskPrioritySet .....</a>	317
<a href="#">vTaskResume .....</a>	317
<a href="#">xTaskResumeAll .....</a>	317
<a href="#">vTaskResumeFromISR .....</a>	317
<a href="#">vTaskSuspend .....</a>	318
<a href="#">vTaskSuspendAll .....</a>	318
<a href="#">xQueueCreate .....</a>	318
<a href="#">vQueueDelete .....</a>	318
<a href="#">xQueuePeek .....</a>	318
<a href="#">xQueueReceive .....</a>	318
<a href="#">xQueueSend .....</a>	318
<a href="#">xQueueSendToBack .....</a>	319
<a href="#">xQueueSendToFront .....</a>	319
<a href="#">vSemaphoreCreateBinary .....</a>	319

<u>xSemaphoreCreateCounting .....</u>	319
<u>vSemaphoreGive .....</u>	319
<u>xSemaphoreGiveFromISR .....</u>	319
<u>vSemaphoreTake .....</u>	319
<u>pvPortMalloc .....</u>	319
<u>pvPortFree .....</u>	319
<u>Processamento de lista .....</u>	319
<u>vListInitialise ....</u>	319
<u>vListInitialiseItem ....</u>	319
<u>vListInsert ....</u>	320
<u>vListInsertEnd .....</u>	320
<u>Referência lwip .....</u>	320
<u>Sockets .....</u>	320

Página 12

**Página 13**

<u>Funções de temporizador .....</u>	321
<u>os_delay_us .....</u>	321
<u>os_timer_arm .....</u>	321
<u>os_timer_disarm .....</u>	322
<u>os_timer_setfn .....</u>	322
<u>system_timer_reinit .....</u>	323
<u>os_timer_arm_us .....</u>	323
<u>hw_timer_init .....</u>	323
<u>hw_timer_arm .....</u>	323
<u>hw_timer_set_func .....</u>	323
<u>Funções do sistema .....</u>	323
<u>system_adc_read .....</u>	323
<u>system_deep_sleep_set_option .....</u>	323
<u>system_get_boot_mode .....</u>	323
<u>system_get_boot_version .....</u>	324
<u>system_get_chip_id .....</u>	324
<u>system_get_cpu_freq .....</u>	324
<u>system_get_flash_size_map .....</u>	324
<u>system_get_rst_info .....</u>	325
<u>system_get_userbin_addr .....</u>	325
<u>system_get_vdd33 .....</u>	325
<u>system_init_done_cb .....</u>	325
<u>system_os_post .....</u>	326
<u>system_os_task .....</u>	326
<u>system_phys_set_roption .....</u>	327
<u>system_phys_set_max_tpw .....</u>	327
<u>system_phys_set_tpw_via_vdd33 .....</u>	327
<u>system_print_meminfo .....</u>	0,327
<u>system_restart_enhance .....</u>	328
<u>system_rtc_clock_cali_proc .....</u>	328
<u>system_set_os_print .....</u>	328
<u>system_show_malloc .....</u>	328
<u>system_rtc_clock_cali_proc .....</u>	329
<u>system_uart_swap .....</u>	329
<u>system_soft_wdt_feed .....</u>	0,329
<u>system_soft_wdt_stop .....</u>	0,329
<u>system_soft_wdt_restart .....</u>	330
<u>system_uart_de_swap .....</u>	0,330
<u>system_update_cpu_freq .....</u>	330
<u>os_memset .....</u>	330

<u>os_memcmp .....</u>	330
<u>os_mempcpy .....</u>	331
<u>os_malloc .....</u>	331
<u>os_calloc .....</u>	331

**Página 14**

<u>os_realloc .....</u>	331
<u>os_zalloc .....</u>	332
<u>os_frec .....</u>	332
<u>os_bzero .....</u>	332
<u>os_delay_us .....</u>	333
<u>os_printf .....</u>	333
<u>os_install_putcl .....</u>	333
<u>os_random .....</u>	334
<u>os_get_random .....</u>	334
<u>os_strlen .....</u>	334
<u>os_streat .....</u>	334
<u>os_strchr .....</u>	335
<u>os_stremp .....</u>	335
<u>os_strcpy .....</u>	335
<u>os_strncmp .....</u>	335
<u>os_strncpy .....</u>	335
<u>os_sprintf .....</u>	336
<u>os strstr .....</u>	336
<u>SPI Flash .....</u>	336
<u>spi_flash_get_id .....</u>	336
<u>spi_flash_erase_sector .....</u>	336
<u>spi_flash_read .....</u>	337
<u>spi_flash_set_read_func .....</u>	337
<u>system_param_save_with_protect .....</u>	337
<u>spi_flash_write .....</u>	337
<u>system_param_load .....</u>	338
<u>WiFi - ESP8266 .....</u>	338
<u>wifi_fpm_close .....</u>	338
<u>wifi_fpm_do_sleep .....</u>	338
<u>wifi_fpm_do_wakeup .....</u>	338
<u>wifi_fpm_get_sleep_type .....</u>	338
<u>wifi_fpm_open .....</u>	338
<u>wifi_fpm_set_sleep_type .....</u>	338
<u>wifi_fpm_set_wakeup_cb .....</u>	338
<u>wifi_get_channel .....</u>	338
<u>wifi_get_ip_info .....</u>	338
<u>wifi_get_macaddr .....</u>	339
<u>wifi_get_opmode .....</u>	339
<u>wifi_get_opmode_default .....</u>	339
<u>wifi_get_phy_mode .....</u>	340
<u>wifi_get_sleep_type .....</u>	340
<u>wifi_get_user_fixed_rate .....</u>	340
<u>wifi_get_user_limit_rate_mask .....</u>	340
<u>wifi_set_broadcast_if .....</u>	340

**Página 15**

<a href="#">wifi_get_broadcast_if .....</a>	341
<a href="#">wifi_set_sleep_type .....</a>	341
<a href="#">wifi_promiscuous_enable .....</a>	341
<a href="#">wifi_promiscuous_set_mac .....</a>	341
<a href="#">wifi_register_rfif_loop_recv_cb .....</a>	341
<a href="#">wifi_register_send_pkt_freedom_cb .....</a>	341
<a href="#">wifi_register_user_ie_manufacturer_recv_cb .....</a>	341
<a href="#">wifi_rfif_loop_recv_close .....</a>	341
<a href="#">wifi_rfif_loop_recv_open .....</a>	341
<a href="#">wifi_send_pkt_freedom .....</a>	341
<a href="#">wifi_set_channel .....</a>	341
<a href="#">wifi_set_event_handle_cb .....</a>	341
<a href="#">wifi_set_ip_info .....</a>	342
<a href="#">wifi_set_macaddr .....</a>	342
<a href="#">wifi_set_opmode .....</a>	342
<a href="#">wifi_set_opmode_current .....</a>	343
<a href="#">wifi_set_phy_mode .....</a>	343
<a href="#">wifi_set_promiscuous_rx_cb .....</a>	343
<a href="#">wifi_set_sleep_type .....</a>	344
<a href="#">wifi_set_user_fixed_rate .....</a>	344
<a href="#">wifi_set_user_ie .....</a>	344
<a href="#">wifi_set_user_limit_rate_mask .....</a>	344
<a href="#">wifi_set_user_rate_limit .....</a>	344
<a href="#">wifi_set_user_sup_rate .....</a>	344
<a href="#">wifi_status_led_install .....</a>	345
<a href="#">wifi_status_led_uninstall .....</a>	345
<a href="#">wifi_unregister_rfif_loop_recv_cb .....</a>	346
<a href="#">wifi_unregister_send_pkt_freedom_cb .....</a>	346
<a href="#">wifi_unregister_user_ie_manufacturer_recv_cb .....</a>	346
<a href="#">Estação WiFi .....</a>	346
<a href="#">  <a href="#">wifi_station_ap_change .....</a></a>	346
<a href="#">  <a href="#">wifi_station_ap_number_set .....</a></a>	346
<a href="#">  <a href="#">wifi_station_connect .....</a></a>	346
<a href="#">  <a href="#">wifi_station_dhcpc_start .....</a></a>	347
<a href="#">  <a href="#">wifi_station_dhcpc_status .....</a></a>	347
<a href="#">  <a href="#">wifi_station_dhcpc_stop .....</a></a>	347
<a href="#">  <a href="#">wifi_station_disconnect .....</a></a>	347
<a href="#">  <a href="#">wifi_station_get_ap_info .....</a></a>	348
<a href="#">  <a href="#">wifi_station_get_auto_connect .....</a></a>	348
<a href="#">  <a href="#">wifi_station_get_config .....</a></a>	0,348
<a href="#">  <a href="#">wifi_station_get_config_default .....</a></a>	349
<a href="#">  <a href="#">wifi_station_get_connect_status .....</a></a>	349
<a href="#">  <a href="#">wifi_station_get_current_ap_id .....</a></a>	349
<a href="#">  <a href="#">wifi_station_get_hostname .....</a></a>	350

**Página 16**

<a href="#">wifi_station_get_reconnect_policy .....</a>	350
<a href="#">wifi_station_get_rssi .....</a>	350
<a href="#">wifi_station_scan .....</a>	350
<a href="#">wifi_station_set_auto_connect .....</a>	351
<a href="#">wifi_station_set_cert_key .....</a>	352

<a href="#">wifi_station_clear_cert_key .....</a>	352
<a href="#">wifi_station_set_config .....</a>	352
<a href="#">wifi_station_set_config_current .....</a>	353
<a href="#">wifi_station_set_reconnect_policy .....</a>	353
<a href="#">wifi_station_set_hostname .....</a>	353
<a href="#">WiFi SoftAP .....</a>	353
<a href="#">wifi_softap_dhcps_start .....</a>	353
<a href="#">wifi_softap_dhcps_status .....</a>	354
<a href="#">wifi_softap_dhcps_stop .....</a>	354
<a href="#">wifi_softap_free_station_info .....</a>	354
<a href="#">wifi_softap_get_config .....</a>	355
<a href="#">wifi_softap_get_config_default .....</a>	355
<a href="#">wifi_softap_get_dhcps_lease .....</a>	356
<a href="#">wifi_softap_get_dhcps_lease_time .....</a>	356
<a href="#">wifi_softap_get_station_info .....</a>	356
<a href="#">wifi_softap_get_station_num .....</a>	356
<a href="#">wifi_softap_reset_dhcps_lease_time .....</a>	356
<a href="#">wifi_softap_set_config .....</a>	357
<a href="#">wifi_softap_set_config_current .....</a>	357
<a href="#">wifi_softap_set_dhcps_lease .....</a>	357
<a href="#">wifi_softap_set_dhcps_lease_time .....</a>	358
<a href="#">wifi_softap_dhcps_offer_option .....</a>	358
<a href="#">WPS WiFi .....</a>	358
<a href="#">wifi_wps_enable .....</a>	358
<a href="#">wifi_wps_disable .....</a>	359
<a href="#">wifi_wps_start .....</a>	359
<a href="#">wifi_set_wps_cb .....</a>	359
<a href="#">Atualizar APIs .....</a>	359
<a href="#">system_upgrade_flag_check .....</a>	359
<a href="#">system_upgrade_flag_set .....</a>	360
<a href="#">system_upgrade_reboot .....</a>	360
<a href="#">system_upgrade_start .....</a>	0,360
<a href="#">system_upgrade_userbin_check .....</a>	360
<a href="#">wifi_promiscuous_enable .....</a>	360
<a href="#">wifi_promiscuous_set_mac .....</a>	360
<a href="#">wifi_promiscuous_rx_cb .....</a>	360
<a href="#">wifi_get_channel .....</a>	361
<a href="#">wifi_set_channel .....</a>	361
<a href="#">APIs de configuração inteligente .....</a>	361

---

**Página 17**

<a href="#">smartconfig_start .....</a>	361
<a href="#">smartconfig_stop .....</a>	361
<a href="#">API SNTP .....</a>	361
<a href="#">ntp_setserver .....</a>	361
<a href="#">ntp_getserver .....</a>	361
<a href="#">ntp_setservername .....</a>	362
<a href="#">ntp_getservername .....</a>	362
<a href="#">ntp_init .....</a>	362
<a href="#">ntp_stop .....</a>	363
<a href="#">ntp_get_current_timestamp .....</a>	363
<a href="#">ntp_get_real_time .....</a>	363
<a href="#">ntp_set_timezone .....</a>	363
<a href="#">ntp_get_timezone .....</a>	364
<a href="#">APIs TCP / UDP genéricas .....</a>	364
<a href="#">espconn_delete .....</a>	364

<a href="#">espconn_dns_setserver .....</a>	364
<a href="#">espconn_gethostbyname .....</a>	365
<a href="#">espconn_port .....</a>	366
<a href="#">espconn_regist_sentcb .....</a>	366
<a href="#">espconn_regist_recvcb .....</a>	366
<a href="#">espconn_send .....</a>	367
<a href="#">espconn_sendto .....</a>	367
<a href="#">ipaddr_addr .....</a>	367
<a href="#">IP4_ADDR .....</a>	368
<a href="#">IP2STR .....</a>	368
<a href="#">APIs TCP .....</a>	368
<a href="#">espconn_abort .....</a>	368
<a href="#">espconn_accept .....</a>	369
<a href="#">espconn_get_connection_info .....</a>	369
<a href="#">espconn_connect .....</a>	370
<a href="#">espconn_disconnect .....</a>	370
<a href="#">espconn_regist_connectcb .....</a>	371
<a href="#">espconn_regist_disconcb .....</a>	371
<a href="#">espconn_regist_reconcb .....</a>	371
<a href="#">espconn_regist_write_finish .....</a>	372
<a href="#">espconn_set_opt .....</a>	373
<a href="#">espconn_clear_opt .....</a>	373
<a href="#">espconn_regist_time .....</a>	374
<a href="#">espconn_set_keepalive .....</a>	374
<a href="#">espconn_get_keepalive .....</a>	375
<a href="#">espconn_secure_accept .....</a>	375
<a href="#">espconn_secure_ca_disable .....</a>	375
<a href="#">espconn_secure_ca_enable .....</a>	375
<a href="#">espconn_secure_set_size .....</a>	375

---

**Página 18**

<a href="#">espconn_secure_get_size .....</a>	375
<a href="#">espconn_secure_delete .....</a>	375
<a href="#">espconn_secure_connect .....</a>	375
<a href="#">espconn_secure_send .....</a>	376
<a href="#">espconn_secure_disconnect .....</a>	376
<a href="#">espconn_tcp_get_max_con .....</a>	376
<a href="#">espconn_tcp_set_max_con .....</a>	376
<a href="#">espconn_tcp_get_max_con_allow .....</a>	376
<a href="#">espconn_tcp_set_max_con_allow .....</a>	376
<a href="#">espconn_recv_hold .....</a>	376
<a href="#">espconn_recv_unhold .....</a>	377
<a href="#">APIs UDP .....</a>	377
<a href="#">espconn_create .....</a>	377
<a href="#">espconn_igmp_join .....</a>	377
<a href="#">espconn_igmp_leave .....</a>	377
<a href="#">APIs de ping .....</a>	378
<a href="#">ping_start .....</a>	378
<a href="#">ping_regist_recv .....</a>	378
<a href="#">ping_regist_sent .....</a>	378
<a href="#">APIs mDNS .....</a>	379
<a href="#">espconn_mdns_init .....</a>	379
<a href="#">espconn_mdns_close .....</a>	379
<a href="#">espconn_mdns_server_register .....</a>	379
<a href="#">espconn_mdns_server_unregister .....</a>	379
<a href="#">espconn_mdns_getservername .....</a>	379

<a href="#">espconn_mdns_set_servername .....</a>	379
<a href="#">espconn_mdns_set_hostname .....</a>	380
<a href="#">espconn_mdns_get_hostname .....</a>	380
<a href="#">espconn_mdns_disable .....</a>	380
<a href="#">espconn_mdns_enable .....</a>	380
<a href="#"><b>GPIO - ESP32 .....</b></a>	<b>380</b>
<a href="#">gpio_config .....</a>	380
<a href="#">gpio_get_level .....</a>	382
<a href="#">gpio_input_get .....</a>	382
<a href="#">gpio_input_get_high .....</a>	382
<a href="#">gpio_intr_enable .....</a>	382
<a href="#">gpio_intr_disable .....</a>	382
<a href="#">gpio_isr_register .....</a>	382
<a href="#">gpio_output_set .....</a>	383
<a href="#">gpio_output_set_high .....</a>	383
<a href="#">gpio_set_direction .....</a>	383
<a href="#">gpio_set_intr_type .....</a>	384
<a href="#">gpio_set_level .....</a>	384
<a href="#">gpio_set_pull_mode .....</a>	384

Página 18

**Página 19**

<a href="#"><b>GPIO - ESP8266 .....</b></a>	<b>385</b>
<a href="#">PIN_PULLUP_DIS .....</a>	386
<a href="#">PIN_PULLUP_EN .....</a>	387
<a href="#">PIN_FUNC_SELECT .....</a>	387
<a href="#">GPIO_ID_PIN .....</a>	387
<a href="#">GPIO_OUTPUT_SET .....</a>	387
<a href="#">GPIO_DIS_OUTPUT .....</a>	388
<a href="#">GPIO_INPUT_GET .....</a>	388
<a href="#">gpio_output_set .....</a>	388
<a href="#">gpio_input_get .....</a>	389
<a href="#">gpio_intr_handler_register .....</a>	389
<a href="#">gpio_pin_intr_state_set .....</a>	389
<a href="#">gpio_intr_pending .....</a>	390
<a href="#">gpio_intr_ack .....</a>	390
<a href="#">gpio_pin_wakeup_enable .....</a>	390
<a href="#">gpio_pin_wakeup_disable .....</a>	391
<a href="#">APIs UART .....</a>	391
<a href="#">UART_CheckOutputFinished .....</a>	391
<a href="#">UART_ClearIntrStatus .....</a>	391
<a href="#">UART_ResetFifo .....</a>	391
<a href="#">UART_SetBaudrate .....</a>	391
<a href="#">UART_SetFlowCtrl .....</a>	391
<a href="#">UART_SetIntrEna .....</a>	391
<a href="#">UART_SetLineInverse .....</a>	391
<a href="#">UART_SetParity .....</a>	392
<a href="#">UART_SetPrintPort .....</a>	392
<a href="#">UART_SetStopBits .....</a>	392
<a href="#">UART_SetWordLength .....</a>	392
<a href="#">UART_WaitTx_fifoEmpty .....</a>	392
<a href="#">uart_init ...</a>	392
<a href="#">uart0_tx_buffer .....</a>	393
<a href="#">uart0_sendStr .....</a>	393
<a href="#">uart0_rx_intr_handler .....</a>	393
<a href="#">I2C Master APIs .....</a>	394
<a href="#">i2c_master_checkAck .....</a>	394

<a href="#">i2c_master_getAck .....</a>	394
<a href="#">i2c_master_gpio_init .....</a>	394
<a href="#">i2c_master_init .....</a>	394
<a href="#">i2c_master_readByte .....</a>	395
<a href="#">i2c_master_send_ack .....</a>	395
<a href="#">i2c_master_send_nack .....</a>	395
<a href="#">i2c_master_setAck .....</a>	395
<a href="#">i2c_master_start .....</a>	395
<a href="#">i2c_master_stop .....</a>	395

---

**Página 20**

<a href="#">i2c_master_writeByte .....</a>	395
<a href="#">APIs SPI .....</a>	395
<a href="#">cache_flush .....</a>	395
<a href="#">spi_lcd_9bit_write .....</a>	396
<a href="#">spi_mast_byte_write .....</a>	396
<a href="#">spi_byte_write_espslave .....</a>	396
<a href="#">spi_slave_init .....</a>	396
<a href="#">spi_slave_isr_handler .....</a>	396
<a href="#">hspi_master_readwrite_repeat .....</a>	396
<a href="#">spi_test_init .....</a>	396
<a href="#">APIs PWM .....</a>	396
<a href="#">pwm_init .....</a>	396
<a href="#">pwm_start .....</a>	397
<a href="#">pwm_set_duty .....</a>	397
<a href="#">pwm_get_duty .....</a>	397
<a href="#">pwm_set_period .....</a>	398
<a href="#">pwm_get_period .....</a>	398
<a href="#">get_pwm_version .....</a>	398
<a href="#">set_pwm_debug_en (uint8 print_en) .....</a>	398
<a href="#">Brincando .....</a>	398
<a href="#">ESP agora .....</a>	399
<a href="#">esp_now_add_peer .....</a>	399
<a href="#">esp_now_deinit .....</a>	399
<a href="#">esp_now_del_peer .....</a>	399
<a href="#">esp_now_get_peer_key .....</a>	399
<a href="#">esp_now_get_peer_role .....</a>	399
<a href="#">esp_now_get_self_role .....</a>	399
<a href="#">esp_now_init .....</a>	399
<a href="#">esp_now_register_recv_cb .....</a>	399
<a href="#">esp_now_register_send_cb .....</a>	399
<a href="#">esp_now_send .....</a>	399
<a href="#">esp_now_set_kok .....</a>	399
<a href="#">esp_now_set_peer_role .....</a>	399
<a href="#">esp_now_set_peer_key .....</a>	399
<a href="#">esp_now_set_self_role .....</a>	399
<a href="#">esp_now_unregister_recv_cb .....</a>	399
<a href="#">esp_now_unregister_send_cb .....</a>	399
<a href="#">SPIFFS .....</a>	399
<a href="#">esp_spiffs_deinit .....</a>	400
<a href="#">esp_spiffs_init .....</a>	400
<a href="#">SPIFFS_check .....</a>	401
<a href="#">SPIFFS_clearerr .....</a>	401
<a href="#">SPIFFS_close .....</a>	401
<a href="#">SPIFFS_closedir .....</a>	402

**Página 21**

<u>SPIFFS_creat .....</u>	402
<u>SPIFFS_erase_deleted_block .....</u>	402
<u>SPIFFS_errno .....</u>	402
<u>SPIFFS_fflush .....</u>	402
<u>SPIFFS_format .....</u>	402
<u>SPIFFS_fremove .....</u>	403
<u>SPIFFS_fstat .....</u>	403
<u>SPIFFS_gc .....</u>	403
<u>SPIFFS_gc_quick .....</u>	403
<u>SPIFFS_info .....</u>	403
<u>SPIFFS_lseek .....</u>	404
<u>SPIFFS_mount .....</u>	404
<u>SPIFFS_mounted .....</u>	405
<u>SPIFFS_open .....</u>	405
<u>SPIFFS_open_by_dirent .....</u>	406
<u>SPIFFS_opendir .....</u>	406
<u>SPIFFS_read .....</u>	407
<u>SPIFFS_readdir .....</u>	407
<u>SPIFFS_remove .....</u>	407
<u>SPIFFS_rename .....</u>	408
<u>SPIFFS_stat .....</u>	408
<u>SPIFFS_unmount .....</u>	408
<u>SPIFFS_write .....</u>	408
<u>Lib-C ...</u>	408
<u>atoi ..</u>	409
<u>atol ..</u>	409
<u>bzero ....</u>	409
<u>calloc ..</u>	409
<u>gratis ..</u>	409
<u>malloc ..</u>	409
<u>memcmp ..</u>	409
<u>memcpy ..</u>	409
<u>memmove ..</u>	409
<u>memset ..</u>	409
<u>os_get_random .....</u>	409
<u>os_random .....</u>	410
<u>printf ..</u>	410
<u>coloca ..</u>	410
<u>rand ..</u>	410
<u>realloc ..</u>	410
<u>snprintf ..</u>	410
<u>sprintf ..</u>	410
<u>streat ..</u>	410
<u>strchr ..</u>	410

**Página 22**

<a href="#">strcmp .....</a>	411
<a href="#">strcpy .....</a>	411
<a href="#">strcspn .....</a>	411
<a href="#">strdup .....</a>	411
<a href="#">strlen .....</a>	411
<a href="#">strncat .....</a>	411
<a href="#">strncmp .....</a>	411
<a href="#">strncpy .....</a>	411
<a href="#">strrchr .....</a>	411
<a href="#">strspn .....</a>	411
<a href="#">strstr .....</a>	412
<a href="#">strtok .....</a>	412
<a href="#">strtok_r .....</a>	412
<a href="#">strtol .....</a>	412
<a href="#">zalloc .....</a>	412
<a href="#"><u>Estruturas de dados .....</u></a>	412
<a href="#"><u>esp_spiffs_config .....</u></a>	412
<a href="#"><u>station_config .....</u></a>	412
<a href="#"><u>struct softap_config .....</u></a>	413
<a href="#"><u>struct station_info .....</u></a>	413
<a href="#"><u>struct dhcps_lease .....</u></a>	414
<a href="#"><u>struct bss_info .....</u></a>	414
<a href="#"><u>struct ip_info .....</u></a>	415
<a href="#"><u>struct rst_info .....</u></a>	415
<a href="#"><u>struct espconn .....</u></a>	416
<a href="#"><u>esp_tcp .....</u></a>	417
<a href="#"><u>esp_udp .....</u></a>	417
<a href="#"><u>struct ip_addr .....</u></a>	417
<a href="#"><u>ipaddr_t .....</u></a>	418
<a href="#"><u>struct ping_option .....</u></a>	418
<a href="#"><u>struct ping_resp .....</u></a>	418
<a href="#"><u>struct mdns_info .....</u></a>	419
<a href="#"><u>enum phy_mode .....</u></a>	419
<a href="#"><u>GPIO_INT_TYPE .....</u></a>	419
<a href="#"><u>System_Event_t .....</u></a>	420
<a href="#"><u>códigos de erro espconn .....</u></a>	422
<a href="#"><u>STATUS .....</u></a>	422
<a href="#"><u>Materiais de referência .....</u></a>	424
<a href="#"><u>Programação C ++ .....</u></a>	424
<a href="#"><u>Definição de classe simples .....</u></a>	424
<a href="#"><u>Funções Lambda .....</u></a>	424
<a href="#"><u>Ignorando avisos .....</u></a>	424
<a href="#"><u>Eclipse .....</u></a>	425
<a href="#"><u>Repartição ESPFS .....</u></a>	425

---

## Página 23

<a href="#"><u>EspFsInit .....</u></a>	425
<a href="#"><u>espFsOpen .....</u></a>	425
<a href="#"><u>espFsClose .....</u></a>	425
<a href="#"><u>espFsFlags .....</u></a>	425
<a href="#"><u>espFsRead .....</u></a>	426
<a href="#"><u>mkespfimage .....</u></a>	426
<a href="#"><u>Repartição ESPHTTPD .....</u></a>	426
<a href="#"><u>httpdInit .....</u></a>	426
<a href="#"><u>httpdGetMimetype .....</u></a>	427
<a href="#"><u>httpdUrlDecode .....</u></a>	427

<a href="#">httpdStartResponse .....</a>	427
<a href="#">httpdSend .....</a>	427
<a href="#">httpdRedirect .....</a>	428
<a href="#">httpdHeader .....</a>	428
<a href="#">httpdGetHeader .....</a>	428
<a href="#">httpdFindArg .....</a>	428
<a href="#">httpdEndHeaders .....</a>	428
<a href="#">Makefiles .....</a>	429
<a href="#">Fóruns .....</a>	431
<a href="#">Documentos de referência .....</a>	431
<a href="#">Github .....</a>	432
<a href="#">Cheats rápidos do Github .....</a>	432
<a href="#">SDK .....</a>	433
<a href="#">Comparações de computador de placa única .....</a>	433
<a href="#">Heroes .....</a>	434
<a href="#">Max Filippov - jcmvbkbc - Compilador GCC para Xtensa .....</a>	434
<a href="#">Ivan Grokhotkov - igrr - IDE Arduino para desenvolvimento ESP8266 .....</a>	434
<a href="#">jantje - Arduino Eclipse .....</a>	435
<a href="#">Richard Sloan - proprietário da comunidade ESP8266 .....</a>	435
<a href="#">Mikhail Grigorev - CHERTS - Eclipse para desenvolvimento ESP8266 .....</a>	435
<a href="#">Mmiscool - Intérprete Básico .....</a>	436
<a href="#">Áreas de Pesquisa .....</a>	436

---

## Página 24

### Introdução

Olá pessoal,

Trabalho no ramo de software há mais de 30 anos, mas até recentemente não jogava diretamente com microprocessadores. Quando comprei um Raspberry PI e, em seguida, um Arduino, infelizmente fiquei viciado. Dentro minha casa estou cercado por computadores de todas as formas, tamanhos e capacidades ... qualquer um deles com ordens de magnitude mais poder do que qualquer um desses pequenos dispositivos ... no entanto, eu ainda me encontrei fascinado.

Quando me deparei com o ESP8266 no início de 2015, isso despertou meu interesse. Eu não tinha tocado em C programação em décadas (atualmente sou um homem de Java). Quando comecei a ler o que estava disponível no caminho de documentação da excelente comunidade em torno do dispositivo, descobri que havia apenas pequenas bolsões de conhecimento. A melhor fonte de informação era (e ainda é) os PDFs oficiais para o SDK de Espressif (os fabricantes do ESP8266), mas mesmo assim é bastante "leve" nos exemplos e no plano de fundo. Como eu estudei o dispositivo, comecei a fazer anotações e minhas páginas de anotações continuaram crescendo e crescendo.

Este livro (se quisermos chamá-lo assim) é minha versão compilada e polida dessas notas. Ao invés de manter a mim mesmo, eu os ofereço a todos nós da comunidade ESP8266 na esperança de que sejam de algum valor. Meu plano é continuar a atualizar este trabalho à medida que todos nós aprendemos mais e compartilhamos o que encontramos no fóruns da comunidade. Como tal, vou relançar o trabalho em intervalos regulares, portanto, verifique novamente no página inicial do livro para o mais recente.

Ao ler, certifique-se de compreender totalmente que existem, sem dúvida, imprecisões, erros em meu  
compreensão e erros na minha escrita. Somente com feedback e tempo seremos capazes de corrigi-los.

Por favor, perdoe os erros gramaticais e ortográficos que meu corretor ortográfico não detectou.

Para perguntas ou comentários sobre o livro, poste neste tópico do fórum:

<http://www.esp8266.com/viewtopic.php?f=5&t=4326>

A página inicial do livro é:

<http://neilkolban.com/tech/esp8266/>

Por favor, não me envie um e-mail diretamente com perguntas técnicas. Em vez disso, vamos usar o fórum e perguntar e responder  
as questões como uma grande comunidade de entusiastas, amadores e profissionais especializados em ESP8266.

## Neil Kolban

Texas, EUA

Página 24

## Página 25

### Visão geral

Um microcontrolador é um circuito integrado capaz de executar programas. Existem muitos exemplos dos que estão no mercado hoje, de diversos fabricantes. Os preços desses microcontroladores continuam caindo. No mercado amador, uma fonte aberta arquitetura chamada "Arduino" que usa a gama de processadores Atmel pegou a imaginação de inúmeras pessoas. As placas contendo esses chips Atmel combinadas com uma convenção para conexões e também um conjunto gratuito de ferramentas de desenvolvimento reduziu o ponto de entrada para brincar com a eletrônica praticamente nulo. Ao contrário de um PC, esses processadores são extremamente baixa com pouca quantidade de memória RAM e recursos de armazenamento. Eles não serão substituindo o desktop ou laptop em breve. Para quem quer mais "oomph" em seus processadores, o pessoal da Raspberry PI desenvolveram um produto muito barato (~ \$ 45) placa que é baseada nos processadores ARM que tem muito mais memória e usa micro SD para armazenamento persistente de dados. Esses dispositivos executam uma variante do Linux sistema operacional. Não vou falar mais sobre o Raspberry PI, pois está na classe de "computador" em oposição a microprocessador.

Esses microcontroladores e arquiteturas são ótimos e sempre haverá um lugar para eles. No entanto, há um problema ... e essa é a rede. Esses dispositivos têm um incrível conjunto de recursos, incluindo entradas e saídas elétricas diretas (GPIOs) e suporte para uma variedade de protocolos, incluindo SPI, I2C, UART e mais, no entanto, nenhum dos eles até agora vêm com rede sem fio incluída.

Não há dúvida (em minha mente) de que o Arduino chamou a atenção de todos. O Arduino é baseado nos chips Atmel e tem uma variedade de tamanhos físicos em seus pegadas de hardware. O microcontrolador principal usado é o ATmega328. Um pode encontrar instâncias desses processadores brutos no ebay por menos de \$ 2 com totalmente construído placas contendo-os por menos de \$ 3. Isso é 10-20 vezes mais barato do que a framboesa

PI. Claro, obtém-se drasticamente menos do que o Raspberry PI, então a comparação pode tornam-se estranhos ... no entanto, se o que se quer fazer é mexer na eletrônica ou fazer alguns dispositivos simples que se conectam a LEDs, interruptores ou sensores e, em seguida, os recursos funcionais precisava se tornar mais perto.

Entre eles, o Arduino e o Raspberry PI parecem ter todas as necessidades atendidas. Se fosse esse o caso, este seria um livro muito curto. Vamos adicionar o toque de que nós começou com... rede sem fio. Para que um dispositivo mova um chassi de robô ou flash Padrões de LED ou fazer alguns ruídos ou ler dados de um sensor e bipar quando a temperatura fica muito alta ... todos esses projetos são ótimos e valiosos. No entanto, nós somos todos muito conscientes do valor da Internet. Nossos computadores estão conectados à Internet, nossos telefones estão conectados, assistimos TV (Netflix) pela Internet, jogamos a Internet, a gente socializa (??) pela Internet ... e assim por diante. A Internet se tornou

Página 25

## Página 26

uma mercadoria tão básica que riríamos se alguém nos oferecesse um novo computador ou um telefone que não tinha a capacidade de ficar "on-line".

Agora imagine o que um microcontrolador com Internet sem fio nativa poderia fazer por nós? Esta seria um processador que poderia executar aplicativos tão bem ou melhor do que um Arduino, que teria GPIO e suporte de protocolo de hardware, teria RAM e flash memória ... mas teria o novo recurso matador que também seria capaz de formar Conexões de Internet. E isso ... simplesmente ... é o que o dispositivo ESP8266 é. É um microprocessador alternativo aos já mencionados, mas também possui WiFi e TCP / IP (Transmission Control Protocol / Internet Protocol) suporte já integrado. O que é mais, também não é muito mais caro do que um Arduino. Pesquisando no ebay, encontramos Embarque ESP8266 abaixo de \$ 3.

### O ESP8266

O ESP8266 é o nome de um microcontrolador projetado pela Espressif Systems. A Espressif é uma empresa chinesa com sede em Xangai. O ESP8266 anuncia a si mesmo como uma solução de rede WiFi independente, oferecendo-se como uma ponte entre os micro controlador para WiFi... e... também é capaz de executar aplicativos independentes.

A produção em volume do ESP8266 não começou até o início de 2014, o que significa que, no esquema das coisas, esta é uma entrada totalmente nova na linha de processadores. E ... Em nosso mundo ávido por tecnologia, novo comumente equivale a interessante. Alguns anos após a produção IC, 3<sup>rd</sup> OEMs partido estão tomando esses chips e construção de "fuga placas "para eles. Se eu lhe desse um ESP8266 bruto direto da fábrica, é improvável que saibamos o que fazer com um. Eles são muito pequenos e virtualmente impossíveis para aficionados anexar fios para permitir que eles sejam plugados em placas de ensaio. Agradecidamente, esses OEMs compram em massa os ICs, projetam circuitos básicos, projetam placas de circuito impresso e construir placas pré-fabricadas com os ICs pré-anexados imediatamente prontos para o nosso usar. São essas placas que captam nosso interesse e que podemos comprar por alguns dólares no ebay.

Há uma variedade de estilos de placa disponíveis. Os dois em que vou me concentrar têm recebido os nomes ESP-1 e ESP-12. É importante notar que existe apenas um Processador ESP8266 e é este processador que se encontra em TODAS as placas breakout. o que distingue uma placa da outra é o número de pinos GPIO expostos, a quantidade da memória flash fornecida, o estilo dos pinos do conector e várias outras considerações relacionados com a construção. Do ponto de vista da programação, eles são todos iguais.

---

**Página 27****Maturidade**

O ESP8266 é um novo dispositivo na arena. Ele existe desde apenas o verão de 2014, mas já despacha volumes de produção na casa das dezenas de milhões.

Todos e tudo tem que começar de algum lugar. Isso significa que há um novo riqueza do território a ser explorado e novos recursos e funções e padrões de uso para seja descoberto. Por outro lado, ainda não tem a riqueza de tutoriais, amostras e vídeos que acompanham outros sistemas de microcontroladores. Sua documentação não é brilhante e algumas das questões centrais sobre seu uso ainda estão sendo examinadas. Como isto sentar com você é uma função de sua intenção de mexer nessa área. Se você quiser seguir os caminhos que já foram seguidos muitas vezes antes, outros processadores serão mais atraente. No entanto, se você gosta de uma sensação de aventura e de entrar no "andar térreo" de uma nova chegada, os desafios que nós (a comunidade ESP8266) estamos tentando resolver pode excitá-loativamente em vez de dissuadi-lo.

É também uma das principais razões pelas quais pessoas como eu passam muitas e muitas horas estudando e documentando o que encontramos ... para que outros possam construir com base no que foi aprendido sem reinventar a roda.

Será que a empolgação com os processadores ESP8266 pode fracassar? Sim ... esses dispositivos podem apenas ser um flash na panela e alguns anos a partir de agora, o aquarista não vai dar um segundo pensamento sobre eles. Mas o que peço é que aborde o dispositivo com a mente aberta.

**A especificação ESP8266**

Quando nos aproximamos de um novo dispositivo eletrônico, gostamos de saber sobre suas especificações. Aqui estão alguns dos pontos mais importantes:

Voltagem	3,3 V
Consumo atual	10uA - 170mA
Memória flash anexável	16 MB máx. (512 K normal)
Processador	Tensilica L106 de 32 bits
Velocidade do processador	80-160 MHz
RAM	32K + 80K
GPIOs	17 (multiplexado com outras funções)
Analógico para Digital	1 entrada com 1024 passo (10 bits) resolução
Supporte 802.11	b / g / n / d / e / i / k / r
Máximo de conexões TCP simultâneas	5

A questão de determinar por quanto tempo um ESP8266 pode funcionar com baterias é um interessante 1. O consumo atual está longe de ser constante. Ao transmitir com potência total,

---

**Página 28**

pode consumir 170mA, mas quando em sono profundo, ele só precisa de 10uA. Isso é bastante diferença. Isso significa que o tempo de execução de um ESP8266 em um reservatório de corrente fixa é não apenas em função do tempo, mas também do que está fazendo durante esse tempo ... e isso é uma função do programa nele implantado.

O ESP8266 é projetado para ser usado com um módulo de memória de parceiro e isso é a maioria comumente memória flash. A maioria dos módulos vem com algum flash associado com eles. Perceba que o flash tem um número finito de apagamentos por página antes que algo falhe. Eles são avaliados em cerca de 10.000 apagamentos. Normalmente, isso não é um problema de configuração alterar gravações ou gravações diárias de log ... mas se seu aplicativo estiver gravando novos dados continuamente extremamente rápido, então isso pode ser um problema e sua memória flash falhará.

**Módulos ESP8266**

O circuito integrado ESP8266 vem em um pacote pequeno, talvez cinco milímetros quadrado. Obviamente, a menos que você seja um soldador mestre, você não vai fazer muito com aquela. A boa notícia é que vários fornecedores criaram painéis de discussão que tornar o trabalho muito mais fácil para você. Aqui, listamos alguns dos módulos mais comuns.

**ESP-12**

A configuração atual mais popular e flexível disponível hoje é chamada de ESP-12. Ele expõe a maioria dos pinos GPIO para uso. O módulo ESP-12 básico realmente precisa de seu próprio módulo expansor para torná-lo amigável e protoboard de 0,1".

Esta é a aparência de um dispositivo ESP-12 quando montado em um extensor de placa de ensaio borda:

O pino fora da placa extensora tem a seguinte aparência:

---

**Página 29**

O ESP-12 possui um LED azul montado na superfície no canto superior direito. Este LED pisca quando há tráfego UART.

Aqui está uma descrição dos vários pinos:

<b>Nome</b>	<b>Descrição</b>
VCC	3,3V.
GPIO 13	Também usado para SPI MOSI.
GPIO 12	Também usado para SPI MISO.
GPIO 14	Também usado para SPI Clock.
GPIO 16	
CH_PD	Ativar chip. Deve ser alto para operação normal. <ul style="list-style-type: none"> <li>• 0 - Desativado</li> <li>• 1 - habilitado</li> </ul>
ADC	Entrada analógica para digital
DESCANSO	Reinicialização externa. <ul style="list-style-type: none"> <li>• 0 - Reiniciar</li> <li>• 1 - normal</li> </ul>
TXD	UART 0 transmitir.
RXD	UART 0 Receba.
GPIO 4	GPIO regular.
GPIO 5	GPIO regular.
GPIO 0	Deve ser <b>alto</b> na inicialização, baixo para atualização do flash.
GPIO 2	Deve ser <b>alto</b> na inicialização.
GPIO 15	Deve estar com <b>pouca</b> inicialização e flash.
GND	Chão.

Aqui está um esquema para conectar uma instância:

A seguir, vemos uma imagem desse circuito construído em uma placa de ensaio.

Página 30

---

**Página 31**

Se desejarmos apenas usar nossa placa de breakout, temos o seguinte quando montados em um breadboard, podemos ter a seguinte configuração:

Isso nos dá dois conjuntos de conectores de 8 pinos. O primeiro conjunto é:

Página 31

---

## Página 32

### Conjunto 1

Alfinete	Cor
GND	laranja
GPIO15	Amarelo
GPIO2	Verde
GPIO0	Azul
GPIO5	Roxo
GPIO4	Cinza
RXD	Branco
TXD	Preto

O segundo conjunto é:

### Conjunto 2

Alfinete	Cor
VCC	laranja
GPIO13	Amarelo
GPIO12	Green
GPIO14	Blue
GPIO16	Roxo
CH_PD	Cinza
ADC	Branco
DESCANSO	Preto

### ESP-1

A placa ESP-1 é uma ESP8266 em uma placa de 8 pinos. Não é nada amigável para a placa de ensaio mas felizmente podemos fazer adaptadores para ele com extrema facilidade. O ESP-1 foi um dos

as primeiras pranchas disponíveis e que não sejam o preço, devem ser totalmente descontadas. Ele apenas fornece um pequeno subconjunto das imagens fornecidas por outras placas. No entanto, o ESP-1 foi produzido em grande volume e ainda mantém uma das menores pegadas físicas. Se você só precisa de alguns I / Os ou precisa de um tamanho muito pequeno, ainda pode haver algum valor em esta seleção.

Página 32

## Página 33

O pino para fora do dispositivo é o seguinte:

### Descrição da Cor da Função

TX	Transmite
RX	Receber. <b>Sempre</b> use um conversor de nível para entrada dados. Este dispositivo <b>não</b> é tolerante a 5V.
CH_PD	Ativar chip. Deve ser alto para operação normal. <ul style="list-style-type: none"> <li>• 0 - Desativado</li> <li>• 1 - habilitado</li> </ul>
RST	Reinicialização externa. <ul style="list-style-type: none"> <li>• 0 - Reiniciar</li> <li>• 1 - normal</li> </ul>
GPIO 0	Deve ser <b>alto</b> na inicialização, baixo para atualização do flash.
GPIO 2	Deve ser <b>alto</b> na inicialização.
VCC	3,3 V
GND	Chão

Um circuito simples é mostrado abaixo. Observe que os pinos TX e RX são mostrados **não** conectados. Lembre-se de **sempre** usar um conversor de nível para o pino RX no dispositivo como é **não** 5V tolerante.

aqui está um circuito alternativo:

---

**Página 34**

Aqui está o circuito em uma placa de ensaio que demonstrou funcionar perfeitamente.

---

**Página 35**

Se desejarmos adicionar botões de aterramento para RESET e GPIO 0, a seguir estão alguns circuitos:

Página 35

---

**Página 36**

Quando pressionamos o botão de reinicialização, faz sentido que seja apenas um toque momentâneo.  
Aqui está um circuito para isso:

Página 36

---

**Página 37**

A velocidade de conexão serial padrão parece ser 115200.

Veja também:

- YouTube: [Detalhes do PIN ESP8266 ESP-01, primeiros passos](#)
- YouTube: [ESP8266 ESP-01 e conexões de conversor USB para serial \(CP2102 Silicon Labs\)](#)

Página 37

---

## Página 38

### Adafruit HUZZAH

O Adafruit HAZZAH é uma placa de apoio para o ESP8266. É o mais protoboard amigável das soluções que encontrei até agora.

Veja também:

- [Adafruit HUZZAH](#)

### NodeMCU devKit

Este módulo vem com um conector USB integrado e uma rica variedade de pinagens. Isto é também imediatamente amigável ao breadboard (se um deles tiver duas placas).

O mapeamento de pinos neste dispositivo é o seguinte:

Página 38

---

## Página 39

Existem atualmente dois tipos de placa NodeMCU chamados v0.9 e v1.0. Aqui estão as principais diferenças:

Função	NodeMCU v0.9	NodeMCU v1.0
USB	CH340 baseado	Baseado em CP2012
ESP8266	ESP-12E	ESP-12E

Quando conectado via USB em uma máquina Windows, o conector serial mostra como " USB-SERIAL CH340 ".

## Página 40

Se por algum motivo o conector serial USB não aparecer, pesquise na internet por drivers para o CH340G para o sistema operacional do seu PC.

Este dispositivo não é especialmente amigável para protoboard único (embora possa deixar uma linha de pinos expostos), mas se encaixa perfeitamente em duas placas de ensaio que estão lado a lado. eu recomendo adicionar um segundo USB → UART conectado da seguinte forma:

- UART GND → GND
- UART RX → GPIO2 (TXD1) [NodeMCU: D4]

Além disso, um botão entre GND e RST também fornecerá uma capacidade de redefinição.

Veja também:

- [Página inicial do NodeMCU](#)
- [GitHub: nodemcu / nodemcu-devkit-v1.0](#)
- [GitHub: nodemcu / nodemcu-devkit](#)
- 
- 

### node.IT (também conhecido como ESP-210)

Veja também:

- [ESP-210](#)

### SparkFun WiFi Shield - ESP8266

SparkFun produziu um escudo WiFi para o Arduino. Este é um ESP8266 montado em um PCB bem projetado que combina com o Arduino. Isso torna a comunicação com o ESP8266 via comandos AT extremamente fácil, sem necessidade de fiação. Simplesmente empurre a placa de proteção nos soquetes do Arduino e pronto.

Veja também:

- [SparkFun WiFi Shield - ESP8266](#)

---

**Página 41****Espresso Lite**

Veja também:

- [Espresso Lite](#)

**Wemos D1**

O Wemos D1 fornece uma placa no estilo Arduino Uno completa com vários poderes escolhas e cabeçalhos femininos em ambos os lados do quadro. Você deveria ser mais confortável trabalhando em uma plataforma do tamanho do Arduino Uno, isso é um bom candidato.

Veja também:

- [Wemos D1](#)

**Carvalho por digistump**

Veja também:

- [Carvalho por digistump](#)

**Conectando-se ao ESP8266**

O ESP8266 é um dispositivo WiFi e, portanto, iremos eventualmente conectar a ele usando WiFi protocolos, mas algum bootstrapping é necessário primeiro. O dispositivo não sabe o que rede à qual se conectar, qual senha usar e outros parâmetros necessários. Este de curso assume que estamos nos conectando como uma estação, se desejarmos que o dispositivo seja um acesso apontar ou desejarmos carregar nossos próprios aplicativos nele, a história se aprofunda. Isso implica que existe uma forma de interagir com o dispositivo diferente de WiFi e existe ... o a resposta é UART (serial). O ESP8266 tem uma interface UART dedicada com pinos rotulados como TX e RX. O pino TX é a transmissão ESP8266 (saída de ESP8266) e o pino RX é usado para receber dados (entrada no ESP8266). Esses os pinos podem ser conectados a um parceiro UART. De longe o mais fácil e conveniente nosso parceiro é um conversor USB → UART. Eles são discutidos em detalhes posteriormente no livro. Por enquanto, vamos supor que já os tenhamos configurado. Através do UART, podemos anexar um emulador de terminal para enviar pressionamentos de tecla e receber dados do ESP8266 exibidos como personagens na tela. Isso é usado extensivamente quando trabalhando com os comandos AT. Um segundo objetivo do UART é receber binário dados usados para "piscar" a memória flash do dispositivo para gravar novos aplicativos para execução. Há uma variedade de ferramentas técnicas à nossa disposição para realizar essa tarefa.

Quando usamos um UART, precisamos considerar o conceito de taxa de transmissão. Isto é o velocidade de comunicação de dados entre o ESP8266 e seu parceiro. Durante a inicialização, o ESP8266 tenta determinar automaticamente a taxa de transmissão do parceiro e combiná-la. Ele assume um padrão de 74880 e se você tiver um terminal serial conectado, verá um mensagem como:

Página 41

---

**Página 42**

data de 8 de janeiro de 2013, primeira causa: 2, modo de inicialização: (1,0)

se estiver configurado para receber em 74880.

O ESP8266 tem um segundo UART associado a ele que é apenas de saída. Um dos

os principais objetivos deste segundo UART é gerar diagnósticos e depuração em formação. Isso pode ser extremamente útil durante o desenvolvimento e, como tal, recomendamos conectar **dois** conversores USB → UART ao dispositivo. O segundo UART é multiplexado com pino GPIO2.

Veja também:

- [Conversores USB para UART](#)
- [Programação de Comando AT](#)
- [Carregando um programa no ESP8266](#)

## Teoria WiFi

Ao trabalhar com um dispositivo orientado para WiFi, é importante que tenhamos pelo menos alguns compreensão dos conceitos relacionados ao WiFi. Em um alto nível, WiFi é a capacidade de participar de conexões TCP / IP por meio de um link de comunicação sem fio. WiFi é especificamente o conjunto de protocolos descritos na arquitetura de LAN sem fio IEEE 802.11.

Nesta história, um dispositivo chamado Ponto de Acesso Wireless (ponto de acesso ou AP) atua como o centro de todas as comunicações. Normalmente, ele está conectado (ou age como) como um roteador TCP / IP para o resto da rede TCP / IP. Por exemplo, em sua casa, é provável que você tenha um Ponto de acesso WiFi conectado ao seu modem (cablo ou DSL). As conexões WiFi são então formado para o ponto de acesso (através de dispositivos chamados estações) e fluxos de tráfego TCP / IP através do ponto de acesso à Internet.

Página 42

---

Página 43

Os dispositivos que se conectam aos pontos de acesso são chamados de "estações":

Um dispositivo ESP8266 pode desempenhar o papel de um ponto de acesso, uma estação ou ambos ao mesmo tempo.

Muito comumente, o ponto de acesso também tem uma conexão de rede com a Internet e atua como uma ponte entre a rede sem fio e a rede TCP / IP mais ampla que é a Internet.

Uma coleção de estações que desejam se comunicarumas com as outras é chamada de Básica Conjunto de serviços (BSS). A configuração comum é conhecida como Infraestrutura BSS. Neste modo, todas as comunicações de entrada e saída de uma estação individual são roteados através do ponto de acesso.

Página 43

---

## Página 44

Uma estação deve se associar a um ponto de acesso para participar da história. UMA estação só pode ser associada a um único ponto de acesso de cada vez.

Cada participante da rede possui um identificador exclusivo denominado endereço MAC. Isso é um valor de 48 bits.

Quando temos vários pontos de acesso dentro do alcance sem fio, a estação precisa saber com qual conectar. Cada ponto de acesso tem um identificador de rede chamado BSSID (ou mais comumente apenas SSID). SSID é **um** identificador de s et **id de serviço**. É um valor de 32 caracteres que representa o destino dos pacotes de informações enviados pela rede.

Veja também:

- Wikipedia - [Ponto de acesso sem fio](#)
- Wikipedia - [IEEE 802.11](#)
- Wikipedia - [Acesso protegido por Wi-Fi](#)
- Wikipedia - [IEEE 802.11i-2004](#)

## Programação de Comando AT

A maneira mais rápida e fácil de começar a usar um ESP8266 é acessá-lo por meio do AT interface de comando.

Quando pensamos em um dispositivo ESP8266, descobrimos que ele tem um UART (Serial) integrado conexão. Isso significa que ele pode enviar e receber dados usando o UART protocolo. Também sabemos que o dispositivo pode se comunicar com wi-fi. E se tivéssemos um

aplicativo executado no ESP8266 que recebeu "instruções" recebidas pelo link serial, os executou e depois retornou uma resposta? Isso nos permitiria usar o ESP8266 sem nunca ter que saber as linguagens de programação que são nativas para o dispositivo. Isso é exatamente o que um programa que até agora foi encontrado para ser pré-instalado em o ESP8266 faz por nós. O programa é chamado de "processador de comando AT", denominado após a formatação dos comandos enviados pelo link serial. Esses comandos são todos prefixado com "AT" e segue (aproximadamente) o estilo conhecido como "conjunto de comandos Hayes".

Página 44

---

## Página 45

Se pensarmos em um aplicativo que deseja utilizar os serviços do ESP8266 como cliente e o ESP8266 como um servidor capaz de atender a esses comandos como um servidor, então o cliente envia sequências de caracteres por meio da conexão UART ao servidor e ao servidor responde com o resultado.

A Espressif publica um conjunto completo de documentação de comandos AT que pode ser encontrada em a página do fórum deles em:

- [http://espressif.com/sites/default/files/documentation/4a-esp8266\\_at\\_instruction\\_set\\_en.pdf](http://espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf)

Existem dois documentos principais:

- Conjunto de instruções ESP8266EX AT
- Exemplos de comando ESP8266EX AT

### Comandos

Quando alguém conecta um ESP8266 a um conversor serial, a próxima pergunta será "É está funcionando? ". Quando conectamos um monitor serial, o primeiro comando que podemos enviar é " AT " que deve responder com um simples " OK ".

Uma instrução passada ao dispositivo segue uma das seguintes opções de sintaxe:

Modelo	Formato	Descrição
Teste	AT +<x>=?	Consulte os parâmetros e sua faixa de valores.
Inquerir	AT +<x>?	Retorna o valor atual do parâmetro.
Definir	AT +<x>=<...>	Defina o valor de um parâmetro.
Executar	AT +<x>	Execute um comando.

---

## Página 46

Todas as instruções "AT" terminam com o par "\r\n".

Página 46

**Página 47**

Comando	Descrição
NO	Retorna OK
AT + RST	Reinic peace o ESP8266.
AT + GMR	Retorna a versão do firmware para o processador de comando AT e o SDK em usar. Atualmente, a resposta retornada é semelhante a: Versão AT: 0.21.0.0 Versão do SDK: 0.9.5
AT + GSLP = <hora>	Coloque o dispositivo em um sono profundo por um tempo em milissegundos. Vai acordar depois este período.
ATE [0   1]	Comandos Echo AT. <ul style="list-style-type: none"><li>• ATE0 - Comandos de eco desligados</li><li>• ATE1 - Comandos de eco ativados</li></ul>
AT + RESTORE	Restaure os padrões das configurações na memória flash.
AT + UART_CUR = <baudrate>, <databits>, <stopbits>, <paridade>, <fluxo controle>	Os bits de dados podem ser 5, 6, 7 ou 8. paridade pode ser 0 = nenhuma, 1 = ímpar, 2 = par o controle de fluxo pode ser: 0 - desabilitar 1 - habilitar RTS 2 - habilitar CTS 3 - habilitar RTS e CTS
AT + UART_DEF = <taxa baud>, <databits>, <stopbits>, <parity>, <controle de fluxo>	
AT + SLEEP?	
AT + SLEEP = <modo de hibernação>	
AT + RFPOWER = <potência TX>	
AT + RFVDD?	
AT + RFVDD = <VDD33>	
AT + RFVDD	
<b>WI-FI</b>	
AT + CWMODE_CUR = <modo>	Define o modo de operação atual. <ul style="list-style-type: none"><li>• 1 - Modo estação</li><li>• 2 - modo AP</li><li>• 3 - Modo AP + Estação</li></ul>

Página 47

**Página 48**

AT +CWMODE_CUR?	Obtenha o modo de operação atual.
AT +CWMODE_CUR=?	Obtenha a lista de modos disponíveis.
AT +CWMODE_DEF = <mode>	Define o modo de operação atual. <ul style="list-style-type: none"> <li>• 1 - Modo estação</li> <li>• 2 - modo AP</li> <li>• 3 - Modo AP + Estação</li> </ul>
AT +CWMODE_DEF?	Obtenha o modo de operação atual.
AT +CWMODE_DEF=?	Obtenha a lista de modos disponíveis.
AT +CWJAP_CUR = <ssid>, <senha> [, <bssid>]	Junte-se à rede WiFi (JAP = Join Access Point).
AT +CWJAP_CUR?	Obtenha as informações de conexão atuais.
AT +CWJAP_DEF = <ssid>, <senha> [, <bssid>]	Junte-se à rede WiFi (JAP = Join Access Point).
AT +CWJAP_DEF?	Obtenha as informações de conexão atuais.
AT +CWLAP	Liste a "Lista de pontos de acesso". A resposta é: +CWLAP: <ecn>, <ssid>, <rssi>, <mac>, <ch> Onde: <ul style="list-style-type: none"> <li>• ecn <ul style="list-style-type: none"> <li>◦ 0 - ABRIR</li> <li>◦ 1 - WEP</li> <li>◦ 2 - WPA_PSK</li> <li>◦ 3 - WPA2_PSK</li> <li>◦ 4 - WPA_WPA2_PSK</li> </ul> </li> <li>• ssid - SSID do AP</li> <li>• rssi - Força do sinal</li> <li>• mac - endereço MAC</li> <li>• ch - Canal</li> </ul>
AT +CWLAP = <ssid>, <mac>, <ch>	Liste um conjunto filtrado de pontos de acesso.
AT +CWQAP	Desconecte-se do AP.
AT +CWSAP_CUR?	Configuração do modo softAP
AT +CWSAP_CUR = <ssid>, <pwd>, <chl>, <ecn>	
AT +CWSAP_DEF?	Configuração do modo softAP

Página 48

**Página 49**

AT +CWSAP_DEF = <ssid>, <pwd>, <chl>, <ecn>	
AT +CWLIF	Lista de IPs conectados no modo softAP
AT +CWDHCP_CUR?	
AT +CWDHCP_CUR = <modo> <en>	Ative ou desative o DHCP.

- modo
  - 0 - softAP
  - 1 - estação
  - 2 - estação softAP +
- en
  - 0 - Habilitar
  - 1 - Desativar

AT +CWDHCP\_DEF?

AT +CWDHCP\_DEF = <mode> <en>

Ative ou desative o DHCP.

- >
  - modo
    - 0 - softAP
    - 1 - estação
    - 2 - estação softAP +
  - en
    - 0 - Habilitar
    - 1 - Desativar

AP +CWAUTOCONN = <ativar>

AT +CIPSTAMAC\_CUR?

Defina / obtenha o endereço MAC da estação.

AT +CIPSTAMAC\_CUR = <mac>

Defina / obtenha o endereço MAC da estação.

AT +CIPSTAMAC\_DEF?

Defina / obtenha o endereço MAC da estação.

AT +CIPSTAMAC\_DEF = <mac>

Defina / obtenha o endereço MAC da estação.

AT +CIPAPMAC\_CUR?

Defina / obtenha o endereço MAC do softAP.

AT +CIPAPMAC\_CUR = <mac>

Defina / obtenha o endereço MAC do softAP.

AT +CIPAPMAC\_DEF?

Defina / obtenha o endereço MAC do softAP.

AT +CIPAPMAC\_DEF = <mac>

Defina / obtenha o endereço MAC do softAP.

AT +CIPSTA\_CUR = <iP>

Defina o endereço IP da estação.

AT +CIPSTA\_CUR?

Obtenha o endereço IP da estação. Por exemplo:  
+ CIPSTA: "0.0.0.0"

Página 49

## Página 50

AT +CIPSTA\_DEF = <iP>

Defina o endereço IP da estação.

AT +CIPSTA\_DEF?

Obtenha o endereço IP da estação. Por exemplo:  
+ CIPSTA: "0.0.0.0"

AT +CIPAP\_CUR?

Defina o endereço IP do softAP.

AT +CIPAP\_CUR = <iP> [, <gat  
eway>, <netmask>]

Defina o endereço IP do softAP.

AT +CIPAP\_DEF?

Defina o endereço IP do softAP.

AT +CIPAP\_DEF = <iP> [, <gat  
eway>, <netmask>]

Defina o endereço IP do softAP.

AT +CIFSR

Retorna o endereço IP e o endereço IP do gateway.

**Rede TCP / IP**

AT + CIPSTATUS

Informações sobre a conexão. O formato da resposta é:  
**STATUS: <stat>**  
+ CIPSTATUS: <id>, <tipo>, <addr>, <porta>, <tipo>  

- Estado
  - 2 - IP obtido
  - 3 - Conectado
  - 4 - Desconectado
- id - Id da conexão
- tipo - TCP ou UDP
- addr - endereço IP
- porta - número da porta
- tipo
  - 0 - ESP8266 é executado como cliente
  - 1 - ESP8266 é executado como servidor

AT + CIPSTART = <tipo>,  
<addr>, <port> [, <local  
porta>, <modo>]

Inicie uma conexão quando CIPMUX = 0.  

- tipo - TCP ou UDP
- addr - endereço IP remoto
- porta - porta remota
- porta local - apenas para UDP
- modo - apenas para UDP
  - 0 - entidade de mesmo nível de destino do UDP é fixa
  - 1 - entidade par de destino pode mudar uma vez
  - 2 - entidade par de destino pode mudar

AT + CIPSTART = <id>,  
<tipo>, <addr>,

Inicie uma conexão quando CIPMUX = 1.  

- id - valor 0-4 de conexão

Página 50

**Página 51**<port> [, <porta local>,  
<modo>]

- tipo - TCP ou UDP
- addr - endereço IP remoto
- porta - porta remota
- porta local - apenas para UDP
- modo - apenas para UDP
  - 0 - entidade de mesmo nível de destino do UDP é fixa
  - 1 - entidade par de destino pode mudar uma vez
  - 2 - entidade par de destino pode mudar

AT + CIPSTART =?

???

AT + CIPSEND = &lt;comprimento&gt;

Envie caracteres de comprimento.

AT + CIPCLOSE

Feche uma conexão.

AT + CIFSR

Obtenha o endereço IP local.

AT + CIPMUX = &lt;modo&gt;

Habilite conexões múltiplas.

- 0 - Conexão única.
- 1 - Múltiplas conexões.

AT + CIPMUX?

Retorna o valor atual para CIPMUX.

- 0 - Conexão única.
- 1 - Múltiplas conexões.

AT + CIPSERVER = &lt;modo&gt; [, &lt;port&gt;]

Configure como um servidor TCP. Se nenhuma porta for fornecida, o padrão é 333. Um servidor pode somente ser criado quando CIPMUX = 1 (permitir conexões múltiplas).

- modo
  - 0 - Excluir servidor (é necessário reiniciar depois)
  - 1 - Criar servidor

AT + CIPMODE = &lt;modo&gt;

Defina o modo de transferência.

- 0 - modo normal.

- 1 - Modo sem verniz.

AT + CIPSTO = <hora>

Defina o tempo limite do servidor. Um valor no intervalo de 0 a 7200 segundos.

AT + CIUPDATE

???

Veja também:

- YouTube - [Comandos AT do tutorial do ESP8266](#)

### **Instalando o processador de comando AT mais recente**

O mais recente processador de comando AT pode sempre ser baixado na forma binária do Site do Espressif. Sempre veja o README que vem com os arquivos e siga as instruções contidas nele. Para carregar as imagens do firmware, você precisará de um

Página 51

## **Página 52**

flashtool. Além disso, você precisará dos arquivos fornecidos com o download. A partir de v0.50, os arquivos necessários e os endereços a serem carregados são:

Arquivo	Endereço
nonboot / eagle.flash.bin	0x00000
nonboot / eagle.irom0text.bin	0x40000
blank.bin	0x3E000
blank.bin	0x7E000

Estas instruções são para chips flash de 512K.

Veja também:

- Espressif - [Download do processador de comando AT - V0.50](#)

### **Montagem de circuitos**

Uma vez que o ESP8266 é um componente eletrônico real, alguns conjuntos físicos são obrigatório. Este livro não tentará cobrir eletrônicos não ESP8266, pois isso é muito assunto grande e amplo por si só. No entanto, o que faremos é descrever algumas das componentes que consideramos extremamente úteis durante a construção de soluções ESP8266.

### **Conversores USB para UART**

Você não pode programar um ESP8266 sem fornecer dados por meio de um UART. O mais fácil maneira de conseguir isso é através do uso de um conversor USB para UART. Eu uso os dispositivos que se baseiam no CP2102, que pode ser encontrado barato no ebay por menos de \$ 2 cada. Outra marca popular são os dispositivos da Future Technology Devices Internacional (FTDI). Você vai querer pelo menos dois. Um para programação e outro para depuração. Eu sugiro comprar mais do que dois apenas no caso ...

Ao fazer o pedido, não se esqueça de obter alguns cabos extensores USB macho-fêmea, pois estão improvável que você consiga conectar seus dispositivos USB a uma placa de ensaio e ao PC em ao mesmo tempo por meio de conexão direta e embora os cabos de conexão funcionem, conectando na placa de ensaio é muito mais fácil. Os cabos conectores USB permitem que você facilmente conecte do PC ao soquete USB ao plugue UART USB. Aqui está uma imagem do

Página 52

---

## Página 53

tipo de cabo conector que eu recomendo. Obtenha-os com um comprimento de cabo tão curto quanto possível. Deve-se dar preferência a 12-24 polegadas.

Quando conectamos um USB → UART em uma máquina Windows, podemos aprender a porta COM que a nova porta serial aparece abrindo o Gerenciador de Dispositivos do Windows. Lá há várias maneiras de fazer isso, uma delas é iniciá-lo a partir do comando DOS janela com:

mmc devmgmt.msc

Na seção chamada Portas (COM e LPT), você encontrará entradas para cada um dos COM portas. As portas COM não fornecem um mapeamento de que um determinado soquete USB é hospedar uma porta COM específica, então minha sugestão é puxar o USB de cada soquete um por um e anote qual porta COM desaparece (ou aparece se você está inserindo um USB).

---

**Página 54**

Veja também:

- [Conectando-se ao ESP8266](#)
- [Trabalhando com serial](#)

**Tábuas de pão**

Acho que nunca posso ter placas de ensaio demais. Eu sugiro pegar um tamanho grande e meio tamanho de placas junto com algum fio conector 24 AWG e um bom par de desencapadores de fio. Mantenha uma lixeira por perto, caso contrário, você se verá despojado até os joelhos. Isolamento e corte partes do fio antes que você perceba. Eu também recomendo alguns homens Dupont fios pré-fabricados machos. O cabo de fita também pode ser útil.

---

**Página 55**

## Poder

Precisamos de eletricidade para fazer esses dispositivos funcionarem. Eu escolho a placa de ensaio MB102 adaptadores de energia acopláveis. Estes podem ser alimentados por uma verruga de parede comum (rede adaptador) ou de USB. Parece que o plugue para alimentação de verrugas na parede é 2,1 mm e o centro positivo, no entanto, eu sugiro fortemente que você leia as folhas de dados de seu fornecedor específico muito cuidado. Também existe uma preocupação potencial de que o soquete do cilindro esteja conectado em paralelo com a entrada USB, o que pode significar que, se você conectar uma entrada de alta tensão (por exemplo, 12V) além de ter uma fonte USB conectada, você pode muito bem danificar seu USB dispositivo. Os dispositivos têm um interruptor principal liga / desliga e um jumper para configurar 3,3 V ou 5 V saídas. Você pode até ter um trilho de placa de ensaio de 3,3 V e o outro 5 V ... mas pegue cuidado para não aplicar 5 V ao seu ESP8266. Por ter dois barramentos de alimentação, um de 3,3 V e o outro a 5V, você pode alimentar o ESP8266 e dispositivos / circuitos que requerem 5V.

Quando o ESP8266 começa a transmitir sem fio, isso pode consumir muita corrente que pode causar ondulações em sua fonte de alimentação. Você também pode ter outros sensores ou dispositivos conectados ao seu abastecimento também. Essas flutuações na tensão podem causar problemas. É altamente recomendável que você coloque um capacitor de 10 micro farad entre + ve e -ve o mais próximo possível de seu ESP8266. Isso fornecerá um reservatório de energia para equilibrar quaisquer ondulações transitórias. Essa é uma das dicas que você ignore por sua conta e risco. Tudo pode funcionar bem sem o capacitor ... até que não ou até que você comece a ter problemas intermitentes e não consiga explicá-los. Deixe-me colocar desta forma, pelos poucos centavos que custa e o dano zero que causa, por que não?

## Multímetro / Sonda lógica / Analisador lógico

Quando o seu circuito não funciona e você está olhando para ele se perguntando o que está errado, você ficaria grato se você tiver um multímetro e uma ponta de prova lógica. Se o seu orçamento esticar, eu também recomendo um analisador lógico baseado em USB, como os feitos por Saleae. Esses permitem que você monitore os sinais que entram ou são produzidos por seu ESP8266. Pense nisso como a melhor fonte de depuração disponível para você.

Veja também:

- [Analisadores lógicos Saleae](#)

## Componentes diversos

Você vai querer o conjunto usual de suspeitos para diversos componentes, incluindo LEDs, resistores, capacitores e muito mais.

### Construção física

Quando você tiver montado seu circuito e escrito seu aplicativo, pode vir um momento em que você deseja tornar sua solução permanente. Nesse ponto, você precisará de um ferro de solda, solda e um pouco de strip-board. Eu também recomendo alguns cabeçalhos femininos soquetes para que você não precise soldar seus ESP8266s diretamente nos circuitos. Não isso só permite que você reutilize os dispositivos (se desejar), mas no infeliz caso você frite um, será mais fácil substituí-lo.

### Configuração recomendada para programação ESP8266

Obviamente, para programar um ESP8266, você realmente precisará obter um ESP8266, mas não é tão fácil. O próprio ESP8266 é um pequeno circuito integrado e é improvável que você consiga usá-lo diretamente. Em vez disso, você vai comprar um dos muitos estilos de placas de breakout que já existem. Os mais comuns são o ESP-1 que expõe 2 pinos GPIO e o ESP-12 que expõe 9. Eu recomendo o ESP-12, pois é apenas marginalmente mais caro para os pinos extras expostos.

Página 56

---

### Página 57

Você também precisará de uma placa de montagem, pois o ESP-12 por si só não tem conector alfinetes. Normalmente, você pode comprar o ESP-12 e a placa de montagem juntos no mesmo tempo. No entanto, verifique com atenção, as placas de montagem podem ser adquiridas separadamente e você precisa validar isso ao fazer o pedido e assumir que está recebendo os dois não estão comprando *apenas* as placas de montagem sem o ESP8266. Você ficará desapontado.

O ESP-12 é então soldado na placa de montagem, então você precisará de um ferro de solda

e algum controle de mão de granulação fina. A soldagem não é das mais fáceis do mundo, pois a os pinos estão extremamente próximos. Por este motivo e por outros, sugiro comprar vários ESP-12s e placas de montagem em vez de apenas um. Também não é difícil fritar seu ESP-12 se você errar na fiação. Depois de montado, deve ter a seguinte aparência:

O meu nunca parece tão "limpo" quando construído, pois minha resina de solda parece descolorir o original atraente base branca da placa de montagem. No entanto, a aparência não é importante.

Supondo que agora você tenha um ESP-12 montado com pinos, sua próxima pergunta será "agora o quê"? É aqui que você vai querer algumas placas de ensaio e fios de conexão. Você poderia use conectores duplos com soquetes fêmeas anexados ao ESP-12 e pinos machos em o outro para anexar à sua placa de ensaio, mas você verá que os fios inevitavelmente se soltarão nos piores momentos possíveis. Você pode montar o ESP-12 em uma placa de ensaio, mas eu costumo descubra que não há espaço suficiente para os fios do conector embaixo dele.

Página 57

## Página 58

Depois de protegido, recomendo **dois** conectores USB → UART. Por que dois? Um dedicado para atualizar o dispositivo e um para depuração.

Para alimentação, eu recomendo o uso de fontes de alimentação de placa de ensaio MB102, no entanto, certifique-se de que você configurou os cabos de jumper para 3,3V. Você vai arruinar o seu ESP8266 se tentar e ligue-o a 5V.

Assim que estiver tudo conectado, você precisará de um PC com duas portas USB abertas.

### Lista de peças

- Pranchas de pão - 2 meio tamanho - \$ 3,50 para 2
- ESP-12 mais placas de montagem - 3 conjuntos - \$ 3,80 cada - \$ 11,40
- CP2102 USB → UARTs - 2 peças - \$ 3,10
- Extensores USB macho para fêmea - 2 peças - \$ 1,00 cada - \$ 2,00
- Fio 24 AWG - 5 metros por \$ 1,12
- Cabeçalhos femininos empilháveis de 8 pinos 2,54 mm - 10 peças por US \$ 3,95
- LEDs difusos vermelhos - Um punhado - \$ 1,00
- Resistores - Alguns 10K, alguns 20K, alguns 330Ohm - Um punhado - \$ 1,00
- Capacitores - Cerca de 10 micro farad - \$ 1,00

Ao todo, trata-se de cerca de US \$ 30 + algum frete. Eu compro todos os meus componentes através do ebay de fornecedores chineses que me dão o preço / qualidade que procuro. O nome do o jogo é paciência. Depois de fazer o pedido, geralmente leva de 2 a 3 semanas para as peças chegue então seja paciente e use o tempo para assistir a vídeos de youtube sobre projetos eletrônicos e os fóruns da comunidade relevantes.

Eventualmente, você provavelmente vai querer construir um circuito permanente para o seu desenvolvimento. Em uma placa de strip, o circuito que construí se parece com:

Página 58

---

## Página 59

### **Configuração para fazer o flash do dispositivo**

Mais adiante no livro, você descobrirá que, quando chega a hora de atualizar o dispositivo com o seu novos aplicativos, você terá que definir alguns dos pinos GPIO como baixos e, em seguida, reiniciar. Esta é a indicação de que agora está pronto para ser exibido. Obviamente, você pode construir um circuito que você usa para atualizar seu firmware e, em seguida, coloque o dispositivo em seu circuito final mas você vai descobrir que durante o desenvolvimento, você vai querer piscar e testar bastante freqüentemente. Isso significa que você vai querer usar fios de jumper e permitir que você se mova os links dos pinos em suas placas de ensaio de sua posição de "flash" para seu "uso normal" posição.

---

**Página 60**

## **Programação**

O ESP8266 permite que você escreva aplicativos que podem ser executados nativamente no dispositivo. Você pode compilar o código da linguagem C e implantá-lo no dispositivo por meio de um processo conhecido como piscando. Para que seus aplicativos façam algo útil, eles devem ser capazes de interagir com o meio ambiente. Isso pode ser fazer conexões de rede ou enviar / receber dados de sensores anexados, entradas e saídas. A fim de fazer isso acontecer, o ESP8266 contém um conjunto básico de funções que podemos vagamente pensar como o sistema operacional do dispositivo. Os serviços do sistema operacional são expostos para ser chamado de seu aplicativo fornecendo um contrato de serviços que você pode aproveitar. Esses serviços são totalmente documentados. A fim de escrever aplicativos com sucesso para implantação, você precisa estar ciente da existência desses serviços. Eles se tornam ferramentas indispensáveis em sua caixa de ferramentas. Por exemplo, se você precisa se conectar a um WiFi ponto de acesso, existe uma API para isso. Para obter o seu endereço IP atual, existe uma API para isso e para obter o tempo desde que o dispositivo foi iniciado, existe uma API para isso. Na verdade, há MUITAS APIs disponíveis para usarmos. A boa notícia é que ninguém está esperando que memorizemos todos os detalhes de seu uso. Em vez disso, é suficiente para saiba que eles existem e têm aonde ir quando quiser consultar os detalhes de como usá-los.

Para gerenciar de maneira sensata o número e a variedade dessas APIs expostas, podemos coletar conjuntos deles juntos em grupos significativos de funções relacionadas. Isso nos dá mais um e a melhor maneira de gerenciar nosso conhecimento e aprendizado deles.

A principal fonte de conhecimento sobre a programação do ESP8266 é o SDK do ESP8266 Guia de API. Links diretos para todos os documentos relevantes podem ser encontrados em [Referência documentos](#).

Veja também:

- [Sistemas Espressif](#) - Fabricantes do ESP8266
- [Espressif Bulletin Board System](#) - local para SDKs, documentos e fóruns

## **Modo de inicialização**

Quando o ESP8266 inicializa, os valores dos pinos conhecidos como MTDO , GPIO0 e GPIO2 são examinado. A combinação dos valores altos ou baixos desses pinos fornece um valor de 3 bits número com um total de 8 valores possíveis de 000 a 111. Cada valor tem um possível significado interpretado pelo dispositivo quando ele inicializa.

Valor [15-0-2]	Decimal Valor	Significado
000	0	Remapeando... detalhes desconhecidos.
001	1	Inicialize a partir dos dados recebidos do UART0. Também inclui piscar a memória flash para os próximos começa normal.
010	2	Acelerador
011	3	Inicialize da memória flash
100	4	SDIO baixa velocidade V2
101	5	SDIO V1 de alta velocidade
110	6	SDIO baixa velocidade V1
111	7	SDIO V2 de alta velocidade

De uma perspectiva prática, o que isso significa é que se quisermos que o dispositivo funcione normalmente, queremos inicializar a partir do flash com os pinos tendo valores 011, enquanto quando deseja atualizar o dispositivo com um novo programa, queremos fornecer 001 para inicializar a partir de UART0.

Observe que MTDO também é conhecido como GPIO15 .

## ESP8266 - Kit de Desenvolvimento de Software (SDK)

### Incluir diretórios

A linguagem de programação C usa um pré-processador baseado em texto para incluir dados no compilação. O pré-processador C tem a capacidade de incluir arquivos de origem C adicionais que, por convenção, são chamados de arquivos de cabeçalho e terminam com o prefixo ".h". Dentro destes arquivos que comumente encontramos definições de tipos de dados e protótipos de funções que são usados durante a compilação. O ESP8266 SDK fornece um diretório chamado "incluir" que contém os arquivos de inclusão fornecidos pela Espressif para uso com o ESP8266. A lista de arquivos de cabeçalho que podemos usar são descritos na tabela a seguir:

Arquivo	Notas
at_custom.h	Definições para extensões personalizadas para o manipulador de comandos AT.
c_types.h	Definições de linguagem C.
eagle_soc.h	Definições e macros de baixo nível. Altamente relacionado à manipulação de bits no nível da CPU. Não faço ideia por que o arquivo é chamado de "água".
espconn.h	Definições de TCP e UDP. Isso tem pré-requisitos de c_types.h e ip_addr.h.

espnow.h	Funções relacionadas ao esp agora suportam.
ets_sys.h	Desconhecido.
gpio.h	Definições para interações GPIO.
ip_addr.h	Definições e macros de endereço IP.
mem.h	Definições para manipulação e acesso à memória.
os_type.h	Definições de tipo de sistema operacional.
osapi.h	Inclui um cabeçalho fornecido pelo <b>usuário</b> chamado "user_config.h".
ping.h	Definições para a capacidade de ping.
pwm.h	Definições para PWM.
queue.h	Definições de fila e lista.
smartconfig.h	Definições para configuração inteligente.
sntp.h	Definições para SNTP.
spi_flash.h	Definições para flash.
upgrade.h	Definições para atualizações.
user_interface.h	Definições de sistema operacional e WiFi. Não tenho nenhuma explicação de por que este arquivo se chama "user_interface", pois obviamente não há IU envolvida com ESP8266s.

## Compilando

O código do aplicativo para um programa ESP8266 é comumente escrito em C. Antes que possamos implantar um aplicativo, devemos compilar o código em instruções de código de máquina binário. Antes disso, porém, vamos passar alguns minutos pensando sobre o código.

Escrevemos código usando um editor e, de preferência, um editor que entende a programação idioma no qual estamos trabalhando. Esses editores fornecem assistência de sintaxe, palavra-chave coloração e até sugestões contextuais. Quando salvamos nosso código inserido, nós compile-o, implante-o e teste-o. Este ciclo é repetido tantas vezes que nós costumam usar um produto que engloba edição, compilação, execução e teste como um todo integrado. O nome genérico para esse produto é um "Desenvolvimento Integrado Ambiente "ou" IDE ". Existem instâncias de ambos gratuitos e pagos. No gratuito acampamento, minhas armas de escolha são Eclipse e Arduino IDE.

---

## Página 63

O Eclipse IDE é um ambiente extremamente rico e poderoso. Escrito originalmente por IBM, seu código-fonte foi aberto há muitos anos. É implementado em Java, o que significa que ele é executado e se comporta de forma idêntica em todas as plataformas comuns (Windows, Linux, OSx). A natureza do Eclipse é que ele é arquitetado como uma série de plug-ins extensíveis. Porque disso, muitos colaboradores em muitas disciplinas ampliaram o ambiente e agora é uma estrutura coesa para quase tudo. Incluído nesta mistura está um conjunto de plug-ins que, em conjunto, são chamados de "Ferramentas do desenvolvedor C" ou "CDT". Se um pega um Eclipse básico e adiciona o CDT, um agora tem um IDE C de primeira linha. No entanto, o que o CDT não fornece (e por um bom motivo) são os C reais compiladores e as próprias ferramentas associadas. Em vez disso, "define-se" as ferramentas que deseja usar para o CDT e o CDT parte daí.

Para nossa história ESP8266, isso significa que se pudermos encontrar (o que podemos) um conjunto de C ferramentas de compilador que pegam o código C e geram o binário Xtensa, podemos usar o CDT para construir nossos programas.

Para tornar as coisas mais interessantes, porém, precisamos perceber que C não é o único linguagem que podemos usar para construir aplicativos ESP8266. Também podemos usar C ++ e conjunto. Você pode se surpreender que eu mencionei a montagem, pois é um nível tão baixo quanto nós pode conseguir, no entanto, há momentos estranhos em que precisamos exatamente disso (felizmente, raramente) ... especialmente quando percebemos que estamos programando diretamente para o metal. As bibliotecas do Arduino (por exemplo) têm pelo menos um arquivo de linguagem assembly.

Para tipos de arquivos físicos, os sufixos usados para diferentes arquivos que encontraremos durante o desenvolvimento inclui:

- .h - arquivo de cabeçalho de linguagem C e C ++
- .c - arquivo fonte da linguagem C
- .cpp - arquivo de origem C ++
- .S - arquivo fonte Assembler
- .o - Arquivo de objeto (fonte compilada)
- .a - Biblioteca de arquivos

Para realizar as compilações, precisamos de um conjunto de ferramentas de desenvolvimento.

Minha preferência pessoal é o pacote para Eclipse, que tem tudo pré-construído e pronto para uso. No entanto, essas ferramentas também podem ser baixadas da Internet como abertas projetos de origem peça por peça.

A macro LOCAL é um sinônimo para a palavra-chave da linguagem C " estática ".

## Página 64

Com a leitura dos documentos, nenhum exemplo publicado de como compilar foi encontrado. Contudo, quando se usa o projeto de código aberto Eclipse, pode-se ver os Makefiles que são usado e isso expõe exemplos de compilação.

Uma compilação típica se parece com:

```
17:57:16 **** Construção da configuração Padrão para o projeto k_blinky ****
mingw32-make.exe -f
C:/Users/IBM_ADMIN/Documents/RaspberryPi/ESP8266/EclipseDevKit/WorkSpace/k_blinky/Make
arquivar tudo
Usuário CC /dominio_usuário.c
AR build /app_app.a
LD build /app.out
```

Informações da seção:

build / app.out: formato de arquivo elf32-xtensa-le

Seções:

Nome Idx	Tamanho VMA LMA File off Align
0 .data	0000053c 3ffe8000 3ffe8000 000000e0 2 ** 4
	CONTEÚDO, ALLOC, LOAD, DATA
1 .rodata 00000878 3ffe8540 3ffe8540 00000620 2 ** 4	CONTEÚDO, ALLOC, LOAD, READONLY, DATA
2 .bss 00009130 3ffe8db8 3ffe8db8 00000e98 2 ** 4	ALLOC
3 .texto 00006f22 40100000 40100000 00000e98 2 ** 2	CONTEÚDO, ALLOC, LOAD, READONLY, CODE
4 .irom0.text 00028058 40240000 40240000 00007dc0 2 ** 4	CONTEÚDO, ALLOC, LOAD, READONLY, CODE

Informações da seção:

Seção |

Descrição Iniciar (hex) | Fim (hex) | Espaço usado

```
-----  
dados | Dados inicializados (RAM) | 3FFE8000 | 3FFE853C | 1340  
rodata | Dados somente leitura (RAM) | 3FFE8540 | 3FFE8DB8 | 2168  
bss | Dados não inicializados (RAM) | 3FFE8DB8 | 3FFF1EE8 | 37168  
texto | Código em cache (IRAM) | 40100000 | 40106F22 | 28450  
irom0_text | Código não anexado (SPI) | 40240000 | 40268058 | 163928  
RAM total usada: 40676  
RAM livre: 41244  
IRam gráis: 4336  
-----
```

```
Execute objcopy, por favor aguarde ...  
objcopy feito  
Execute gen_appbin.exe  
Nenhuma inicialização necessária.  
Gere eagle.flash.bin e eagle.irom0text.bin com sucesso no firmware da pasta.  
eagle.flash.bin -----> 0x00000  
eagle.irom0text.bin -----> 0x40000  
Feito
```

17:57:19 Versão concluída (levou 3s.141ms)

Página 64

---

## Página 65

Podemos construir soluções usando os Makefiles pré-fornecidos, mas, pessoalmente, não gosto mistério, então aqui está uma receita para construir uma solução do zero.

1. Crie um novo projeto em Arquivo> Novo> Projeto C
2. Selecione um projeto Makefile

3. Adicione o diretório de inclusão ESP8266

[https://translate.googleusercontent.com/translate\\_f](https://translate.googleusercontent.com/translate_f)

---

**Página 66**

4. Crie as pastas chamadas " usuário " e " incluir "

5. Crie o arquivo chamado " user\_config.h " no include .

6. Crie o arquivo C chamado " user\_main.c " em user .

**Página 67****7. Crie um Makefile**

```
# Diretório base para o compilador
XTENSA_TOOLS_ROOT? = C:/Espressif/xtensa-lx106-elf/bin
SDK_BASE ?= c:/Espressif/ESP8266_SDK
SDK_TOOLS ?= c:/Espressif/utils
ESPPORT = COM18
#ESPBAUD = 115200
ESPBAUD = 230400

# selecione quais ferramentas usar como compilador, bibliotecário e vinculador
CC : = $(XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-gcc
AR : = $(XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-ar
LD : = $(XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-gcc
OBJCOPY: = $(XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-objcopy
OBJDUMP: = $(XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-objdump
ESPTOOL ?= $(SDK_TOOLS)/esptool.exe

# sinalizadores de compilador usando durante a compilação de arquivos de origem
ALVO = myApp
CFLAGS = -Os -g -O2 -std=gnu90 -Wpointer-arith -Wundef -Werror -WI,-fno-inline-functions -nostdlib -flongcalls -ftext-section-literals -fno-serialize-volatile
-D_ets_-DICACHE_FLASH
MÓDULOS = usuário
BUILD_BASE = construir
FW_BASE = firmware
SDK_LIBDIR = lib
SDK_LDDIR = ld

#
# Nada para configurar ao sul daqui.
#
# sinalizadores de linker usados para gerar o arquivo de objeto principal
LDFLAGS = -nostdlib -WI,-no-check-sections -u call_user_start -WI,-static
# bibliotecas usadas neste projeto, fornecidas principalmente pelo SDK
LIBS = c gcc hal phy pp net80211 lwip wpa main

# script de linker usado para a etapa de linkar acima
LD_SCRIPT = eagle.app.v6.ld

flashimageoptions = --flash_freq 40m --flash_mode qio --flash_size 4m
SDK_LIBDIR : = $(addprefix $(SDK_BASE)/,$(SDK_LIBDIR))
LD_SCRIPT : = $(addprefix -T $(SDK_BASE)/$(SDK_LDDIR)/,$(LD_SCRIPT))
LIBS : = $(addprefix -l,$(LIBS))
APP_AR : = $(addprefix $(BUILD_BASE)/,$(TARGET)_app.a)
TARGET_OUT : = $(adicionar prefixo $(BUILD_BASE)/,$(TARGET).out)
BUILD_DIRS = $(adicionar prefixo $(BUILD_BASE)/,$(MÓDULOS)) $(FW_BASE)
SRC = $(foreach moduleDir,$(MODULES),$(wildcard $(moduleDir)/*.c))
# Substitua todos os xc por xo
OBJS = $(patsubst%.c,$(BUILD_BASE)%.o,$(SRC))

todos: checkdirs $(TARGET_OUT)
echo "Arquivo de imagem construído!"
```

**Página 68**

```
# Crie o arquivo do aplicativo.
# Depende dos objetos compilados.
```

```

$ (APP_AR): $ (OBJS)
$ (AR) -cru $ (APP_AR) $ (OBJS)

# Construa os objetos a partir dos arquivos de origem C
$ (BUILD_BASE) %.o.% .c
$ (CC) $ (CFLAGS) -I $ (SDK_BASE) / include -finclude -c $ <-o $ @

# Verifique se os diretórios necessários estão presentes
checkdirs: $ (BUILD_DIRS)

# Crie a estrutura de diretório que contém as compilações (compilações)
$ (BUILD_DIRS):
    mkdir --parents --verbose $ @

$ (TARGET_OUT): $ (APP_AR)
$ (LD) -L $ (SDK_LIBDIR) $ (LD_SCRIPT) $ (LDFLAGS) -Wl, -start-group $ (LIBS) $ 
$ (APP_AR) -Wl, -grupo-fim -o $ @
$ (OBJDUMP) --headers --section = .data \
    --section = .rodata \
    --section = .bss \
    --section = .text \
    --section = .irom0.text $ @
$ (OBJCOPY) --only-section .text --output-target binary $ @ eagle.app.v6.text.bin
$ (OBJCOPY) --only-section .data --output-target binary $ @ eagle.app.v6.data.bin
$ (OBJCOPY) --apenas seção .rodata - binário de destino de saída $ @
eagle.app.v6.rodata.bin
$ (OBJCOPY) --only-section .irom0.text --output-target binário $ @
eagle.app.v6.irom0text.bin
$ (SDK_TOOLS) /gen_appbin.exe $ @ 0 0 0
mv eagle.app.flash.bin $ (FW_BASE) /eagle.flash.bin
mv eagle.app.v6.irom0text.bin $ (FW_BASE) /eagle.irom0text.bin
rm eagle.app.v6. *

#
# Atualize o ESP8266
#
flash: tudo
$ (ESPTOOL) --port $ (ESPPORT) --baud $ (ESPBAUD) write_flash $ (flashimageoptions)
0x00000 $ (FW_BASE) /eagle.flash.bin 0x40000 $ (FW_BASE) /eagle.irom0text.bin

#
# Limpe todas as construções anteriores
#
limpar:
# Remova forçosamente e recursivamente
rm --recursive --force --verbose $ (BUILD_BASE) $ (FW_BASE)

flashId:
$ (ESPTOOL) --port $ (ESPPORT) --baud $ (ESPBAUD) flash_id

readMac:
$ (ESPTOOL) --port $ (ESPPORT) --baud $ (ESPBAUD) read_mac

```

Página 68

## Página 69

```

imageInfo:
$ (ESPTOOL) image_info $ (FW_BASE) /eagle.flash.bin

```

### 8. Adicionar alvos para pelo menos todos e flash

Veja também:

- [Programação usando Eclipse](#)

### Carregando um programa no ESP8266

Assim que o programa for compilado, ele precisa ser carregado no ESP8266. Esta tarefa é chamada de "flashing". Para fazer o flash do ESP8266, ele precisa ser colocado em um modo onde aceitará o novo programa de entrada para substituir o antigo programa existente. O maneira de fazer isso é reiniciar o ESP8266 removendo e reaplicando a energia ou trazendo o pino REST para baixo e, em seguida, para alto novamente. No entanto, apenas reiniciando o dispositivo não é o suficiente. Durante a inicialização, o dispositivo examina o valor do sinal encontrado no GPIO0 . Se o sinal está baixo, então esta é a indicação de que uma sessão de programação flash está prestes a acontecer. Se o sinal no GPIO0 estiver alto, ele entrará em seu modo de operação normal. Porque disso, é recomendado não deixar o GPIO0 flutuar. Não queremos que entre accidentalmente modo de piscar quando não desejado. Um resistor pull-up de 10k é perfeito.

Podemos construir um circuito que inclua alguns botões. Um para realizar uma reinicialização e um para diminuir o GPIO0 . Pressionar o botão de reinicialização por si só reinicializará o dispositivo. Só isso já é útil. No entanto, se estivermos segurando o botão "GPIO0 baixo" enquanto pressione reset, então somos colocados no modo flash.

Aqui está um diagrama esquemático de exemplo que ilustra um ESP-12 incluindo os botões:

Observe que há um divisor de tensão na saída do conversor de USB para UART TX alfinete. O pensamento por trás disso é lidar com o caso em que a tensão TX de saída é maior do que os 3,3 V desejados na entrada RX do ESP8266. Isso é necessário? A crença é que ele é **não** necessário se tiver certeza de que a tensão de saída TX será 3.3V. Este parece ser o caso da faixa CP2102 de USB para UARTs, no entanto, eu tenho nenhum conhecimento sobre outros dispositivos. O que posso afirmar é que ter um divisor de tensão que reduz 5 V para 3,3 V ainda resulta em uma tensão de nível de saída utilizável para indicar um sinal alto quando alimentado com uma saída real de 3,3V. Eu não sei o quanto perto estou chegando do mínimo Tensão de entrada RX no ESP8266 indicando um alto.

Quando construído em uma placa de ensaio, pode ter a seguinte aparência:

Página 70

---

## Página 71

No entanto, isso tem a desvantagem de nos obrigar a pressionar manualmente alguns botões para carregar um novo aplicativo. Esta não é uma situação horrível, mas talvez tenhamos alternativas?

Quando estamos atualizando nossos ESP8266s, normalmente os conectamos a USB->UART conversores. Esses dispositivos são capazes de fornecer UART usado para programar o ESP8266. Estamos familiarizados com os pinos identificados como RX e TX, mas os pinos identificados como RTS

e DTR ... o que eles podem fazer por nós?  
 RTS que está "pronto para enviar" é uma saída do UART para informar o downstream dispositivo que agora pode enviar dados. Isso é comumente conectado à entrada do parceiro CTS que é "Limpar para enviar", o que indica que agora é aceitável enviar dados. Ambos RTS e CTS são ativos baixos.

O DTR que está "Data Terminal Ready" é usado no controle de fluxo.

Ao fazer o flash do dispositivo usando as ferramentas e receitas do Eclipse, o seguinte é o flash comandos que são executados (por exemplo) e as mensagens registradas:

```
22:34:17 **** Construção da configuração Padrão para o projeto k_blinky ****
mingw32-make.exe -f C:/Users/User1/WorkSpace/k_blinky/Makefile flash
c:/Espressif/utils/esptool.exe -p COM11 -b 115200 write_flash -ff 40m -fm qio -fs 4m
0x000000 firmware / eagle.flash.bin 0x40000 firmware / eagle.irom0text.bin
```

Página 71

## Página 72

Conectando ...  
 Apagando flash ...  
 cabeça: 8; total: 8  
 apagar tamanho: 16384

```
Escrevendo em 0x00000000 ... (3%)
Escrevendo em 0x00000400 ... (6%)
...
Escrevendo em 0x00007000 ... (96%)
Escrevendo em 0x00007400 ... (100%)
30720 bytes gravados em 3,01 segundos (81,62 kbit / s) ...
Apagando flash ...
cabeça: 16; total: 41
tamanho de apagamento: 102400

Escrevendo em 0x00040000 ... (0%)
Escrevendo em 0x00040400 ... (1%)
...
Escrevendo em 0x00067c00 ... (99%)
Escrevendo em 0x00068000 ... (100%)
Gravou 164864 bytes em 16,18 segundos (81,53 kbit / s) ...
```

Deixando...

22:34:40 Versão concluída (levou 23s.424 ms)

Como um exemplo da aparência das mensagens se **não** conseguirmos colocar o ESP8266 em flash modo, temos o seguinte:

```
13:47:09 **** Construção da configuração Padrão para o projeto k_blinky ****
mingw32-make.exe -f C:/Users/User1/WorkSpace/k_blinky/Makefile flash
c:/Espressif/utils/esptool.exe -p COM11 -b 115200 write_flash -ff 40m -fm qio -fs 4m
0x000000 firmware / eagle.flash.bin 0x40000 firmware / eagle.irom0text.bin
Conectando...
Traceback (última chamada mais recente):
  Arquivo "esptool.py", linha 558, em <module>
  Arquivo "esptool.py", linha 160, em conectar
Exceção: Falha ao conectar
C:/Users/User1/WorkSpace/k_blinky/Makefile: 313: a receita para 'flash' de destino falhou
mingw32-make.exe: *** [flash] Erro 255
```

13:47:14 Versão concluída (levou 5s.329 ms)

A ferramenta chamada esptool.py fornece um ambiente excelente para atualizar o dispositivo, mas ele também pode ser usado para "ler" o que está armazenado nele. Isso pode ser usado para fazer backups dos aplicativos contidos antes de atualizá-los com um novo

programa. Dessa forma, você sempre pode voltar ao que tinha antes de sobrescrever. Para exemplo, no Unix:

Página 72

## Página 73

```
esptool.py --port / dev / ttyUSB0 read_flash 0x00000 0xFFFF backup-0x00000.bin
esptool.py --port / dev / ttyUSB0 read_flash 0x10000 0x3FFF backup-0x10000.bin
```

Veja também:

- [Conversores USB para UART](#)
- [Configuração recomendada para programação ESP8266](#)
- [Trabalhando com a memória](#)
- [O que é um UART?](#)
- [esptool.py](#)
- [esptool-e](#)

### Ambientes de programação

Podemos programar o ESP8266 usando o SDK fornecido do Espressif no Windows usando Eclipse. Um capítulo separado sobre a configuração desse ambiente é fornecido. Nos tambem temos a capacidade de programar o ESP8266 usando o IDE do Arduino. Este é potencialmente um jogo mudando a história e também recebeu seu próprio capítulo importante.

Veja também:

- [Programação usando Eclipse](#)
- [Programação usando o Arduino IDE](#)

### Ferramentas de compilação

Há uma série de ferramentas que são essenciais ao construir ESP8266 baseado em C formulários.

ar

A ferramenta de arquivamento é usada para empacotar arquivos-objeto compilados em bibliotecas. Esses bibliotecas terminam com ".a" (arquivo). Uma biblioteca pode ser nomeada ao usar um vinculador e os objetos contidos nele serão usados para resolver fatores externos.

Alguns dos sinalizadores mais comuns usados com esta ferramenta incluem:

- -c - Cria uma biblioteca
- -r - Substitui membros existentes na biblioteca
- -u - Atualizar membros existentes na biblioteca

A sintaxe do comando é:

```
ar -cru libraryName member.o member.o....
```

Veja também:

- [GNU - ar](#)
- [nm](#)

Página 73

**Página 74**

esptool.py

Esta ferramenta é uma implementação de código aberto usada para atualizar o ESP8266 por meio de um porta. Está escrito em Python. As versões estão disponíveis como janelas executáveis que parecem ter sido gerados arquivos ".EXE" a partir do código Python adequado para execução no Windows sem um suporte de instalação de tempo de execução Python.

- -p porta | --port port - A porta serial a ser usada
- -b baud | --baud baud - A taxa de baud a ser usada para serial
- -h - Ajuda
- {command} -h - Ajuda para esse comando
- load\_ram {nome do arquivo} - Baixe uma imagem para a RAM e execute
- dump\_mem {endereço} {tamanho} {nome do arquivo} - Despeja a memória arbitrária para o disco
- read\_mem {address} - Lê a localização arbitrária da memória
- write\_mem {endereço} {valor} {máscara} - Leitura-modificação-gravação para memória arbitrária localização
- write\_flash - Grava um blob binário no flash
  - --flash\_freq {40m, 26m, 20m, 80m} | -ff {40m, 26m, 20m, 80m} - SPI Flash frequência
  - --flash\_mode {qio, qout, dio, dout} | -fm {qio, qout, dio, dout} - SPI Flash modo
  - --flash\_size {4m, 2m, 8m, 16m, 32m, 16m-c1,32m-c1,32m-c2} | -fs {4m, 2m, 8m, 16m, 32m, 16m-c1,32m-c1,32m-c2} - tamanho do Flash SPI em Mbit
  - {address} {fileName} - Endereço para escrever, arquivo para escrever ... repetível
- run - executa o código do aplicativo em flash
- image\_info {arquivo de imagem} - Dump cabeçalhos de uma imagem do aplicativo. Aqui está um exemplo de saída:

Ponto de entrada: 40100004

3 segmentos

Segmento 1: 25356 bytes em 40100000

Segmento 2: 1344 bytes em 3ffe8000

Segmento 3: 924 bytes em 3ffe8540

Soma de verificação: 40 (válido)

- make\_image - Cria uma imagem de aplicativo a partir de arquivos binários
  - --segfile SEGFILE, -f SEGFILE - Arquivo de entrada de segmento

Página 74

**Página 75**

- --segaddr SEGADDR, -a SEGADDR - Endereço básico do segmento
- --entrypoint ENTRYPOINT, -e ENTRYPOINT - Endereço do ponto de entrada
- saída
- elf2image - Cria uma imagem de aplicativo a partir de um arquivo ELF

- --output OUTPUT, -o OUTPUT - Prefixo do nome do arquivo de saída
- --flash\_freq {40m, 26m, 20m, 80m}, -ff {40m, 26m, 20m, 80m} - SPI Flash frequência
- --flash\_mode {qio, qout, dio, dout}, -fm {qio, qout, dio, dout} - SPI Flash modo
- --flash\_size {4m, 2m, 8m, 16m, 32m, 16m-c1,32m-c1,32m-c2}, -fs {4m, 2m, 8m, 16m, 32m, 16m-c1,32m-c1,32m-c2} - tamanho do Flash SPI em Mbit
- - símbolo de entrada ENTRY\_SYMBOL, -es ENTRY\_SYMBOL - símbolo de ponto de entrada nome (padrão 'call\_user\_start')

- read\_mac - Lê o endereço MAC da ROM OTP. Aqui está um exemplo de saída:

MAC AP: 1A-FE-34-F9-43-22  
MAC STA: 18-FE-34-F9-43-22

- flash\_id - Leia o fabricante do flash SPI e a ID do dispositivo. Aqui está um exemplo resultado:

cabeça: 0; total: 0  
apagar tamanho: 0  
Fabricante: c8  
Dispositivo: 4014

- read\_flash - Leia o conteúdo flash SPI
  - endereço - endereço inicial
  - size - tamanho da região a ser despejada
  - nome do arquivo - Nome do dump binário
- erase\_flash - Execute o apagamento do chip no flash SPI. Este é um especialmente útil comando se alguém acaba travando o dispositivo, pois ele deve redefinir o dispositivo para seus padrões.

Veja também:

- [esptool-ck](#)
- [nodemcu-pisca-pisca](#)
- [Carregando um programa no ESP8266](#)
- [Trabalhando com a memória](#)
- Github: [themadinventor / esptool](#)

---

## Página 76

esptool-ck

Outra ferramenta também chamada de esptool-ck . A nomenclatura dessas ferramentas é tão semelhante está começando a ficar desconfortável.

- -eo <filename> - Abre um objeto ELF.
- -es <seção> <nome do arquivo> - Lê a seção nomeada do objeto e escreve para o arquivo nomeado.
- -cc - Fecha o arquivo ELF.
- -bo <filename> - Prepara um arquivo de firmware para o ESP.
- -bm <qio | qout | dio | dout> - Define o modo de interface do chip flash.
- -bz <512K | 256K | 1M | 2M | 4M | 8M | 16M | 32M> - Defina o tamanho do chip flash.
- -bf <40 | 26 | 20 | 80> - Defina a frequência do chip de flash.

- -bs <seção> - Leia a seção ELF e grave a imagem do firmware.
  - -bc - Fecha a imagem do firmware.
  - -v - Aumenta o detalhamento da saída ( -v , -vv , -vvv )
  - -q - Desativa a maior parte da saída
  - -cp <dispositivo> - Dispositivo serial (por exemplo, COM1 )
  - -cd <board> - Selecione o método de reinicialização para reiniciar a placa.
    - nenhum
    - ck
    - wifio
    - nodemcu
  - -cb <baudrate> - Selecione a taxa de transmissão a ser usada.
  - -ca <address> - Endereço da memória flash como o destino do upload.
  - -cf <filename> - Carrega o arquivo nomeado para o flash.

Aqui, por exemplo, está um comando para atualizar uma placa NodeMCU devKit:

```
esptool -cp COM15 -cd nodemcu -cb 115200 -ca 0x00000 -cf myApp_0x00000.bin
```

Aqui está um registro limpo de um upload do IDE do Arduino:

esptool v0.4.5 - (c) 2014 Ch. Klippe [ck@atelier-klippe.de](mailto:ck@atelier-klippe.de)  
definindo a placa para ck  
definindo a taxa de transmissão de 115200 a 115200  
configuração da porta de COM1 para COM11

Página 76

Página 77

```

apagando flash
tamanho: 04c670 endereço: 000000
first_sector_index: 0
total_sector_count: 77
head_sector_count: 16
Adjust_sector_count: 61
tamanho_justificado: 03d000
especomm_send_command: enviando cabeçalho de comando
especomm_send_command: envio de carga útil do comando
definindo tempos limite da porta serial para 10000 ms
definindo tempos limite da porta serial para 1000 ms
especomm_send_command: recebendo 2 bytes de dados
escrevendo flash
.....
```

```

.....
```

iniciando o aplicativo sem reiniciar

```

especomm_send_command: enviando cabeçalho de comando
especomm_send_command: envio de carga útil do comando
especomm_send_command: recebendo 2 bytes de dados
```

fechando bootloader

```
flush start
```

Página 77

## Página 78

```

definindo tempos limite da porta serial para 1 ms
definindo tempos limite da porta serial para 1000 ms
rubor completo
```

Binários correspondentes às versões da ferramenta podem ser encontrados na seção de versões:

<https://github.com/igrr/esptool-ck/releases>

Veja também:

- [esptool.py](#)
- [nodemcu-pisca-pisca](#)
- Github: <https://github.com/igrr/esptool-ck>

gcc

A coleção de compiladores GNU de código aberto inclui compiladores para C e C++. Se nós observar atentamente as sinalizações fornecidas para compilar e vincular o código para o ESP8266 encontramos o seguinte:

Compilando

- -c - Compila o código em um arquivo de objeto .o .
- -Os - Otimiza a geração de código para o tamanho.
- -O2 - Otimiza para desempenho cujo código resulta em um tamanho de código maior. Para Por exemplo, em vez de fazer uma chamada de função, o código pode ser embutido.
- -ggdb - Gera código de depuração que pode ser usado pelo depurador gdb .
- -std = gnu90 - Dialetos de C suportados.
- -Werror - Faz todos os erros de avisos.
- -Wno-address - Não avisa sobre o uso suspeito de endereços de memória.
- -Wpointer-arith - Avisa quando a aritmética de ponteiro é tentada que depende de sizeof .
- -Wundef - Avisa quando um identificador é encontrado em uma diretiva #if que não é uma macro.
- -fno-inline-functions - Não permite que funções sejam substituídas por código em linha.

- `-mlongcalls` - Converte chamadas diretas de linguagem assembly em chamadas indiretas.
- `-mtext-section-literals` - Permite que literais sejam misturados com a seção de texto.
- `-mno-serialize-volatile` - Instruções especiais para definições de voláteis.

Vinculando:

- `-nostdlib` - Não usa bibliotecas de inicialização do sistema C ou C ++ padrão

Página 78

---

## Página 79

Veja também:

- [GCC - A coleção de compiladores GNU](#)

`gen_appbin.py`

A sintaxe desta ferramenta é:

`gen_appbin.py app.out boot_mode flash_mode flash_clk_div flash_size`

- modo de flash
  - 0 - QIO
  - 1 - QOUT
  - 2 - DIO
  - 3 - DOUT
- `flash_clk_div`
  - 0 - 80m / 2
  - 1 - 80m / 3
  - 2 - 80m / 4
  - 0xf - 80m / 1
- `flash_size_map`
  - 0 - 512 KB (256 KB + 256 KB)
  - 1 - 256 KB
  - 2 - 1024 KB (512 KB + 512 KB)
  - 3 - 2.048 KB (512 KB + 512 KB)
  - 4 - 4096 KB (512 KB + 512 KB)
  - 5 - 2.048 KB (1.024 KB + 1.024 KB)
  - 6 - 4096 KB (1024 KB + 1024 KB)

Espera-se que os seguintes arquivos existam:

- `eagle.app.v6.irom0text.bin`
- `eagle.app.v6.text.bin`
- `eagle.app.v6.data.bin`
- `eagle.app.v6.rodata.bin`

A saída desse comando é um novo arquivo chamado `eagle.app.flash.bin`.

---

**Página 80**

faço

Make é um motor de compilação usado para rastrear o que deve ser compilado a fim de construir seu aplicativo de destino. O Make é conduzido por um Makefile. Embora poderoso e simples o suficiente para projetos C simples, pode se tornar complexo muito rapidamente. Se você se encontrar estudando Makefiles escritos por outros, pegue a excelente documentação GNU make e estude-o profundamente.

nodemcu-pisca-pisca

Esta ferramenta é outra instância de um pisca-pisca ESP8266. Ao contrário de algumas das outras ferramentas disponíveis, este é baseado em GUI. De dentro da ferramenta, pode-se selecionar todas as opções que pode-se esperar a inclusão de um ou mais arquivos para flash, a conexão serial e informações e muito mais.

Ao entrar, pode-se clicar no botão "Flash" e o piscar começa com um atraente Barra de progresso.

A seguir está a aparência da ferramenta após completar um flash:

Esta é a aparência da guia de seleção de arquivo flash:

Página 80

---

**Página 81**

E, finalmente, aqui estão as configurações de comunicação:

Embora visualmente atraente, parece ter uma grande desvantagem. Parece muito mais lento para flash do que algumas das outras ferramentas. Isso, é claro, pressupõe que uma tentativa de piscar na mesma taxa de transmissão.

No entanto, mesmo com esta pequena fraqueza, ainda é uma das ferramentas de pisca-pisca mais fáceis de usar disponível e parece ser perfeito para o exibicionista casual. Se eu recomendasse uma ferramenta para ser usado por alguém que só precisava instalar um aplicativo em seu ESP8266 raramente, provavelmente seria isso.

Veja também:

- [esptool.py](#)
- [esptool-ek](#)
- GitHub: [nodemcu / nodemcu-pisca-pisca](#)
- YouTube: [ESP8266 como fazer o flash do firmware do NodeMcu](#)
- [Atualizando o firmware NodeMCU no ESP8266 \(Windows\) - Guia](#)

Página 81

---

## Página 82

nm

Lista os símbolos dos arquivos de objeto.

Sinalizadores úteis:

- --defined-only - Mostra apenas as exportações definidas
- --undefined-only - Mostra apenas exportações indefinidas
- --números de linha

Veja também:

- [ar](#)
- GNU - [nm](#)

objcopia

Veja também:

- GNU - [objcopia](#)

objdump

O comando é xtensa-lx106-elf-objdump localizado em

C:\ Espressif\ xtensa-lx106-elf\ bin .

Algumas das sinalizações mais importantes são:

- --syms - Despeja os símbolos no arquivo.

Veja também:

- Wikipedia - [objdump](#)
- GNU - [objdump](#)
- página man - [objdump\(1\)](#)

xxd

Esta é uma ferramenta aparentemente simples, mas útil. O que ele faz é despejar dados binários contidos dentro de um arquivo em um formato formatado. Um uso poderoso dele é pegar um arquivo binário e produzir uma estrutura de dados em linguagem C que representa o conteúdo do arquivo. Isso significa que você pode pegar dados binários e incluí-los em seus aplicativos. Uma cópia do xxd.exe é distribuído com o SDK fornecido pela Espressif na pasta de ferramentas .

O seguinte irá ler o conteúdo de inFile como dados binários e produzir um arquivo de cabeçalho em outFile .

`xxd -include <inFile> <outFile>`

Página 82

## Página 83

### ESP8266 Linking

Quando os arquivos de origem C e C ++ que constituem o seu projeto foram compilados para seus arquivos objeto, é hora de vinculá-los a bibliotecas para finalizar o executável para ser implantado. Aqui está um exemplo de um comando de vinculação usado para construir um executável.

```
xtensa-lx106-elf-gcc
    -g
    -Os
    -nostdlib
    -WI, -no-check-sections -u call_user_start
    -WI, -static
        "-L ?? / tools / sdk // lib"
        "-L ?? tools / sdk // ld"
        "-Teagle.flash.512k.ld"
    -WI, -wrap, system_restart_local
    -WI, -wrap, register_chipv6_phy
    -o "Release / Test_ESP_RESTClient.elf"
    -WI, -start-group
    xo
    yo
    zo
    -lm
    -lgcc
    -lhal
```

```
-lphy
-lnet80211
-llwip
-lwpa
-lmain
-lpp
-lsmartconfig
-lwps
-lcrypto
-Wl, - grupo final
"-LRelease"
```

Observe que algumas bibliotecas são usadas durante a vinculação. Muitas dessas bibliotecas são fornecidas com o Espressif SDK.

Página 83

## Página 84

Nome da biblioteca	Descrição
no	
cripto	
espnow	
hal	
json	
lwip	
lwip_536	
a Principal	
net80211	
phy	
pp	
pwm	
smartconfig	
ssc	
ssl	
melhoria	
wpa	
wps	

Veja também:

- [Trabalhando com a memória](#)
- O GNU Linker
- Uma introdução ao GNU - compilador e vinculador

**Piscando no ar - FOTA**

Imagine que você construiu um aplicativo ESP8266 fantástico que está contido em um dispositivo embutido. Você envia para seus clientes e está tudo ótimo. De repente, você começa a obter relatórios de que está falhando periodicamente. Você diagnostica o erro e para seu horror, você descobrir que houve um erro de codificação ... mas, felizmente, é fácil e rapidamente corrigido. Vocês agora descubra que você tem um problema. Seu dispositivo embutido não tem um UART exposto e, mesmo que isso acontecesse, seus clientes ficariam em pé de guerra se tivessem que conectá-lo em um computador. Pior, seria um pesadelo de suporte caminhar com os consumidores através da mecânica de conseguir uma recarga quando não fizemos suposições sobre o habilidades técnicas do consumidor final.

Para resolver esse problema, apresentamos o conceito de flashing (ou, mais especificamente, piscando) um aplicativo na Internet por meio de uma conexão sem fio. Esta noção é chamado "Flashing Over The Air" ou FOTA.

Página 84

---

**Página 85**

Genericamente, funciona da seguinte maneira.

O ESP8266 possui uma quantidade de memória flash disponível. Vamos dividir isso memória em duas metades iguais.

Quando um esp8266 navios, a sua candidatura será carregado para o 1<sup>st</sup> metade do flash e vontade ignore a segunda metade. De vez em quando, ele "liga para casa" pela Internet e pergunta se há um conjunto de substituição de firmware (uma nova versão). Se houver, ele fará o download esse novo firmware na 2<sup>nd</sup> metade do flash. Se for totalmente bem-sucedido, o dispositivo será reiniciado e começar a correr o novo firmware a partir do 2<sup>nd</sup> semestre de flash ... agora vai ignorar a 1<sup>st</sup> metade. Uma substituição subsequente do firmware por outra versão será carregada nos 1<sup>st</sup> meio E a história se repete.

Efetivamente, somos capazes de alternar entre duas versões. Com este alto nível teoria sob nossos cintos, vamos agora cavar um pouco mais fundo. Obviamente, se seguirmos esta história, vemos que reduzimos efetivamente a quantidade de flash disponível para hospedar nosso programas pela metade. Isso não parece bom ... por que fazemos isso? A resposta é na verdade, muito simples. Se considerarmos a realidade de que as conexões WiFi podem falhar, a Internet o acesso pode ser perdido e a energia pode simplesmente ser removida de um dispositivo, podemos acabar em a situação em que o upload de um novo código para o flash realmente falha antes de ser concluído. Isso nos deixaria com um código quebrado na área do flash. Se tentássemos substituir "no local" nosso aplicativo existente e tal falha ocorresse, teríamos "brickado" o dispositivo. É provável que um aplicativo substituído que estava apenas meio carregado nem mesmo Bota. Para contornar esse problema, o ESP8266 contém uma bandeira que define qual dos

duas metades possíveis de firmware é atualmente a usada para executar o programa. Apenas quando uma nova versão do firmware foi validada como tendo sido carregada é a bandeira trocada para a outra metade. Se ocorrer um erro durante a substituição upload, então o sinalizador não é alterado e nenhum dano terá sido feito desde que tivemos efetivamente ignorou a segunda metade do flash em primeiro lugar.

Página 85

## Página 86

Vamos agora ir ainda mais fundo. Espressif fornece um código chamado "boot" que é o responsável para inicializar um ESP8266. Quando um ESP8266 é ligado, é este código de inicialização que obtém o controle. É a inicialização que determina como o restante da inicialização do dispositivo irá prosseguir. Quando instalamos um ESP8266, devemos fornecer o boot aplicação e nossa própria lógica de aplicação. De uma perspectiva de espaço de endereço, a bota o programa é carregado no endereço flash 0x00000 para 4KBytes. Nossa aplicativo será carregado do endereço 0x01000 em diante.

Como o ESP8266 pode ter uma variedade de tamanhos de flash, examinamos cada um deles separadamente.

### Flash de 512 KB

Conteúdo	Endereço	Tamanho
Bota	0x0 0000 - 0x0 0FFF	4 KB
App 1 (user1.bin)	0x0 1000 - 0x3 BFFF	236 KB
Parâmetros do usuário	0x3 C000 - 0x3 FFFF	16 KB
Não use	0x4 0000 - 0x4 0FFF	4 KB
App 2 (user2.bin)	0x4 1000 - 0x7 BFFF	236 KB
Parâmetros do sistema	0x7 C000 - 0x7 FFFF	16 KB

### Flash de 1024 KB

Conteúdo	Endereço	Tamanho
Bota	0x0 0000 - 0x0 0FFF	4 KB
App 1 (user1.bin)	0x0 1000 - 0x7 BFFF	492 KB
Parâmetros do usuário	0x7 C000 - 0x7 FFFF	16 KB
Não use	0x8 0000 - 0x8 0FFF	4 KB
App 2 (user2.bin)	0x8 1000 - 0xF BFFF	492 KB
Parâmetros do sistema	0xF C000 - 0xF FFFF	16 KB

### Flash de 2.048 KB - Opção 1

**Página 87**

<b>Conteúdo</b>	<b>Endereço</b>	<b>Tamanho</b>
Bota	0x00 0000 - 0x00 0FFF	4 KB
App 1 (user1.bin)	0x00 1000 - 0x07 BFFF	492 KB
Parâmetros do usuário	0x07 C000 - 0x07 FFFF	16 KB
Não use	0x08 0000 - 0x08 0FFF	4 KB
App 2 (user2.bin)	0x08 1000 - 0x0F BFFF	492 KB
Dados do usuário	0x0F C000 - 0x1F BFFF	1008 KB
Parâmetros do sistema	0x1F C000 - 0x1F FFFF	16 KB

Flash de 2.048 KB - Opção 2

<b>Conteúdo</b>	<b>Endereço</b>	<b>Tamanho</b>
Bota	0x00 0000 - 0x00 0FFF	4 KB
App 1 (user1.bin)	0x00 1000 - 0x07 BFFF	1004 KB
Parâmetros do usuário	0x0F C000 - 0x0F FFFF	16 KB
Não use	0x10 0000 - 0x10 0FFF	4 KB
App 2 (user2.bin)	0x10 1000 - 0x1F BFFF	1004 KB
Parâmetros do sistema	0x1F C000 - 0x1F FFFF	16 KB

Flash de 4096 KB - Opção 1

<b>Conteúdo</b>	<b>Endereço</b>	<b>Tamanho</b>
Bota	0x00 0000 - 0x00 0FFF	4 KB
App 1 (user1.bin)	0x00 1000 - 0x07 BFFF	492 KB
Parâmetros do usuário	0x07 C000 - 0x07 FFFF	16 KB
Não use	0x08 0000 - 0x08 0FFF	4 KB
App 2 (user2.bin)	0x08 1000 - 0x0F BFFF	492 KB
Dados do usuário	0x0F C000 - 0x3F BFFF	3072 KB
Parâmetros do sistema	0x3F C000 - 0x3F FFFF	16 KB

Flash de 4096 KB - Opção 2

**Página 88**

<b>Conteúdo</b>	<b>Endereço</b>	<b>Tamanho</b>
Bota	0x00 0000 - 0x00 0FFF	4 KB
App 1 (user1.bin)	0x00 1000 - 0x07 BFFF	1004 KB
Parâmetros do usuário	0x0F C000 - 0x0F FFFF	16 KB

Não use	0x10 0000 - 0x10 0FFF	4 KB
App 2 (user2.bin)	0x10 1000 - 0x1F BFFF	1004 KB
Dados do usuário	0x1f C000 - 0x3F BFFF	2048 KB
Parâmetros do sistema	0x3F C000 - 0x3F FFFF	16 KB

Veja também:

- Documento Espressif: 99C - ESP8266 - Atualização OTA

## Depurando

Ao escrever programas, podemos descobrir que eles nem sempre são executados conforme o esperado. Executar a depuração em um SOC pode ser difícil, pois não temos depuradores de nível de origem.

### Registro ESP-IDF

A estrutura ESP-IDF fornece um conjunto de recursos de registro. Esses itens de registro podem em seguida, ser inserido em seu próprio aplicativo para diagnosticar problemas ou capturar rastros.

Para usar as funções de registro, devemos incluir " esp\_log.h ".

A função de registro de alto nível é chamada de " esp\_log\_write () " que tem o seguinte assinatura:

```
void esp_log_write (nível esp_log_level_t, tag const char *, formato const char *, ...)
```

Pense nisso como um registrador printf especializado. O formato e os seguintes parâmetros seguem a convenção de estilo printf.

Por padrão, quando o registro é solicitado, a saída é enviada para o fluxo serial primário.

No entanto, podemos substituir esse destino usando a função chamada

esp\_log\_set\_vprintf () . Isso toma como parâmetro uma referência a uma função C que tem a mesma sintaxe de printf . Especificamente:

```
int myPrintFunction (const char * format, va_list arg)
```

Quando desejamos registrar uma mensagem, escolhemos um nível de registro para escrever. Os níveis de registro disponíveis são:

- ESP\_LOG\_NONE

Página 88

---

## Página 89

- ESP\_LOG\_ERROR
- ESP\_LOG\_WARN
- ESP\_LOG\_INFO
- ESP\_LOG\_DEBUG
- ESP\_LOG\_VERBOSE

A saída registrada é do formato:

```
<nível de registro> (<carimbo de hora>) <tag>: <mensagem>
```

Onde o nível de registro é " E ", " W ", " I ", " D " ou " V ". O carimbo de data / hora é o número de milissegundos desde a inicialização.

Também temos uma configuração global que é o nível máximo de registro que devemos registrar. Para exemplo, se definirmos ESP\_LOG\_WARN , as mensagens no nível ESP\_LOG\_WARN , ESP\_LOG\_ERROR

será registrado, mas `ESP_LOG_INFO`, `ESP_LOG_DEBUG` e `ESP_LOG_VERBOSE` serão excluídos. O parâmetro de tag para a função de registro fornece uma indicação de qual lógica componente / módulo emite a mensagem. Isso fornece contexto para o que de outra forma poderia ser mensagens ambíguas.

As macros da linguagem C são fornecidas para simplificar o uso do registro. As macros são:

- `ESP_LOGE` (tag, formato, ...) - Registra um erro.
- `ESP_LOGW` (tag, formato, ...) - Registra um aviso.
- `ESP_LOGI` (tag, formato, ...) - Informações de registro.
- `ESP_LOGD` (tag, formato, ...) - Log de depuração.
- `ESP_LOGV` (tag, formato, ...) - Registra informações detalhadas.

Uma vez que o registro é incluído ou excluído em tempo de compilação, podemos especificar o nível de registro para incluir em nossas construções. Em tempo de compilação, isso pode excluir certas instruções de log do código. O sinalizador de compilação `-DLOG_LOCAL_LEVEL` controla os níveis de registro incluído.

Para as instruções de log que permanecem no código após a compilação que não foram excluídas em tempo de construção, podemos controlar o nível de log em tempo de execução chamando `esp_log_level_set()`. A assinatura desta função é:

```
void esp_log_level_set(const char * tag, nível esp_log_level_t)
```

A tag nomeia os grupos de registro que mostraremos. Se a tag especial de nome "`*`" for fornecido, corresponde a todas as tags.

Página 89

## Página 90

Se estivermos escrevendo rotinas de tratamento de interrupções, não use essas funções de registro dentro delas.

- [Erro: fonte de referência não encontrada](#)

### Registrando no UART1

Podemos inserir instruções de diagnóstico usando `os_pprintf()`. Isso faz com que o texto e os dados associados a essas funções sejam gravados no UART1 (GPIO 2). Se conectarmos um USB → Dispositivo UART no circuito, podemos então olhar para os dados registrados. No meu desenvolvimento ambiente Sempre tenho dois dispositivos USB → UART em jogo. Um para piscar novo aplicativos e um para usar como saída de diagnóstico.

O sistema operacional também é capaz de gravar informações de depuração. Por padrão, está ativado, mas pode ser desligado com uma chamada para `system_set_os_print()`.

Veja também:

- [Conversores USB para UART](#)
- [Trabalhando com serial](#)
- [system\\_set\\_os\\_print](#)

### Execute um Blinky

Olhando fisicamente para um ESP8266, não há muito para ver que diga que tudo está funcionando bem dentro dele. Há uma luz de energia e uma luz ativa de transmissão de rede ... mas isso é sobre isso. Uma técnica que eu recomendo é sempre fazer com que seu dispositivo execute um

"led piscando", comumente conhecido como "Blinky". Isso pode ser alcançado por conectar um pino GPIO a um resistor limitador de corrente e, em seguida, a um LED. Quando o O sinal GPIO fica alto, o LED acende. Quando o sinal GPIO fica baixo, o LED fica escuro. Se definirmos um retorno de chamada do temporizador que é chamado (por exemplo) uma vez por segundo e alterna o valor do sinal do pino GPIO a cada invocação, teremos um piscar simples CONDUZIU. Você ficará surpreso com a sensação boa que lhe dará simplesmente sabendo que *algo* está vivo dentro do dispositivo cada vez que você o vê piscar.

O custo de execução do temporizador e alteração do valor de I / O para obter um piscar deve não seria um problema durante o tempo de desenvolvimento, então não me preocuparia com os efeitos colaterais de Fazendo isso. Obviamente, para um aplicativo publicado, você pode não desejar isso e pode simplesmente removê-lo.

No entanto, embora este seja um circuito trivial, ele tem muitos usos durante o desenvolvimento. Primeiro, você sempre saberá que o dispositivo está funcionando. Se o LED estiver piscando, você sabe o dispositivo tem controle de processamento lógico e de energia. Se a luz parar de piscar, você saberá que algo travou ou você entrou em um loop infinito.

Outro propósito útil para incluir o Blinky é validar que você digitou o flash modo ao programar o dispositivo. Se entendermos que o dispositivo pode inicializar em

Página 90

## Página 91

modo normal ou flash e inicializamos no modo flash, então o Blinky irá parar executando. Isso pode ser útil se você estiver usando botões ou jumpers para alternar o boot modo, pois irá fornecer evidências de que você *não* está no modo normal. Na ocasião eu tenho pressionou mal alguns botões de controle e foi rapidamente capaz de perceber que algo estava errado antes mesmo de tentar piscá-lo, pois o Blinky ainda estava funcionando.

Aqui está um código simples para configurar um Blinky. Neste exemplo, usamos GPIO4 como o Driver de LED. Primeiro, o código que colocamos em user\_init :

```
PIN_FUNC_SELECT (PERIPHS_IO_MUX_GPIO4_U, FUNC_GPIO4);
os_timer_disarm (& blink_timer);
os_timer_setfn (& blink_timer, (os_timer_func_t *) blink_cb, (void *) 0);
os_timer_arm (& blink_timer, 1000, 1);
```

Isso pressupõe um global chamado blink\_timer definido como:

```
LOCAL os_timer_t blink_timer;
```

A função de retorno de chamada neste exemplo é chamada blink\_cb e se parece com:

```
LOCAL void ICACHE_FLASH_ATTR blink_cb (void * arg)
{
    led_state =! led_state;
    GPIO_OUTPUT_SET (4, led_state);
}
```

A variável global chamada led\_state contém o estado atual do LED (1 = ligado, 0 = desligado):

```
LOCAL uint8_t led_state = 0;
```

### Despejando endereços IP

Por ser um dispositivo WiFi e TCP / IP, você imaginaria que o ESP8266 funciona muito com Endereços IP e você estaria certo. Podemos gerar uma representação de string de um IP endereço usando:

```
os_printf (IPSTR, IP2STR (plpAddrVar))
```

a macro IPSTR é "% d.% d.% d.% d", então o acima é equivalente a:

```
os_printf ("%d.%d.%d.%d", IP2STR (plpAddrVar))
```

que pode ser mais útil em certas situações.

Veja também:

- [ipaddr\\_t](#)

### Manipulação de exceção

Em tempo de execução, as coisas nem sempre funcionam conforme o esperado e uma exceção pode ser lançada.

Por exemplo, você pode tentar acessar o armazenamento em um local inválido ou escrever para ler apenas memória ou realizar uma divisão por zero.

---

## Página 92

Quando tal ocorrência acontecer, o dispositivo se reinicializará, mas não antes de escrever alguns diagnósticos para UART1. O diagnóstico pode ser semelhante a:

Exceção fatal (28):

```
epc1 = 0x40243182, epc2 = 0x00000000, epc3 = 0x00000000, excvaddr = 0x00000050,
depc = 0x00000000
```

Os códigos são os seguintes:

- desculpa - Código que descreve a causa
- epc1 - contador de programa de exceção
- excvaddr - endereço virtual que causou a busca, carregamento ou armazenamento mais recente exceção. Por exemplo, se ocorrer uma gravação na memória e essa memória não for RAM será lançada uma exceção e o valor aqui será o endereço que foi tentou ser escrito.

Os códigos de exceção principais são:

Código	Código	Nome da causa
0	0x00	IllegalInstructionCause
1	0x01	SyscallCause
2	0x02	InstructionFetchErrorCause
3	0x03	LoadStoreErrorCause
4	0x04	Level1InterruptCause
5	0x05	Allocacause
6	0x06	IntegerDivideByZeroCause
7	0x07	Reservado
8	0x08	PrivilegedCause
9	0x09	LoadStoreAlignmentCause
10	0x0a	Reservado
11	0x0b	Reservado
12	0x0c	InstrPIFDataErrorCause
13	0x0d	LoadStorePIFDataErrorCause
14	0x0e	InstrPIFAddrErrorCause
15	0x0f	LoadStorePIFAddrErrorCause
16	0x10	InstTLBMissCause
17	0x11	InstTLBMultiHitCause
18	0x12	InstFetchPrivilegeCause
19	0x13	Reservado
20	0x14	InstFetchProhibitedCause

---

**Página 93**

```

21    0x15      Reservado
22    0x16      Reservado
23    0x17      Reservado
24    0x18      LoadStoreTLBMissCause
25    0x19      LoadStoreTLBMultiHitCause
26    0x1a      LoadStorePrivilegeCause
27    0x1b      Reservado
28    0x1c      LoadProhibitedCause
29    0x1d      StoreProhibitedCause
30    0x1e      Reservado
31    0x1f      Reservado
32-39 0x20-0x27 CoprocessornDisabled
40-63 0x28-0x3f Reservado

```

Se soubermos a localização da exceção, podemos analisar o executável ( app.out ) para descobrir qual parte do código causou o problema. Por exemplo:

```
xtensa-lx106-elf-objdump -x app.out -d
```

#### Usando um depurador (GDB)

GDB é o GNU Debugger e é uma excelente ferramenta para depurar código C compilado código. No entanto, é projetado principalmente para depurar aplicativos hospedados no sistema operacional, como aqueles compilado para Windows ou Linux e não tinha muita aplicabilidade para o ESP8266. Esta foi até o Espressif lançar seu esboço GDB.

A versão do depurador deve ser a ferramenta xtensa-lx106-elf-gdb .

Para se preparar para usar a ferramenta, é necessário compilar com os seguintes sinalizadores adicionais:

- -ggdb
- -Og

Além disso, devemos inicializar o GDB com uma chamada para `gdbstub_init()` em algum lugar no início do código de inicialização em nosso aplicativo. Por fim, vinculamos a biblioteca `gdbstub` .

Veja também:

- Github: [espressif / esp-gdbstub](https://github.com/espressif/esp-gdbstub)

#### Depuração e teste de conexões TCP e UDP

Ao trabalhar com TCP / IP, você provavelmente desejará ter alguns aplicativos que possa usar para enviar e receber dados para que você possa ter certeza de que o ESP8266 está funcionando. Lá

---

**Página 94**

há uma série de excelentes ferramentas e utilitários disponíveis e variam de acordo com a plataforma e função.

#### Android - protocolo de soquete

O protocolo de soquete é um aplicativo Android gratuito disponível na loja de aplicativos do Google Play.

Ver:

- <https://play.google.com/store/apps/details?id=aprisco.app.android>

#### Android - remetente / receptor UDP

O UDP Sender / Receiver é outro aplicativo Android gratuito disponível no Google play loja de aplicativos. O que torna este interessante é sua capacidade de ser um UDP (em oposição a TCP) emissor e receptor. Ver:

- <https://play.google.com/store/apps/details?id=com.jca.udpsendreceive>

#### Windows - Hercules

Hercules é um aplicativo mais antigo para Windows que ainda parece funcionar muito bem com os mais recentes lançamentos. Parece um pouco velho no dente agora, mas ainda parece funcionar muito bem.

Ver:

- [http://www.hw-group.com/products/hercules/index\\_en.html](http://www.hw-group.com/products/hercules/index_en.html)

#### Ondulação

Curl é uma ferramenta de linha de comando poderosa e abrangente para executar todo e qualquer URL comandos relacionados. Ele pode transmitir solicitações HTTP de todos os formatos diferentes e receber suas respostas. Ele tem um conjunto desconcertante de parâmetros disponíveis, que é ao mesmo tempo um bêncão e maldição. Você pode ter certeza de que, se isso pode ser feito, Curl também ... no entanto, esteja preparado para examinar muita documentação.

Veja também:

- Ondulação

#### Eclipse - TCP / MON

Uma das ferramentas mais poderosas e úteis disponíveis é chamada Monitor TCP / IP que é fornecido como parte do Eclipse e distribuído com o "Eclipse Web Developer Tools".

O monitor TCP / IP é aberto por meio da visualização do Eclipse chamada "Monitor TCP / IP".

Se você não conseguir encontrá-lo no visor, é provável que você não tenha instalado "Eclipse Web Developer Tools". Uma vez iniciado, abra seu painel de propriedades:

A partir daí, você pode adicionar ouvintes locais. Estes serão ouvintes TCP / IP que ouvirão em um porta local onde o Eclipse está sendo executado. Durante a configuração, você especifica outro IP endereço e número da porta. Quando o tráfego TCP chega ao ouvinte no qual o TCP / IP

Página 95

---

## Página 96

O monitor está observando, ele encaminhará esse tráfego para o parceiro ao mesmo tempo registrando-o na tela TCP / IP Monitor.

Por exemplo, aqui o monitor TCP / IP está escutando em 192.168.1.2 (localhost) que é onde Eclipse está em execução. Ele está escutando na porta 9999. Quando o tráfego TCP / IP chega a esse

endereço, ele será enviado para 192.168.1.17 (que passa a ser o meu ESP8266 dispositivo) para a porta 80.

Aqui está um exemplo de log que vi ao enviar uma solicitação do navegador:

Como você pode ver, as informações capturadas aqui são poderosas. Podemos ver cada tráfego solicitação, seu conteúdo e cabeçalhos HTTP.

Página 96

## Página 97

<httpbin.org>

Ao testar os protocolos HTTP, conectando-se ao site em <http://httpbin.org> pode ser inestimável. Ele fornece uma série de serviços para testar solicitações HTTP.

### Arquitetura ESP8266

Para começar a pensar em escrever aplicativos para o ESP8266, precisamos entender o arquitetura de alto nível do dispositivo.

#### Programas personalizados

Programas personalizados são aplicativos que você pode escrever e são o foco principal deste livro. Esses programas podem ser escritos em C ou C ++ e então compilados no binário arquivos. Espera-se que os programas tenham funções "bem conhecidas" definidas dentro desse servem como pontos de entrada arquitetados e retornos de chamada.

Os programadores escrevem um arquivo de linguagem C com um nome sugerido de " user\_main.c ". Contida dentro está uma função com a assinatura:

```
void user_init (void)
```

Isso fornece a entrada inicial no código do aplicativo. É chamado uma vez durante a inicialização. Ao executar dentro desta função, perceba que nem todo o ambiente ainda está operacional. Se você precisa de um ambiente totalmente funcional, registre uma função de retorno de chamada que será chamado quando o ambiente estiver 100% pronto. Esta função de retorno de chamada pode ser registrado com uma chamada para `system_init_done_cb`.

A inicialização de RF também deve ser fornecida por meio de:

```
void user_rf_pre_init (void)
```

Ao executar no código do usuário, precisamos estar cientes de que o objetivo principal do dispositivo é comunicações de rede. Uma vez que estes são tratados no software, quando o usuário o código obtém controle, o que significa simplesmente que o código de rede não. Uma vez que só temos um fio de controle, não podemos estar em dois lugares ao mesmo tempo. A duração recomendada para o gasto em código de usuário em uma única sessão é inferior a 10 mseg.

Veja também:

- [system\\_init\\_done\\_cb](#)

### **WiFi na inicialização**

O ESP8266 armazena informações de inicialização do WiFi na memória flash. Isso permite que executar suas funções na inicialização sem ter que pedir ao usuário qualquer especial ou informação adicional. Em minha opinião, isso é mais problemático do que vale a pena. Se eu vou para escrever um aplicativo ESP8266, quero controlar quando, como e a que ele se conectará

Página 97

## **Página 98**

ou ser um ponto de acesso. Felizmente, existe uma função chamada `wifi_station_set_auto_connect()` e seu parceiro chamado `wifi_station_get_auto_connect()`. Isso nos permite substituir a conexão automática funciona quando somos uma estação.

### **Trabalhando com WiFi - ESP8266**

O ESP8266 pode ser uma estação na rede, um ponto de acesso para outros dispositivos ou ambos. Esta é uma consideração fundamental e queremos escolher como o dispositivo se comporta logo no início de nosso design. Depois de escolher o que queremos, definimos um global propriedade mode que indica qual dos modos operacionais nosso dispositivo irá executar (estação, ponto de acesso ou estação E ponto de acesso).

#### **Procurando pontos de acesso**

Se o ESP8266 estiver desempenhando o papel de uma estação, precisaremos conectar a um acesso apontar. Podemos solicitar uma lista dos pontos de acesso disponíveis contra os quais podemos tentar se conectar. Fazemos isso usando a função `wifi_station_scan()`. Esta função recebe um ponteiro de função de retorno de chamada como um de seus parâmetros. Este retorno de chamada será invocado quando a varredura for concluída. O retorno de chamada é necessário porque pode demorar algum tempo (alguns segundos) para a verificação ser realizada e não podemos bloquear a operação do sistema como um todo até ser concluído. A função de retorno de chamada de varredura recebe um link lista de estruturas BSS. Contidos em uma estrutura BSS estão:

- O SSID para a rede
- O BSSID para o ponto de acesso
- O canal
- A força do sinal
- ... outras

Por exemplo:

```
void scanCB (void * arg, status STATUS) {
    struct bss_info * bssInfo;
    bssInfo = (struct bss_info *) arg;
    // pula o primeiro na cadeia ... é inválido
    bssInfo = STAILQ_NEXT (bssInfo, próximo);
```

```

        while (bssInfo != NULL) {
            os_pprintf ("ssid:% s \ n", bssInfo->ssid);
            bssInfo = STAILQ_NEXT (bssInfo, próximo);
        }
    }

// ...

```

Página 98

## Página 99

```

{
    // Certifique-se de que estamos no modo de estação
    wifi_set_opmode_current (STATION_MODE);

    // Solicita uma varredura da rede chamando "scanCB" na conclusão
    wifi_station_scan (NULL, scanCB);
}

```

Observe o uso da macro STAILQ\_NEXT () para navegar para a próxima entrada na lista. O fim da lista é indicado quando retorna NULL .

Veja também:

- [Amostra - Scanner WiFi](#)
- [struct bss\\_info](#)
- [STATUS](#)

### Definindo o modo de operação

O ESP8266 pode ser executado como uma estação WiFi, um ponto de acesso WiFi ou uma estação e um ponto de acesso. Estes são considerados os três modos de operação globais possíveis. O modo de operação que é usado quando o dispositivo inicializa é retido na memória flash, mas pode ser alterado com uma chamada para wifi\_set\_opmode () . Isso também mudará o modo atual como registro o modo a ser usado na próxima reinicialização. Para simplesmente mudar o modo sem mudando o próximo modo de inicialização, podemos usar wifi\_set\_opmode\_current () . Para recuperar o modo atual, podemos usar wifi\_get\_opmode () e recuperar o modo usado na inicialização, podemos usar wifi\_get\_opmode\_default () . Bem porque temos a opção de alterar o modo atual sem salvá-lo na memória flash é um mistério. Presumivelmente, há algum ocasião em que tal recurso era necessário e, portanto, exposto, mas seja qual for o motivo pode ser não é óbvio.

Veja também:

- [wifi\\_get\\_opmode](#)
- [wifi\\_get\\_opmode\\_default](#)

### Tratamento de eventos WiFi

Durante a operação como um dispositivo WiFi, certos eventos podem ocorrer que ESP8266 precisa saber sobre. Estes podem ser importantes ou interessantes para o aplicativos em execução nele. Já que não sabemos quando, ou mesmo se, algum evento irá acontecer, não podemos ter nosso bloco de aplicativos esperando que eles ocorram. Em vez disso, o que nós deve fazer é definir uma função de retorno de chamada que será invocada se um evento realmente ocorrer. A função chamada wifi\_set\_event\_handler\_cb () faz exatamente isso. Registra um função que será chamada quando o ESP8266 detectar certos tipos de wi-fi relacionados eventos. A função registrada é chamada e passada uma rica estrutura de dados que inclui o tipo de evento e os dados associados correspondentes a esse evento. Os tipos de eventos que causam o retorno de chamada são:

---

## Página 100

- Conectamos a um ponto de acesso
- Nós nos desconectamos de um ponto de acesso
- O modo de autorização mudou
- Temos um endereço IP emitido por DHCP
- Uma estação conectada a nós quando estamos no modo Access Point
- Uma estação desconectada de nós quando estamos no modo Access Point

Aqui está um exemplo de uma função de manipulador de eventos que simplesmente registra o nome do evento que foi visto:

```
void eventHandler (System_Event_t * event) {switch (event-> event) {
    case EVENT_STAMODE_CONNECTED:
        os_printf ("Evento: EVENT_STAMODE_CONNECTED \ n");
        pausa;
    case EVENT_STAMODE_DISCONNECTED:
        os_printf ("Evento: EVENT_STAMODE_DISCONNECTED \ n");
        pausa;
    case EVENT_STAMODE_AUTHMODE_CHANGE:
        os_printf ("Evento: EVENT_STAMODE_AUTHMODE_CHANGE \ n");
        pausa;
    case EVENT_STAMODE_GOT_IP:
        os_printf ("Evento: EVENT_STAMODE_GOT_IP \ n");
        pausa;
    case EVENT_SOFTAPMODE_STACONNECTED:
        os_printf ("Evento: EVENT_SOFTAPMODE_STACONNECTED \ n");
        pausa;
    case EVENT_SOFTAPMODE_STADISCONNECTED:
        os_printf ("Evento: EVENT_SOFTAPMODE_STADISCONNECTED \ n");
        pausa;
    padrão:
        os_printf ("Evento inesperado: % d \ n", evento-> evento);
        pausa;
    }
}
```

A função de retorno de chamada pode ser registrada em `user_init()` da seguinte forma:

```
wifi_set_event_handler_cb (eventHandler);
```

Estamos limitados ao que devemos fazer em um retorno de chamada do manipulador de eventos. Especificamente, parece que não devemos tentar formar novas conexões. Em vez disso, devemos postar um tarefa que agora podemos fazer um trabalho adicional.

Veja também:

- [System\\_Event\\_t](#)

---

## Página 101

### Configuração da estação

Quando pensamos em um ESP8266 como uma estação WiFi, percebemos que, a qualquer momento, só pode ser conectado a um ponto de acesso. Colocando de outra forma, não há significado

em dizer que o dispositivo está conectado a **dois** ou mais pontos de acesso ao mesmo tempo. A identidade do ponto de acesso ao qual desejamos ser associados é conhecido como o "station\_config" e é modelado como a estrutura C chamada " struct station\_config ". Contidos nessa estrutura estão dois campos muito importantes chamados " ssid " e " senha ". O campo ssid é o SSID do ponto de acesso ao qual nos conectaremos. O campo da senha é o valor de texto não criptografado da senha que será usado para autenticar nosso dispositivo no ponto de acesso de destino para permitir a conexão.

Quando inicializado, o ESP8266 lembra o último station\_config que definimos. Podemos definir explicitamente os dados station\_config usando a função wifi\_station\_set\_config () .

Isso definirá a configuração atual e salvará para recuperação posterior após uma reinicialização. Se nós desejamos apenas definir a configuração da estação atual e **não** ter a informação persistente, nós podemos usar o wifi\_station\_set\_config\_current () .

Não devemos tentar realizar nenhuma operação WiFi até que o dispositivo esteja totalmente inicializado. Sabemos que somos inicializados registrando um retorno de chamada usando o system\_init\_done\_cb () função.

Por exemplo:

```
void initDone () {
    wifi_set_opmode_current(STATION_MODE);
    struct station_config stationConfig;
    strcpy(stationConfig.ssid, "myssid", 32);
    strcpy(stationConfig.password, "mypassword", 64);
    wifi_station_set_config(& stationConfig);
}
```

Veja também:

- [system\\_init\\_done\\_cb](#)
- [wifi\\_station\\_get\\_config\\_default](#)
- [wifi\\_station\\_set\\_config\\_current](#)
- [station\\_config](#)

### Conectando a um ponto de acesso

Uma vez que o ESP8266 foi configurado com os detalhes de configuração da estação, que inclui o SSID e a senha, estamos prontos para realizar uma conexão com o acesso de destino apontar. A função wifi\_station\_connect () formará a conexão. Perceba que isso não é instantâneo e você não deve presumir que imediatamente após este comando que você está conectado. Nada nos blocos ESP8266 e, como tal, nem a chamada para esta função. Algum tempo depois, estaremos realmente conectados. Veremos dois eventos de retorno de chamada disparados. O primeiro é EVENT\_STAMODE\_CONNECTED, indicando que temos

conectado ao ponto de acesso. O segundo evento é EVENT\_STAMODE\_GOT\_IP que indica que um endereço IP foi atribuído a nós pelo servidor DHCP. Só nesse ponto podemos realmente participar nas comunicações. Se estivermos usando endereços IP estáticos para nosso dispositivo, então veremos apenas o evento conectado.

Há mais uma consideração associada à conexão com pontos de acesso e que é a ideia de conexão automática. Existe um sinalizador booleano armazenado em flash que indica se o ESP8266 deve ou não tentar se conectar automaticamente ao último ponto de acesso usado. Se definido como verdadeiro, depois que o dispositivo for iniciado e sem você tendo que codificar quaisquer chamadas de API, ele tentará se conectar ao último ponto de acesso usado. É uma comodidade que prefiro desligar. Normalmente, eu quero o controle do meu dispositivo para determinar quando eu conecto. Podemos ativar ou desativar o recurso de conexão automática por fazendo uma chamada para wifi\_station\_set\_auto\_connect () .

Veja também:

- [Tratamento de eventos WiFi](#)

### **Controle e fluxos de dados ao conectar como uma estação**

Estamos agora no estágio em que podemos desenhar um fluxo sequencial das partes. Algum funções que você é responsável e deve fornecer, incluindo:

- user\_init - ponto de entrada no aplicativo
- initDoneCB - Retorno de chamada quando a inicialização for concluída
- eventCB - Callback quando um evento relacionado ao WiFi é detectado

As outras funções são responsáveis por chamar. Vamos considerar esta parte do sequência concluída quando temos uma indicação de que temos um endereço IP válido.

Página 102

## **Página 103**

### **Ser um ponto de acesso**

Até agora, consideramos apenas o ESP8266 como uma estação WiFi para um acesso existente ponto, mas também tem a capacidade de **ser** um ponto de acesso para outros dispositivos WiFi (estações) incluindo outros ESP8266s.

Para ser um ponto de acesso, precisamos definir o SSID que permite outros dispositivos para distinguir nossa rede. Este SSID pode ser sinalizado como oculto, se não desejarmos para ser verificado. Além disso, também teremos que fornecer o modo de autenticação que será usado quando uma estação deseja se conectar a nós. Isso é usado para permitir estações e proibir as não autorizadas. Somente estações que sabem nossa senha irão ter permissão para se conectar. Se estivermos usando autenticação, também teremos que escolher uma senha que as estações conectadas terão que saber e fornecer para conectar.

A primeira tarefa em ser um ponto de acesso é sinalizar o ESP8266 como tal usando o funções `wifi_set_opmode()` ou `wifi_set_opmode_current()` e passar a sinalização de que solicita que sejamos um ponto de acesso dedicado ou um ponto de acesso e uma estação.

Aqui está um trecho de código que pode ser usado para configurar o ESP8266 como um ponto de acesso:

```
// Defina nosso modo como um ponto de acesso
wifi_set_opmode_current(SOFTAP_MODE);

// Construir nossos detalhes de configuração do ponto de acesso
os_strcpy(config.ssid, "ESP8266");
```

```
os_strcpy (config.password, "senha");
config.ssid_len = 0;
config.authmode = AUTH_OPEN;
config.ssid_hidden = 0;
config.max_connection = 4;
wifi_softap_set_config_current (& config);
```

Quando uma estação remota se conecta ao ESP8266 como um ponto de acesso, veremos um mensagem de depuração gravada em UART1 que pode ser semelhante a:

estação: f0: 25: b7: ff: 12: junção c5, AID = 1

Contém o endereço MAC da nova estação que se conecta à rede. Quando a estação desconectar, veremos uma mensagem de log de depuração correspondente que pode ser:

estação: f0: 25: b7: ff: 12: c5 licença, AID = 1

De dentro do ESP8266, podemos determinar quantas estações estão conectado com uma chamada para `wifi_softap_get_station_num()`. Se quisermos encontrar os detalhes dessas estações, podemos chamar `wifi_softap_get_station_info()` que retornará um lista vinculada de `struct station_info`. Temos que liberar explicitamente o armazenamento alocado por esta chamada com uma invocação de `wifi_softap_free_station_info()`.

Aqui está um exemplo de um snippet de código que lista os detalhes das estações conectadas:

Página 103

## Página 104

```
uint8 stationCount = wifi_softap_get_station_num ();
os_printf ("stationCount =% d \ n", stationCount);
struct station_info * stationInfo = wifi_softap_get_station_info ();
if (stationInfo != NULL) {
    while (stationInfo != NULL) {
        os_printf ("IP da estação:% d.% d.% d.% d \ n", IP2STR (& (stationInfo-> ip)));
        stationInfo = STAILQ_NEXT (stationInfo, próximo);
    }
    wifi_softap_free_station_info ();
}
```

Quando um ESP8266 atua como um ponto de acesso, isso permite que outros dispositivos se conectem a ele e formar uma conexão sem fio. No entanto, parece que dois dispositivos conectados ao mesmo ESP8266 atuando como um ponto de acesso não pode se comunicar diretamente entre cada um outro. Por exemplo, imagine dois dispositivos conectando-se a um ESP8266 como um acesso apontar. Eles podem ser alocados nos endereços IP 192.168.4.2 e 192.168.4.3. Nós pode imaginar que 192.168.4.2 poderia executar ping em 192.168.4.3 e vice-versa, mas isso não é permitido. Parece que eles apenas a conexão direta de rede permitida é entre os estações recém-conectadas e o próprio ponto de acesso (o ESP8266).

Isso parece limitar a aplicabilidade do ESP8266 como um ponto de acesso. O primário a intenção do ESP8266 como um ponto de acesso é permitir que dispositivos móveis (por exemplo, seu telefone) conecte-se ao ESP8266 e converse com um aplicativo executado nele.

Veja também:

- [wifi\\_softap\\_set\\_config\\_current](#)
- [wifi\\_softap\\_get\\_station\\_num](#)

### O servidor DHCP

Quando o ESP8266 está desempenhando a função de um ponto de acesso, é provável que você deseja que ele também se comporte como um servidor DHCP para que as estações conectadas possam ser endereços IP atribuídos automaticamente e aprender suas máscaras de sub-rede e gateways.

O servidor DHCP pode ser iniciado e interrompido dentro do dispositivo usando as APIs chamadas

**Redefinindo o intervalo padrão do servidor DHCP** A situação atualizado com uma chamada para `wifi_softap_dhcps_status()`.

O intervalo padrão de endereços IP oferecidos pelo servidor DHCP é 192.168.4.1 ou mais. O primeiro endereço é atribuído ao próprio ESP8266. É importante perceber que este intervalo de endereços **não** é o mesmo intervalo de endereços de sua LAN onde você pode estar trabalhando. O ESP8266 formou seu próprio espaço de endereço de rede e embora eles podem aparecer com os mesmos tipos de números (192.168.xx), eles são isolados e redes independentes. Se você iniciar um ponto de acesso no ESP8266 e se conectar a ele

Página 104

## Página 105

do seu telefone, não se surpreenda ao tentar fazer o ping da sua Internet PC conectado e não obtiver resposta.

Veja também:

- [wifi\\_softap\\_dhcps\\_stop](#)

### Endereço IP atual, máscara de rede e gateway

Se precisarmos, podemos consultar o ambiente do sistema operacional para o endereço IP atual, máscara de rede e gateway. Os valores destes são normalmente definidos para nós por um servidor DHCP quando nos conectamos a um ponto de acesso. A função chamada `wifi_get_ip_info()` retorna nossas informações atuais, enquanto a função chamada `wifi_set_ip_info()` nos permite definir nossos endereços.

Quando nos conectamos a um ponto de acesso e escolhemos usar DHCP, quando estamos alocado um endereço IP, é gerado um evento que pode ser usado como uma indicação de que agora temos um endereço IP válido.

Para configurar corretamente os endereços IP estáticos, no retorno de chamada `init_done`, chame `wifi_station_dhcpc_stop()` para desabilitar o cliente DHCP rodando no ESP8266. Após esta chamada `wifi_station_connect()` para iniciar a fase de conexão do ponto de acesso. Quando o chega o evento que indica que estamos conectados a um ponto de acesso como uma estação (`EVENT_STAMODE_CONNECTED`), podemos chamar `wifi_set_ip_info()` e passar o IP endereço, gateway e máscara de rede que desejamos usar. Observe que quando usamos um IP estático endereço, não receberemos o evento de retorno de chamada que indica que recebemos um IP endereço (`EVENT_STAMODE_GOT_IP`) como já o temos.

Veja também:

- [Tratamento de eventos WiFi](#)
- [wifi\\_station\\_dhcpc\\_stop](#)
- [struct\\_ip\\_info](#)

### Configuração protegida de WiFi - WPS

O ESP8266 é compatível com WiFi Protected Setup no modo de estação. Isso significa que se o ponto de acesso suporta, o ESP8266 pode se conectar ao ponto de acesso sem apresentando uma senha. Atualmente, apenas o "modo de botão" de conexão é implementado. Usando este mecanismo, um botão físico é pressionado no ponto de acesso e, por um período de dois minutos, qualquer estação dentro do alcance pode se conectar à rede usando o Protocolos WPS. Um exemplo de uso seria o botão WPS do ponto de acesso sendo pressionado e, em seguida, o dispositivo ESP8266 chamando `wifi_wps_enable()` e, em seguida, `wifi_wps_start()`. O ESP8266 então se conectararia à rede.

Veja também:

- [wifi\\_wps\\_enable](#)

---

## Página 106

- [wifi\\_wps\\_start](#)
- [wifi\\_set\\_wps\\_cb](#)
- [Perguntas simples: O que é WPS \(WiFi Protected Setup\)](#)
- Wikipedia: [Configuração de Wi-Fi protegido](#)

### Trabalhando com TCP / IP

TCP / IP é o protocolo de rede usado na Internet. É o protocolo que o ESP8266 nativamente entende e usa com WiFi como o transporte. Livros sobre livros já foram escritos sobre TCP / IP e nosso objetivo não é tentar reproduzir um discussão detalhada de como funciona, no entanto, existem alguns conceitos que vamos tentar e captura.

Primeiro, existe o endereço IP. Este é um valor de 32 bits e deve ser exclusivo para cada dispositivo Conectado a internet. Um valor de 32 bits pode ser considerado como quatro valores distintos de 8 bits ( $4 \times 8 = 32$ ). Uma vez que podemos representar um número de 8 bits como um valor decimal entre 0 e 255, geralmente representamos endereços IP com a notação <número>. <número>. <número>. <número> por exemplo 173.194.64.102. Estes IP endereços não são comumente inseridos em aplicativos. Em vez disso, um nome textual é digitado como "google.com" ... mas não se deixe enganar, esses nomes são uma ilusão no TCP / IP nível. Todo o trabalho é realizado com endereços IP de 32 bits. Existe um sistema de mapeamento que pega um nome (como "google.com") e recupera seu endereço IP correspondente. O a tecnologia que faz isso é chamada de "Sistema de Nomes de Domínio" ou DNS.

Quando pensamos em TCP / IP, existem três protocolos distintos em jogo aqui. O primeiro é o IP (Internet Protocol). Esta é a passagem de datagrama da camada de transporte subjacente protocolo. Acima da camada IP está o TCP (Transmission Control Protocol), que fornece o ilusão de uma conexão através do protocolo IP sem conexão. Finalmente, há UDP (usuário Protocolo de Datagrama). Este também vive acima do protocolo IP e fornece datagramas (sem conexão) transmissão entre aplicativos. Quando dizemos TCP / IP, estamos **não** apenas falando sobre TCP rodando sobre IP, mas na verdade estão usando isso como uma abreviação para o protocolos principais que são IP, TCP e UDP e nível de aplicativo relacionado adicional protocolos como DNS, HTTP, FTP, Telnet e mais.

### A arquitetura espconn

Como não temos permissão para bloquear o controle no ESP8266 por qualquer período de tempo, nós deve registrar funções de retorno de chamada que serão invocadas quando alguma ação de longa duração foi concluído ou ocorre um evento assíncrono. Por exemplo, quando desejamos receber uma conexão de rede de entrada, não podemos simplesmente esperar por essa conexão para chegar. Em vez disso, registramos uma função de retorno de chamada de conexão e, em seguida, retornamos o controle para o sistema operacional. Quando a conexão eventualmente chegar no futuro, a função de retorno de chamada que registramos anteriormente é invocado em nosso nome.

A tabela a seguir lista as funções de retorno de chamada que o ESP8266 oferece suporte Conexões e eventos TCP.

Função de registro	Ligue de volta	Descrição
espconn_register_connectcb	espconn_connect_callback	TCP conectado com sucesso
espconn_register_disconnectcb	espconn_disconnect_callback	TCP desconectado com sucesso
espconn_register_reconcb	espconn_reconnect_callback	Erro detectado ou TCP desconectado
espconn_register_sentcb	espconn_sent_callback	Dados TCP ou UDP enviados
espconn_register_recvcb	espconn_recv_callback	Dados recebidos de TCP ou UDP
espconn_register_write_finish	espconn_write_finish_callback	Gravar dados no TCP-send-buffer

Veja também:

- [espconn\\_register\\_connectcb](#)
- [espconn\\_register\\_disconnectcb](#)
- [espconn\\_register\\_reconcb](#)
- [espconn\\_register\\_sentcb](#)
- [espconn\\_register\\_recvcb](#)
- [espconn\\_register\\_write\\_finish](#)

## TCP

Uma conexão TCP é um tubo bidirecional através do qual os dados podem fluir em ambas as direções. Antes de a conexão ser estabelecida, um lado está atuando como servidor. É passivamente ouvindo as solicitações de conexão de entrada. Ele simplesmente ficará lá pelo tempo que for necessário até que uma solicitação de conexão chegue. O outro lado da conexão é responsável por iniciar a conexão e solicitar ativamente que uma conexão seja formada. Uma vez que a conexão foi construída, ambos os lados podem enviar e receber dados. Em ordem para o "cliente" para solicitar uma conexão, ele deve saber as informações de endereço em que o servidor está escutando. Este endereço é composto por duas partes distintas. A primeira parte é o endereço IP do servidor e a segunda parte é o "número da porta" para o específico ouvinte. Se pensarmos em um PC, você pode ter muitos aplicativos nele, cada um dos quais pode receber uma conexão de entrada. Apenas saber o endereço IP do seu PC não é suficiente para endereçar uma conexão ao aplicativo correto. A combinação de IP e endereço mais o número da porta fornecem todo o endereçamento necessário.

Como analogia, pense no seu telefone celular. É sentado passivamente lá até que alguém chama isso. Em nossa história, é o ouvinte. O endereço que alguém usa para formar uma conexão é o seu número de telefone, que é composto por um código de área mais o restante. Por exemplo, um número de telefone (817) 555-1234 alcançará um determinado telefone. No entanto, o código de área 817 é para Fort Worth, no Texas ... chamando isso por si só não é suficiente para entrar em contato com um indivíduo ... o número de telefone completo é necessário.

Não, veremos como um ESP8266 pode se configurar como um ouvinte de um Conexão TCP / IP.

---

## Página 108

Começamos apresentando uma estrutura de dados absolutamente vital que é chamada de " struct espconn ". Esta estrutura de dados contém muito do "estado" de nossa conexão e é passada para a maioria de nossas APIs TCP.

Nós o inicializamos definindo vários de seus campos:

- tipo - Este é o tipo de conexão que vamos usar. Já que queremos usar uma conexão TCP em oposição a uma conexão UDP, fornecemos ESPCONN\_TCP como O valor que.

- estado - O estado da conexão mudará com o tempo, mas nós o inicializamos para tem um estado inicial vazio fornecendo ESPCONN\_NONE .

Por exemplo:

```
struct espconn conn1;

void init () {
    conn1.type = ESPCONN_TCP;
    conn1.state = ESPCONN_NONE;
}
```

Agora apresentamos outra estrutura chamada " esp\_tcp ". Esta estrutura contém TCP configurações específicas. Para nossa história, é aqui que fornecemos o número da porta que nosso TCP conexão ouvirá as conexões do cliente. Isso é fornecido na propriedade chamada " local\_port ".

```
esp_tcp tcp1;

void init () {
    tcp1.local_port = 25867;
}
```

Dentro do tipo de dados struct espconn , há um campo chamado " proto " que é um ponteiro para uma estrutura de dados específica do protocolo. Para uma conexão TCP, este será um ponteiro para um instância " esp\_tcp " ... e é aqui que colamos a história. O código completo torna-se:

```
struct espconn conn1;
esp_tcp tcp1;

void init () {
    tcp1.local_port = 25867;
    conn1.type = ESPCONN_TCP;
    conn1.state = ESPCONN_NONE;
    conn1.proto.tcp = & tcp1;
}
```

Podemos agora iniciar nosso servidor ouvindo conexões TCP de entrada usando espconn\_accept () . Isso leva a estrutura espconn como entrada que é usada para indicar qual porta devemos escutar (entre outras coisas). Aqui está um exemplo:

Página 108

## Página 109

```
espconn_accept (& conn1);
```

Depois de chamar isso, o ESP8266 agora estará ouvindo passivamente o TCP de entrada conexões na porta especificada no campo local\_port . É importante notar que seu código não bloqueia a espera por uma solicitação de entrada. Em algum lugar no coração de o ESP8266 agora sabe aceitar conexões nessa porta. A próxima pergunta é um simples ... o que acontece quando uma conexão eventualmente chega?

A resposta para isso faz parte da arquitetura central do dispositivo e gira em torno do noção de callbacks. Em seu próprio código de aplicativo, é sua responsabilidade registrar um função de retorno de chamada que será invocada quando a conexão chegar. É aqui que o A função espconn\_regist\_connectb () entra em ação . Esta função registra um usuário função de retorno de chamada fornecida que será chamada quando uma conexão chegar.

```
void connectCB (void * arg) {
    struct espconn * pNewEspConn = (struct espconn *) arg;
    ... Faça algo com a nova conexão
}
```

```

...  

espconn_register_connectcb (& conn1, connectCB);  

espconn_accept (& conn1);
}

```

Visto como um diagrama de fluxo de sequência, podemos ver as relações entre alguns dos componentes. Assumimos que, no caso de retorno de chamada, quando nos foi atribuído um IP endereço, então registramos que estamos interessados em conexões e que estamos dispostos para aceitar novas conexões de entrada. Então, em algum momento no futuro, receberemos um nova solicitação de conexão e o retorno de chamada da conexão é invocado.

O conteúdo da estrutura espconn passada no retorno de chamada incluirá o IP remoto endereço do parceiro que nos conectou. Podemos usar essa informação para registrar ou para autorização. Por exemplo, se o endereço IP não for aquele que desejamos permitir, podemos desconecte neste ponto usando `espconn_disconnect()`. Perceba que esta estrutura de dados representa a **nova** conexão com o parceiro que acabou de invocar e **não** é o mesmo como struct espconn que foi usado para registrar que queríamos aceitar novas conexões. Uma nova estrutura espconn será passada para cada nova conexão formada.

Página 109

## Página 110

Isso cobre o ESP8266 que recebe solicitações de conexão de entrada, mas e se deveria deseja formar uma conexão de saída para um aplicativo TCP remoto? Para realizar um solicitação de conexão de saída, podemos usar a chamada `espconn_connect()`. Antes de fazer esta chamada, devemos configurar a estrutura TCP. O campo `remote_port` deve conter a porta número do parceiro de aplicativo ao qual desejamos nos conectar. Além do

O campo `remote_ip` deve conter o endereço IP da máquina que hospeda o parceiro. O `local_port` deve ser atribuído a uma porta local não usada usando `espconn_port()`. O `local_ip` também deve ser preenchido usando o endereço IP local. Assim como receber um conexão de entrada, fazer uma conexão de saída resultará em uma chamada para o retorno de chamada de conexão quando a conexão é estabelecida. Assim que a conexão for formado, mais uma vez, as duas extremidades da conexão serão pares uma da outra. Isto é **vital** para perceber que apenas emitir um `espconn_connect()` faz **não** resultar em um imediato conexão. Em vez disso, somente após o ConnectCB ter sido recebido, podemos realmente usar a conexão.

Por exemplo:

```

struct espconn conn1;
esp_tcp tcp1;

void init () {
    tcp1.remote_port = 25867;
    tcp1.remote_ip = ipAddress;
    tcp1.local_port = espconn_port ();
    struct ip_info ipconfig;
    wifi_get_ip_info (STATION_IF, & ipconfig);
    os_memcpy (tcp.local_ip, & ipconfig.ip, 4);

    conn1.type = ESPCONN_TCP;
    conn1.state = ESPCONN_NONE;
}

```

```
conn1.proto.tcp = & tcp1;
}
```

Se o parceiro de nossa conversa fechar a conexão, seremos informados de que através da função registramos com `espconn_regist_disconcb()`. O campo do estado da estrutura `espconn` conterá `CLOSE`. Detectar o desligamento normal de um parceiro nos permite realizar a lógica de que podemos precisar, como liberar recursos ou persistir dados.

Se uma conexão TCP for formada e nenhum tráfego fluir pela conexão por pelo menos 10 segundos (padrão), então a conexão é fechada automaticamente a partir da extremidade ESP8266. A propriedade de tempo limite de conexão inativa pode ser definida com `espconn_regist_time()` função.

O ESP8266 suporta no máximo 5 conexões TCP simultâneas.

Veja também:

- [espconn\\_accept](#)

Página 110

## Página 111

- [espconn\\_connect](#)
- [espconn\\_disconnect](#)
- [espconn\\_regist\\_connectcb](#)
- [espconn\\_regist\\_disconcb](#)
- [espconn\\_regist\\_time](#)
- [struct espconn](#)
- [esp\\_tcp](#)

### Envio e recebimento de dados TCP

Neste ponto, vamos supor que temos uma conexão entre um ESP8266 e um aplicativo de parceiro. Ter uma conexão é ótimo, mas agora precisamos ter um conversação. As informações e os dados precisam fluir em uma ou nas duas direções. Existem duas considerações ... podemos receber dados do parceiro ou podemos desejar enviar dados para o parceiro. É importante observar que no TCP, uma conexão é bidirecional. Assim que a conexão for estabelecida, qualquer uma das partes pode enviar dados a qualquer momento. Não existe o conceito de uma das partes com direitos exclusivos de envio ou recebimento. O a escolha de quem é o receptor e quem é o transmissor depende puramente do projeto do aplicativo.

Por exemplo, imagine que temos um projeto para ligar um LED em um ESP8266 quando recebe um caractere "1" e desativa-o quando recebe um caractere "0". Nessa história, o ESP8266 seria exclusivamente um receptor e, simplesmente por nossas escolhas, não precisaria transmitir dados. O parceiro seria exclusivamente um transmissor.

Agora vamos considerar um segundo exemplo. Neste caso, o ESP8266 está conectado a um sensor de temperatura e a cada poucos segundos ele envia a temperatura atual para o parceiro. Nessa história, o ESP8266 é exclusivamente um transmissor e o parceiro apenas um receptor.

Finalmente, podemos criar a imagem de um ESP8266 conectado a vários sensores. Recebe comandos do parceiro como entrada que ele interpreta. Com base nos dados recebidos, o sensor correto é escolhido, seu valor lido e os resultados transmitidos de volta. Nisso história, o ESP8266 é primeiro um receptor e depois se torna um transmissor enquanto o parceiro é o oposto.

Para receber dados de um parceiro, registramos uma função de retorno de chamada usando `espconn_regist_recvcb()`. Passamos na estrutura `espconn` que foi fornecida na callback conectado que identifica nossa conexão. Esta função de retorno de chamada registrada é chamado quando novos dados são disponibilizados pelo parceiro. A função de retorno de chamada é

passou um buffer contendo os dados e um indicador de quantos dados foram recebidos.

A seguir está um exemplo de dados de registro que são recebidos pela rede:

```
void recvCB (void * arg, char * pData, unsigned short len) {
    struct espconn * pEspConn = (struct espconn *) arg;
    os_printf ("Dados recebidos !! - comprimento=%d\n", len);
    int i = 0;
```

Página 111

## Página 112

```
para (i = 0; i <len; i++) {
    os_printf ("%c", pData [i]);
}
os_printf ("\n");
} // Fim do recvCB
```

A função chamada `recvCB ()` é registrada como um retorno de chamada quando os dados estão disponíveis para o conexão. Com isso em mente, podemos começar a executar alguns experimentos e os resultados será interessante.

Se enviarmos dados, vemos o retorno de chamada sendo invocado conforme o esperado. No entanto, como o tamanho dos dados transmitidos, que são recebidos pelo ESP8266, aumenta, em cerca de 1460 bytes, uma coisa estranha acontece. Em vez de `recvCB ()` ser chamado uma vez, vemos que é chamado duas vezes. Na primeira vez, ele obtém os primeiros 1460 bytes e na segunda vez o que permanece. Isso é repetido para incrementos de tamanhos de transmissão de 1460 bytes. Para exemplo, se enviarmos 5000 bytes, `recvCB ()` é chamado 4 vezes. As primeiras três vezes com 1460 bytes de dados e o último com 620 bytes, dando um total de 5000.

Por que isso seria? Parte da resposta é que o ESP8266 tem apenas um pequeno quantidade de RAM disponível e precisa ser capaz de suportar conexões paralelas. Como tal, pode aparentemente limitar os dados enviados do remetente até que o espaço seja disponível para processá-lo.

Nunca é demais enfatizar a importância desse conceito. Dados enviados do servidor através de uma conexão TCP é "transmitido" para o ESP8266. Não há conceito de unidade de transmissão de dados. Em vez disso, os dados enviados no tubo ao remetente chegarão ao ESP8266, mas pode muito bem chegar a taxas diferentes. A ordem dos dados transmitidos é preservado (obviamente). Em princípio, fazer duas transmissões ao remetente de 5 bytes cada um pode resultar em um recebimento no ESP8266 de 10 bytes ou tão facilmente um recebimento de 1 byte e um recebimento de 9 bytes. Não faça **quaisquer** suposições sobre o agrupamento de dados TCP.

Para transmitir dados a um parceiro, usamos a função chamada `espconn_send ()`.

Este comando pega a estrutura `espconn` que identifica qual conexão enviar dados Através dos. A função também leva um ponteiro para um buffer de dados e o comprimento dos dados enviar. Uma consideração vital é que os dados a serem enviados não são enviados imediatamente. Quando chamamos `espconn_send ()` o que estamos fazendo é entregando um buffer de dados para ser transmitido em algum momento no futuro. Prevemos que isso levará alguns milissegundos, mas pode demorar mais. Devemos honrar o contrato. Quando o ESP8266 faz com sucesso transmitir os dados, um retorno de chamada será feito para uma função que foi registrada com o `espconn_register_sentcb ()`. Só depois de ter visto uma confirmação de que o último a solicitação de transmissão foi concluída, devemos executar outro `espconn_send ()` solicitação.

---

**Página 113**

Quando pedimos que os dados sejam transmitidos, fornecemos um ponteiro para um buffer que contém os dados. É importante perceber que devemos manter esses dados até termos certeza seu conteúdo foi enviado. Por exemplo, não podemos solicitar uma transmissão e então descarte imediatamente ou troque o buffer. O que passamos para o sistema operacional é uma indicação para um buffer e até que o sistema operacional nos diga que terminou de consumi-lo, devemos manter seu integridade.

Veja também:

- [espconn\\_regist\\_recvcb](#)
- [espconn\\_send](#)

#### Controle de fluxo

Considere a noção de um ESP8266 na comunicação com um parceiro e o parceiro é enviando 5K de dados por segundo. Agora imagine que o ESP8266 está processando apenas 1K de dados por segundo. Como você pode ver, algo dará errado rapidamente. Nós vamos sobrecarregar o ESP8266 com muitos dados. O que realmente queremos é instituir um fluxo mecanismo de controle de modo que o remetente dos dados seja instruído a reduzir sua entrega de dados a uma taxa que o ESP8266 pode acomodar. O envio pode optar por armazenar em buffer dados no final ou então pode ser instruído a não enviar tantos dados por unidade de tempo pressionando de volta à lógica de transmissão original.

Veja também:

- [espconn\\_recv\\_hold](#)
- [espconn\\_recv\\_unhold](#)

#### Tratamento de erros TCP

Quando uma conexão é formada entre dois parceiros, é essencial que percebamos que não há uma conexão subjacente dedicada real entre eles. Em vez disso, existe apenas uma conexão lógica que parece estar presente no protocolo orientado a datagramas de IP. O que isso pode significar é que, se uma extremidade da conexão terminar de forma anormal, a outra extremidade não saberá imediatamente sobre isso. Por exemplo, se no mundo real eu fizer um telefonema para você, então seu telefone indica que temos uma conexão. Se o a bateria do meu telefone morre, a rede telefônica detecta isso e interrompe a conexão. Seu telefone também desliga e você sabe que não estamos mais em comunicação. No Mundo TCP, isso não acontece. Se meu telefone "TCP" morrer, seu telefone "TCP" não será informado que o meu se foi. Você pode ficar sentado lá indefinidamente ouvindo o silêncio e esperando que eu diga algo.

Para resolver essa situação, o TCP introduz um conceito chamado "keep-alive". A noção é muito simples. Com o keep-alive, os dois parceiros trocam periodicamente um batimento cardíaco comunicação entre si. Contanto que cada um ouça o batimento cardíaco do outro, ambos ainda estão presentes. No entanto, se um lado da conexão for perdido, a pulsação

---

**Página 114**

a solicitação será enviada, mas nenhuma resposta chegará em que ponto, aquele que está enviando o batimento cardíaco assumirá que o parceiro se foi e podemos fazer a limpeza adequada

e ações de desligamento.

Há uma API disponível para controlar as configurações de keep-alive. É chamado `espconn_set_keepalive()`. Possui várias propriedades, incluindo:

- Quanto tempo devemos esperar desde a última vez que ouvimos do parceiro antes enviando um batimento cardíaco?
- Se não houver resposta, quanto tempo entre os batimentos cardíacos subsequentes?
- Quantas vezes devemos enviar um batimento cardíaco até declararmos o parceiro morto?

Recomenda-se que se o processamento keep-alive for usado, então o keep-alive as configurações sejam feitas no manipulador de retorno de chamada do retorno de chamada de conexão. A opção de manter vivo também deve ser explicitamente habilitado usando a chamada `espconn_set_opt()` antes de definir as propriedades keep-alive.

Se a conexão do parceiro for perdida, podemos detectar isso registrando uma função de retorno de chamada com `espconn_reconnect_callback()`.

Veja também:

- [espconn\\_set\\_keepalive](#)
- [espconn\\_get\\_keepalive](#)
- [espconn\\_set\\_opt](#)
- [espconn\\_clear\\_opt](#)

## UDP

Se pensarmos no TCP como formando uma conexão entre duas partes semelhante a um telefone chamada, então UDP é como enviar uma carta pelo sistema postal. Se eu lhe enviasse uma carta, eu precisaria saber seu nome e endereço. Seu endereço é necessário para que a carta pode ser entregue na casa correta enquanto seu nome garante que ela acabe em suas mãos, em oposição a outra pessoa que possa morar com você. Em termos de TCP / IP, o address é o endereço IP e o nome é o número da porta.

Com uma conversa telefônica, podemos trocar tantas ou tão poucas informações quanto nós. Como. Às vezes eu falo, às vezes você fala ... mas não há limite máximo de como muitas informações que podemos trocar em uma conversa. Com uma carta no entanto, há apenas algumas páginas de papel que cabem nos envelopes que tenho à minha disposição.

A noção da analogia do correio é como podemos escolher pensar sobre o UDP. O acrônimo significa User Datagram Protocol e é a noção do datagrama que é semelhante à carta. Um datagrama é uma matriz de bytes que são transmitidos do remetente para o receptor como uma unidade. O tamanho máximo de um datagrama usando UDP é 64 KB. Não a conexão precisa ser configurada entre as duas partes antes que os dados comecem a fluir. Contudo,

há um lado negativo. O remetente dos dados não será informado de um receptor falha ao recuperar os dados. Com o TCP, temos handshaking entre as duas partes que permite ao remetente saber que os dados foram recebidos e, se não, pode automaticamente retransmitir até que seja recebido ou decidirmos desistir. Com UDP, é apenas como um carta, quando enviamos um datagrama, perdemos de vista se ele realmente chega ou não com segurança no destino.

Se quisermos receber datagramas de entrada, devemos registrar o número da porta que estamos interessado em recebê-los. Conseguimos isso por meio do mal nomeado função `espconn_create()`. Esta função faz com que o ESP8266 comece a escutar datagramas de entrada na porta local definida na estrutura `espconn`. Depois de ligar para esta função, você deve então chamar `espconn_register_recvcb()` para registrar uma função de retorno de chamada

que será invocado quando um datagrama chegar.

Aqui está um exemplo de alto nível de configuração de um ouvinte UDP, uma vez que um endereço IP foi alocado:

```
struct espconn conn1;
esp_udp udp1;

void setupUDP () {
    sint8 err;
    conn1.type = ESPCONN_UDP;
    conn1.state = ESPCONN_NONE;
    udp1.local_port = 25867;
    conn1.proto.udp = & udp1;

    err = espconn_create (& conn1);
    err = espconn_regist_recvcb (& conn1, recvCB);
} // Fim da configuração UDP
```

Se desejarmos impedir o ESP8266 de ouvir datagramas, podemos chamar a função chamada `espconn_delete ()`.

Agora é um bom momento para voltar aos endereços IP e números de porta. Devemos começar a esteja ciente de que, em um PC, apenas um aplicativo pode estar escutando em qualquer porta. Para exemplo, se meu aplicativo está escutando na porta 25867, nenhum outro aplicativo também pode estar escutando na mesma porta ... não em seu aplicativo nem em outra cópia / instância minha.

Quando uma conexão de entrada ou datagrama chega a uma máquina, ele chegou porque o endereço IP dos dados enviados corresponde ao endereço IP do dispositivo ao qual eles chegaram. Em seguida, roteamos dentro do dispositivo com base nos números de porta. E aqui é onde eu quero esclareça um detalhe. Nós roteamos dentro da máquina com base no **par** de protocolo e porta número.

Por exemplo, se uma solicitação chega a uma máquina para a porta 25867 por meio de uma conexão TCP, ele é roteado para o aplicativo TCP que observa a porta 25867. Se uma solicitação chega ao mesmo máquina para a porta 25867 sobre UDP, é encaminhado para a porta de observação do aplicativo UDP 25867. O que isto significa é que nós **pode** ter duas aplicações de escuta na mesma

Página 115

## Página 116

porta, mas em protocolos diferentes. Colocando isso de forma mais formal, o espaço de alocação para a porta números é uma função do protocolo e não é permitido que dois aplicativos reservar simultaneamente a mesma porta dentro do mesmo espaço de alocação de protocolo. Embora eu tenha usado a história de um PC executando vários aplicativos, em nosso ESP8266 o A história é semelhante, embora apenas executemos um aplicativo no dispositivo. Se você é solteiro o aplicativo precisa escutar em várias portas, não tente usar a mesma porta com o mesmo protocolo da segunda chamada de função descobrirá que o primeiro já foi alocado o Porto. Este é um detalhe que fico feliz que você esqueça, pois raramente encontrará mas eu queria pegá-lo aqui para completar.

Agora, vejamos o que é necessário para enviar um datagrama. Semelhante a outras funções, precisamos um bloco de controle `struct espconn`. Isso deve ser configurado para usar UDP e nomear o endereço IP remoto e porta. Depois de preenchido, podemos inicializar a estrutura de dados com uma chamada para `espconn_create ()` e agora estamos prontos para enviar dados. Nós usamos o função `espconn_sent ()`. Depois de enviar todos os nossos dados, podemos concluir com um `espconn_delete ()` para liberar os recursos que o ESP8266 mantém para dados enviando.

Aqui está um exemplo:

```
struct espconn sendResponse;
esp_udp udp;
```

```

void sendDatagram (char * datagram, tamanho uint16) {
    sendResponse.type = ESPCONN_UDP;
    sendResponse.state = ESPCONN_NONE;
    sendResponse.proto.udp = & udp;
    IP4_ADDR ((ip_addr_t *) sendResponse.proto.udp-> remote_ip, 192, 168, 1, 7);
    sendResponse.proto.udp-> remote_port = 9876; // Porta remota
    err = espconn_create (& sendResponse);
    err = espconn_send (& sendResponse, "hi123", 5);
    err = espconn_delete (& sendResponse);
}

```

Veja também:

- [espconn\\_create](#)
- [espconn\\_delete](#)
- [espconn\\_send](#)
- [espconn\\_regist\\_recvcb](#)
- [espconn\\_regist\\_sentcb](#)
- [struct espconn](#)

#### Transmitir com UDP

Um dos recursos disponíveis para nós com o UDP é o conceito de transmissão. Isto é o noção de que um remetente de dados pode construir um datagrama e transmiti-lo de forma que todos os dispositivos na mesma sub-rede podem receber uma cópia dele. Os receptores escolhem uma porta UDP e comece a ouvi-lo como fariam normalmente. Um aplicativo de transmissão transmite um

Página 116

## Página 117

mensagem na mesma porta, mas com um endereço IP onde a parte do host do endereço IP são todos binários. Por exemplo, se tivermos uma máscara de rede de 255.255.255.0 e nosso rede é 192.168.1.x , a transmissão no endereço IP 192.168.1.255 será um transmissão. Um endereço IP especial de 255.255.255.255 representa uma transmissão em nosso rede local.

Para o ESP8266, existe uma API chamada wifi\_set\_broadcast\_if () que determina quais interfaces estarão disponíveis para transmissão. As escolhas são a estação, o acesso ponto ou a estação e o ponto de acesso. Uma API correspondente chamada wifi\_get\_broadcast\_if () pode ser usado para recuperar a configuração de transmissão atual Estado.

Veja também:

- [wifi\\_set\\_broadcast\\_if](#)
- [wifi\\_get\\_broadcast\\_if](#)

#### Pedido de ping

No nível TCP / IP, um dispositivo com um endereço IP pode "pingar" outro dispositivo com um IP Morada. O que isso significa é que as mensagens são transmitidas entre eles, o que permite para que saibam que têm uma rota entre si através da rede. Se o destino não está funcionando ou nenhuma rota está disponível, também seremos informados de que houve uma falha.

O ESP8266 fornece uma estrutura chamada struct ping\_option que contém os detalhes de uma solicitação de ping. Isso é passado como um parâmetro para a função chamada ping\_start () que inicia o ping. Antes de chamar esta função, o endereço IP de destino e o número de solicitações de ping deve ser definido na opção struct ping .

Duas funções de retorno de chamada podem ser registradas com ping\_regist\_recv () e ping\_regist\_sent () . O primeiro é chamado quando uma resposta de ping é recebida e o outro é chamado quando uma solicitação de ping é enviada.

Veja também:

- [ping\\_start](#)
- [ping\\_regist\\_recv](#)
- [ping\\_regist\\_send](#)
- [struct ping\\_option](#)

### Serviço de Nome

Na Internet, as máquinas servidoras podem ser encontradas por seu Domain Name Service (DNS) nomes. Este é o serviço que resolve uma representação legível de uma máquina como "google.com" no valor de endereço IP necessário (por exemplo, 216.58.217.206). Dentro para que essa transformação aconteça, o ESP8266 precisa saber o endereço IP do um ou mais servidores DNS que ele usará para executar o nome para o endereço IP

Página 117

## Página 118

mapeamento. Se estivermos usando DHCP, nada mais precisa ser feito, pois o servidor DHCP fornece automaticamente os endereços do servidor DNS. No entanto, se não devemos usar DHCP, então precisamos instruir o ESP8266 sobre os locais dos servidores DNS manualmente. Podemos fazer isso usando a função `espconn_dns_setserver()`. Isso leva um matriz de um ou dois endereços IP como entrada e desse ponto em diante, esses servidores será usado para resolução de DNS. Se dois endereços forem fornecidos e o primeiro for sem resposta, o segundo será usado.

O Google disponibiliza publicamente dois servidores de nomes com os endereços de 8.8.8.8 e 8.8.4.4.

Depois de definir os servidores de nomes, podemos procurar o endereço de um nome de host usando a função `espconn_gethostbyname()`. O código de retorno para esta chamada deve ser cuidadosamente examinado. Podemos ter o endereço imediatamente por causa de um cache ou pode precisar executar uma solicitação de rede e fornecer um retorno de chamada para recuperação posterior. Se o mais tarde, o `ipAddr` é retornado como `NULL` ... no entanto, seu provedor de DNS pode escolher forneça um endereço IP de um mecanismo de pesquisa e, portanto, você receberá um endereço de volta ... mas não aquele para o anfitrião que você esperava !!

Veja também:

- [espconn\\_dns\\_setserver](#)
- [espconn\\_gethostbyname](#)
- Wikipedia: [Sistema de Nome de Domínio](#)
- Google: [DNS público](#)

### Sistemas de nomes de domínio multicast

Em uma rede local com dispositivos dinâmicos indo e vindo, podemos querer um dispositivo para encontrar o endereço IP de outro dispositivo para que possam interagir com cada outro. O problema, porém, é que os endereços IP podem ser alocados dinamicamente por um Servidor DHCP em execução em um ponto de acesso sem fio. Isso significa que o endereço IP de um dispositivo provavelmente não ficará estático. Além disso, não é uma ótima usabilidade de história para referem-se aos dispositivos por seus endereços IP. O que precisamos é de alguma forma de nome dinâmico serviço para encontrar dispositivos por nome onde seus endereços IP não são de administrador configurado. É aqui que o Multicast Domain Name System (mDNS) entra em ação.

Em um alto nível, quando um dispositivo deseja encontrar outro dispositivo com um determinado nome, ele transmite um pedido a todos os membros da rede pedindo uma resposta do dispositivo que tem esse nome. Se uma máquina acredita que tem essa identidade, ela responde com seu sua própria transmissão, que inclui seu nome e endereço IP. Isso não apenas satisfaz a solicitação original, mas outras máquinas na rede podem ver essa interação e cache a resposta por si próprios. Isso significa que eles devem precisar resolver o mesmo

hospedeiro no futuro, eles já têm a resposta.

Página 118

---

## Página 119

Usando o Multicast Domain Name System (mDNS), um ESP8266 pode tentar resolver um nome de host de uma máquina na rede local para seu endereço IP. Ele faz isso por transmitir um pacote solicitando que a máquina com essa identidade responda.

Os demônios do serviço de nomes são implementados por Bonjour e nss-mdns (Linux).

Normalmente, os hosts localizados usando esta técnica pertencem a um domínio que termina em ".local".

Para determinar se o seu PC está participando do mDNS, você pode examinar se ele está ou não ouvindo na porta UDP 5353. Esta é a porta usada para comunicações mDNS.

Veja também:

- Wikipedia - [DNS multicast](#)
- IETF RFC 6762: [DNS multicast](#)
- [DNS multicast](#)
- [Novas tecnologias DNS na Lan](#)
- [Avahi](#) - Implementação do projeto fonte mDNS... para máquinas Unix
- Adafruit - [Rede Bonjour \(Zeroconf\) para Windows e Linux](#)
- [chrome.mdns](#) - descrição da API para Chrome API para mDNS
- Android - [Navegador ZeroConf](#)
- [espconn\\_mdns\\_init](#)
- [espconn\\_mdns\\_close](#)
- [espconn\\_mdns\\_server\\_register](#)
- [espconn\\_mdns\\_server\\_unregister](#)
- [espconn\\_mdns\\_getservername](#)
- [espconn\\_mdns\\_setservername](#)
- [espconn\\_mdns\\_set\\_hostname](#)
- [espconn\\_mdns\\_get\\_hostname](#)
- [espconn\\_mdns\\_disable](#)
- [espconn\\_mdns\\_enable](#)

Instalando Bonjour

Inicie o instalador Bonjour:

Página 119

---

## Página 120

Página 120

---

**Página 121**

Se tudo tiver corrido bem, encontraremos um novo serviço do Windows em execução chamado "Bonjour Serviço":

Página 121

---

**Página 122**

## Trabalhando com SNTP

SNTP é o protocolo de tempo de rede simples e permite que um dispositivo conectado ao Internet para saber a hora atual. Para usar isso, você deve conhecer pelo menos um servidor de horário localizado na Internet. O Instituto Nacional de Ciências dos EUA e Tecnologia (NIST) mantém uma série destes que podem ser encontrados aqui:

- <http://tf.nist.gov/tf-cgi/servers.cgi>

Outros servidores de horário podem ser encontrados em todo o mundo e encorajo você a pesquisar no Google procure o servidor mais próximo ou específico do país.

Depois de saber a identidade de um servidor por seu nome de host ou endereço IP, você pode ligar para qualquer uma das funções chamadas `sntp_setservername()` ou `sntp_setserver()` para declarar que desejamos usar essa instância do servidor de tempo. O ESP8266 pode ser configurado com até três servidores de horário diferentes para que, se um ou dois não estiverem disponíveis, ainda possamos obter um resultado.

O ESP8266 também deve ser informado do fuso horário local em que está sendo executado. Isso está definido com uma chamada para `sntp_set_timezone()` que obtém o deslocamento do número de horas do UTC.

Por exemplo, estou no Texas e meu fuso horário se torna " -5 ".

Página 122

## Página 123

Com eles configurados, podemos iniciar o serviço SNTP no ESP8266 chamando `sntp_init()`. Isso fará com que o dispositivo determine sua hora atual enviando pacotes pela rede para os servidores de horário e examinando suas respostas. Isto é importante notar que imediatamente após chamar `sntp_init()`, você ainda não saberá o que a hora atual pode ser. Isso ocorre porque pode demorar alguns segundos para o ESP8266 envia as solicitações de tempo e obtém suas respostas e tudo isso acontecerá de forma assíncrona aos seus comandos atuais e não será concluído até algum tempo depois.

Quando estiver pronto, podemos recuperar a hora atual com uma chamada para `sntp_get_current_timestamp()` que retornará o número de segundos desde o 1º de Janeiro de 1970 UTC. Também podemos chamar a função chamada `sntp_get_real_time()` que retornará uma representação de string da hora.

Veja também:

- [sntp\\_setserver](#)
- [sntp\\_setservername](#)
- [sntp\\_init](#)
- [sntp\\_set\\_timezone](#)
- [sntp\\_get\\_current\\_timestamp](#)
- [sntp\\_get\\_real\\_time](#)
- [IETF RFC5905: Network Time Protocol Version 4: Protocol and Algorithms Specification](#)

## ESP-NOW

O conceito de ESP-NOW é conseguir um protocolo privado entre conjuntos de ESP8266s. Pense nisso vagamente como uma relação controlador / escravo, onde temos um controlador e potencialmente vários escravos. Os escravos formam conexões "persistentes" com o controlador. O que isso significa é que quando um ESP8266 escravo é ligado, é virtualmente imediatamente

capaz de transmitir para o controlador. Compare isso com a noção do ESP8266 ligando, conectando-se ao ponto de acesso e, em seguida, conectando-se ao dispositivo mestre. Esses fluxos demoram enquanto o ESP-NOW é muito mais rápido na inicialização.

Para começar, um ESP8266 que deseja participar do uso deste protocolo invocará `esp_now_init()`. Caso não queira mais fazer parte deste tipo de rede, pode ligar `esp_now_deinit()`. Antes que a comunicação possa prosseguir, o dispositivo terá que adicionar os pares na rede. Isso é feito por meio de uma chamada para `esp_now_add_peer()`.

Cada dispositivo ESP8266 na rede se declarará como tendo uma função de escravo ou um controlador através de uma chamada para `esp_now_set_self_role()`.

Um `esp_now_delete_peer()` correspondente pode ser usado para esquecer um anterior peer registrado. Quando estiver pronto para transmitir dados, uma chamada pode ser feita para `esp_now_send()` fornecendo o endereço do destinatário, bem como os dados a serem transmitidos. O a quantidade máxima de dados que podem ser enviados atualmente como uma unidade é de 256 bytes.

Página 123

---

## Página 124

Dois retornos de chamada estão disponíveis, os quais são invocados quando uma nova mensagem foi transmitida ou uma nova mensagem foi recebida.

Veja também:

- [esp\\_now\\_init](#)
- [esp\\_now\\_send](#)
- [esp\\_now\\_register\\_recv\\_cb](#)

### GPIOs

O ESP8266 possui 17 pinos GPIO. Quando pensamos em um GPIO, devemos perceber que em qualquer uma vez, cada instância tem dois modos operacionais. Pode ser uma entrada ou um resultado. Quando é uma entrada, podemos ler um valor dele e determinar o nível lógico de o sinal presente no pino físico. Quando é uma saída, podemos escrever um nível lógico para e isso aparecerá como uma saída física.

Lembre-se de distinguir entre o circuito integrado ESP8266, que é um dispositivo minúsculo:

que difere dos vários modelos de placa de breakout, como o ESP-1:

que tem 8 pinos expostos, 4 dos quais são GPIO. O módulo denominado ESP-12:

Página 124

---

## Página 125

que tem 16 pinos expostos, 11 dos quais são GPIO.

Para GPIO, aqui estão os mapeamentos expostos:

Alfinete	ESP-1	ESP-12
GPIO 0	•	•
GPIO 1	•	•
GPIO 2	•	•
GPIO 3	•	•
GPIO 4		•
GPIO 5		•
GPIO 6		
GPIO 7		
GPIO 8		
GPIO 9		
GPIO 10		
GPIO 11		
GPIO 12		•
GPIO 13		•
GPIO 14		•
GPIO 15		•
GPIO 16		•
<b>Totais</b>	<b>4</b>	<b>11</b>

Também é bom nos lembrarmos das pinagens do dispositivo.

---

**Página 126**

Como você pode ver, não há um padrão óbvio para o layout dos pinos e, como tal, você deve ter muito cuidado ao conectar um circuito. É fácil cometer um erro.

Outra consideração vital ao trabalhar com GPIOs é a voltagem. O ESP8266 é um Dispositivo de 3,3 V. Você precisa ser extremamente cauteloso se estiver trabalhando com 5V (ou superior) parceiros MCUs ou sensores. Infelizmente, dispositivos como o Arduino são normalmente 5V Conversores USB → UART e muitos sensores. Isso significa que você provavelmente não será trabalhando em um ambiente de tensão mista. Sob nenhuma circunstância você deve pensar que pode alimentar o ESP8266 com uma tensão direta de mais de 3,3 V. Obviamente, você pode converter tensões mais altas para 3,3 V, mas nunca tente conectar uma tensão maior diretamente. Outra consideração mais útil é ao usar GPIOs para entrada e alimentação de sinal maior que 3,3 V como um valor de sinal alto. Eu sugiro fortemente não fazer isso. Algumas pessoas podem alegar que você pode "se safar" e, se você experimentar, pode (parece) funcionar, mas você está assumindo um risco desnecessário sem nenhuma razão óbvia. Se funcionar ... então funcionará até que não, momento em que será tarde demais e você pode cozinhar seu dispositivo.

Em meus próprios experimentos, acidentalmente sobrechargei ESP8266s, voltagem reversa alimentou ESP8266s e aplicou uma tensão muito alta como entrada. Em cada caso, o resultado era um chip morto e, em alguns casos, tentando ver se ainda funcionava aplicando tensão normal resultou no dispositivo não só não funcionar, mas ficar muito quente ao toque queimou meus dedos.

Como os acidentes acontecem durante a construção de circuitos baseados em GPIO, recomendo comprar mais instâncias ESP8266 do que você precisa. Assim, se acontecer de você se encontrar precisando de um segundo (ou terceiro ou quarto) você os terá à sua disposição.

Antes de fazer uso de quaisquer funções do ESP8266 GPIO, você deve chamar o fornecido função `gpio_init()`. O que isso realmente faz é desconhecido, mas as regras dizem que devemos chamá-lo e devemos chamá-lo.

A maneira como o ESP8266 pensa nos GPIOs é como se cada GPIO estivesse um pouco em um 16 bits variedade.

(Voltaremos a como 17 GPIOs mapeiam para 16 bits posteriormente)

Uma matriz contém uma indicação de se o GPIO é de entrada ou saída. Nós vamos chame isso de array de direção. Uma segunda matriz indica os valores dos GPIOs. Para entrada GPIOs, o valor é o valor no pino. Para GPIOs de saída, o valor é o valor a ser escrito no pino. Chamaremos isso de array de valor.

Uma função é fornecida pelo ESP8266 chamada `gpio_output_set()`. Esta função leva **quatro** valores de 16 bits a serem usados como máscaras em relação às duas matrizes de 16 bits.

A primeira máscara é chamada de " `set_mask` ". Um valor 1 na máscara definida define o o valor do bit correspondente deve ser 1 na matriz de valor.

A segunda máscara é chamada de " `clear_mask` ". Um valor 1 na máscara transparente define o o valor do bit correspondente seja 0 na matriz de valor.

Observe que em ambos os casos, se as máscaras têm valor 0 , os valores originais são inalterado.

A terceira máscara é chamada de máscara " `enable_output` ". Um valor 1 na máscara de saída de habilitação define o GPIO correspondente para o modo de saída.

A quarta máscara é chamada de máscara " `enable_input` ". Um valor 1 na máscara de entrada de habilitação define o GPIO correspondente para o modo de entrada.

Tome cuidado para não definir um GPIO para ser entrada e saída ou para ter um valor de 1 e 0. Os resultados serão indefinidos.

As constantes são definidas para cada uma das posições de bit. Essas constantes são:

- BIT0 -  $2^0$
- BIT1 -  $2^1$
- ...
- BIT31 -  $2^{31}$

Então, por exemplo. Se quisermos definir o GPIO 5 como entrada, podemos codificar:

```
gpio_output_set(0, 0, 0, BIT5);
```

para definir o GPIO 4 para ser gerado e ter um valor alto, podemos codificar:

```
gpio_output_set(BIT4, 0, BIT4, 0);
```

para definir GPIO 0 e 1 como saída e o primeiro como 1 e o segundo como 0:

```
gpio_output_set(BIT0, BIT1, BIT0 | BIT1, 0);
```

---

## Página 128

Se quisermos recuperar os valores dos GPIOs, podemos usar o método `gpio_input_get()` método. Isso retorna uma máscara de bits contendo todos os bits.

Temos algumas macros auxiliares disponíveis. Estes são invólucros úteis em torno `gpio_output_set()` e `gpio_input_get()` .

- `GPIO_OUTPUT_SET(GPIO_NUMBER, valor)` - Define o GPIO correspondente para ser saída e define seu valor.
- `GPIO_DIS_OUTPUT(GPIO_NUMBER)` - Define o GPIO correspondente a ser inserido (saída desativada).
- `GPIO_INPUT_GET(GPIO_NUMBER)` - Obtém o valor da entrada GPIO

Uma vez que os pinos em um ESP8266 podem servir a vários propósitos, devemos primeiro declarar o que função que o pino terá. Para fazer isso, usamos uma macro que define a função do pino lógico:

```
PIN_FUNC_SELECT(pinName, functionUsage)
```

Por exemplo, para definir GPIO2 como um pino GPIO e definir seu valor, podemos codificar:

```
PIN_FUNC_SELECT(PERIPHIS_IO_MUX_GPIO2_U, FUNC_GPIO2);
GPIO_OUTPUT_SET(2, 1);
```

Aqui está a tabela completa de mapeamentos.

Nome do Pin	Função 1	Função 2	Função 3	Função 4	Dispositivos de pinos físicos
MTDI_U	MTDI	I2SI_DATA	HSPIQ MISO	GPIO12	10 12
MTCK_U	MTCK	I2SI_BCK	HSPID MOSI	GPIO13	12 12
MTMS_U	MTMS	I2SI_WS	HSPICLK	GPIO14	9 12
MTDO_U	MTDO	I2SO_BCK	HSPICS	GPIO15	13 12
U0RXD_U	U0RXD	I2SO_DATA		GPIO3	25 1, 12
U0TXD_U	U0TXD	SPICS1		GPIO1	26 1, 12
SD_CLK_U	SD_CLK	SPICLK		GPIO6	21
SD_DATA0_U	SD_DATA0	SPIQ		GPIO7	22
SD_DATA1_U	SD_DATA1	SPIID		GPIO8	23
SD_DATA2_U	SD_DATA2	SPIHD		GPIO9	18
SD_DATA3_U	SD_DATA3	SPIWP		GPIO10	19
SD_CMD_U	SD_CMD	SPICS0		GPIO11	20
GPIO0_U	GPIO0	SPICS2			15 1, 12
GPIO2_U	GPIO2	I2SO_WS	U1TXD		14 1, 12
GPIO4_U	GPIO4	CLK_XTAL			16 12
GPIO5_U	GPIO5	CLK_RTC			24 12

Página 128

## Página 129

A seguir estão as chaves para alguns dos valores da tabela:

- Coluna Dispositivos
  - 1 = ESP-1
  - 12 = ESP-12

Aqui estão os pinos GPIO por mapeamento:

GPIO	Nome do Pin	NodeMCU	Notas	Risco
GPIO0	GPIO0_U	D3	O pino controla o estado do ESP8266 na inicialização. Cuidado quando usado como um pino de saída.	
GPIO1	U0TXD_U	D10	O pino é comumente usado para fazer o flash do dispositivo.	
GPIO2	GPIO2_U	D4	Usado para saída UART1 e, como tal, é provável que seja usado durante o tempo de desenvolvimento para depuração. Escrito para quando brilhou com novo firmware.	
GPIO3	U0RXD_U	D9	O pino é comumente usado para fazer o flash do dispositivo.	
GPIO4	GPIO4_U	D2	Use apenas como GPIO.	
GPIO5	GPIO5_U	D1	Use apenas como GPIO.	
GPIO6	SD_CLK_U		Não exposto nos dispositivos atuais.	
GPIO7	SD_DATA0_U		Não exposto nos dispositivos atuais.	

GPIO8	SD_DATA1_U	Não exposto nos dispositivos atuais.	
GPIO9	SD_DATA2_U SD2	Não exposto nos dispositivos atuais.	
GPIO10	SD_DATA3_U SD3	Não exposto nos dispositivos atuais.	
GPIO11	SD_CMD_U	Não exposto nos dispositivos atuais.	
GPIO12	MTDI_U	D6	
GPIO13	MTCK_U	D7	
GPIO14	MTMS_U	D5	
GPIO15	MTDO_U	D8	Usado para controlar UART0 RTS e, portanto, pode ter um influência no flash do firmware desde os dados do firmware chega via UART0.
GPIO16	???	D0	???

A corrente de saída máxima de um pino GPIO é de apenas 12 mA.

Se você puder escolher, se estiver usando GPIO0, use-o como um pino de entrada em vez de uma saída alfinete. A razão para isso é que quando você está desenvolvendo soluções, você precisa trazer GPIO0 baixo para colocar o ESP8266 em modo flash, onde lê novos programas do UART. Isso significa que você mudará o sinal de entrada para GPIO0. Se você usar o pino como uma saída, existe a possibilidade de que, quando você alterar sua fiação para reduzi-la ou pressione um botão para baixá-lo; se o sinal estiver alto naquele momento, você causará um curto-circuito.

## Página 130

No entanto, se o pino for inserido, isso não será um problema. Idealmente, evite usar GPIO0 completamente e deixe-o especificamente para inicializar o dispositivo em diferentes modos.

Veja também:

- [PIN\\_FUNC\\_SELECT](#)
- [GPIO\\_OUTPUT\\_SET](#)
- [GPIO\\_DIS\\_OUTPUT](#)
- [GPIO\\_INPUT\\_GET](#)
- [gpio\\_output\\_set](#)
- [gpio\\_input\\_get](#)

### Configurações de pullup e pull down

Normalmente pensamos em um pino GPIO de entrada como tendo um sinal alto ou baixo fornecido para isso. Isso significa que ele está conectado a +ve ou terra. Mas se estiver conectado a nem? Nesse caso, o pino é considerado em estado flutuante. Há tempos onde desejamos definir um pino não conectado como sendo alto ou baixo. Um pino desconectado que deve ser considerado alto é denominado "puxado", enquanto um pino desconectado que deve ser considerado baixo é denominado "suspenso". Isso vem da prática de hardware físico de anexar resistores para puxar para cima ou para baixo o sinal quando, de outra forma, seria flutuando.

No SDK ESP8266, podemos definir um GPIO como sendo puxado usando a macro chamado PIN\_PULLUP\_EN e podemos definir o GPIO como não sendo mais puxado usando a macro PIN\_PULLUP\_DIS .

### Tratamento de interrupção GPIO

Se considerarmos que o sinal em um pino pode se mover de alto para baixo ou de baixo para alto, tal mudança pode ser algo que nosso aplicativo estaria interessado em saber. Para determinar quando essa mudança acontece, podemos pesquisar continuamente o valor e detectar um mudança de transição. No entanto, esta não é a melhor solução por uma série de razões. Primeiro,

temos que trabalharativamente para verificar se um valor foi alterado. Em segundo lugar, haverá uma latência desde o momento em que o evento ocorre até o momento em que verificamos. Em terceiro lugar, é possível perder completamente uma mudança de sinal se a duração da mudança for baixo. Por exemplo, se verificarmos o valor de um pino e o encontrarmos alto, imediatamente depois que cair e depois aumentar novamente, da próxima vez que pesquisarmos, ainda veremos o pino alto e nunca soube que estava baixo por um curto período.

A solução para todos esses problemas é a noção de uma interrupção. Uma interrupção é semelhante a sua campainha em sua casa. Sem campainha (ou ouvindo alguém batendo) você teria que verificar periodicamente se há alguém na porta. Isto desperdiça seu tempo para a maioria dos casos em que não há ninguém e também garante que quando há alguém presente, você o atende em tempo hábil.

Página 130

## Página 131

Na terra dos ESP8266s, podemos definir uma função de retorno de chamada de interrupção que será chamada quando um pino muda seu valor de sinal. Também podemos definir o que constitui uma razão para invocando o retorno de chamada. Podemos configurar o manipulador de retorno de chamada (teoricamente chamado de manipulador de interrupções) pino a pino.

Primeiro, vamos considerar a função de retorno de chamada de interrupção. Isso é registrado com uma chamada para `gpio_intr_handler_register()`.

Podemos habilitar ou desabilitar o tratamento de interrupções em um nível global. Uma chamada para `ETS_GPIO_INTR_ENABLE` ativa o tratamento de interrupções durante uma chamada para `ETS_GPIO_INTR_DISABLE` desativa o tratamento de interrupção global.

Para habilitar uma interrupção para um pino específico, usamos a função chamada `gpio_pin_intr_state_set()`. Isso nos permite definir o motivo pelo qual uma interrupção pode ocorrer. Os motivos incluem:

- Desativar - não chame uma interrupção em uma mudança de sinal.
- PosEdge - Chame o manipulador de interrupção em uma mudança de baixo para alto.
- NegEdge - Chame o manipulador de interrupção em uma mudança de alto para baixo.
- AnyEdge - Chame o manipulador de interrupção em uma mudança de baixo para alto ou um mudar de alto para baixo.
- Hi - Chame o manipulador de interrupção enquanto o sinal está alto.
- Lo - Chame o manipulador de interrupção enquanto o sinal está baixo.

Notas: Quando fui implementar um manipulador de interrupções em um projeto, descobri essa teoria e a prática não encontrou. Por enquanto, a única história que tenho trabalhando para interromper manipulador tem a seguinte aparência:

```
static void intrHandlerCB (
    uint32 interruptMask, //! <Uma máscara indicando quais GPIOs foram alterados.
    void * arg           //! <Argumento opcional.
) {
    uint32 gpio_status = GPIO_REG_READ(GPIO_STATUS_ADDRESS);
    os_printf("status: 0x%08x\r\n", gpio_status);
    gpio_intr_ack(interruptMask);
    pin int;
    para (pino = 0; pino < 16; pino++) {
        if ((interruptMask & (1 << pin)) != 0) {
            // Faça alguma coisa
            gpio_pin_intr_state_set(GPIO_ID_PIN(pin), GPIO_PIN_INTR_ANYEDGE);
        }
    }
}
```

Veja também:

- [gpio\\_intr\\_handler\\_register](#)
- [gpio\\_pin\\_intr\\_state\\_set](#)

Página 131

---

## Página 132

- [gpio\\_intr\\_pending](#)
- [gpio\\_intr\\_ack](#)

### Expandindo o número de GPIOs disponíveis

Embora os dispositivos ESP tenham apenas um número limitado de pinos GPIO, isso não precisa ser uma restrição para nós. Temos a capacidade de expandir o número de GPIOs disponíveis para através de alguns circuitos integrados relativamente baratos. Um deles é chamado de PCF8574. (O PFC8574A é o mesmo, mas tem um conjunto diferente de endereços).

Este é um dispositivo I<sub>2</sub>C e, portanto, funciona em apenas dois fios. Usando este IC, fornecemos um 3 endereço de bit (000-111) que é usado para selecionar o endereço escravo do dispositivo. Desde cada endereço tem 8 IOs e podemos ter até 8 dispositivos, o que significa um total de 64 adicionais alfinetes.

Parece que o dispositivo usará um resistor pull-up para uma alta e trará o pino para terreno para baixo. Isso significa que se quisermos usar qualquer um dos pinos para entrada, devemos definir seu modo de gravação para alto primeiro. Isso permitirá que um sinal de entrada alto ou baixo seja detectado. Parece que, se definirmos o sinal de saída para baixo e, em seguida, alimentarmos um sinal alto no dispositivo, teríamos um curto.

Aqui está o diagrama de pinos do dispositivo:

Aqui está uma descrição dos pinos:

**Página 133**

Símbolo	Alfinete	Descrição
A0-A2	1, 2, 3	Endereçando
P0-P7	4, 5, 6, 7, 9, 10, 11, 12	E / S bidirecional
INT	13	Interromper saída
SCL	14	Linha Serial do Relógio
SDA	15	Linha de Dados Seriais
V <sub>dd</sub>	16	Tensão de alimentação (2,5 V - 6 V)
Vss (solo)	8	Chão

O endereço no qual o dispositivo escravo pode ser encontrado é configurável por meio dos pinos A0-A2 . Ele aparece no seguinte endereço:

PCF8574

0 1 0 0 A2 A1 A0

PCF8574A

0 1 1 1 A2 A1 A0

Os pinos A0-A2 não devem flutuar.

Isso resulta na seguinte tabela:

A2	A1	A0	Endereço	Endereço
			PCF8574	PCF8574A
0	0	0	0x20	0x38
0	0	1	0x21	0x39
0	1	0	0x22	0x3a
0	1	1	0x23	0x3b
1	0	0	0x24	0x3c
1	0	1	0x25	0x3d
1	1	0	0x26	0x3e
1	1	1	0x27	0x3f

Aqui está um programa de exemplo que aciona LEDs para criar um efeito Cylon.

```
#include <Wire.h>
#include <Ticker.h>
// SDA - Amarelo - 4
// CLK - Branco - 5

#define SDA_PIN 4
```

Página 133

**Página 134**

```
# define CLK_PIN 5

Ticker ticker;
contador interno = 0;
dir int = 1;

void timerCB() {
    Wire.beginTransmission(0x20);
```

```
Wire.write (~ ((uint8_t) 1 << contador));
Wire.endTransmission ();
contador += dir;
if(contador == 8) {
    contador = 6;
    dir = -1;
} else if (contador == -1) {
    contador = 1;
    dir = 1;
}
}

void setup ()
{
    Wire.begin (SDA_PIN, CLK_PIN);
    ticker.attach (0.1, timerCB);
}

void loop ()
{
```

O circuito correspondente é:

E em uma placa de ensaio:

Veja também:

- [Um video tutorial sobre este tópico](#)
- [Uma classe para PCF8574 \(RobTillaart /Github\)](#)
- [EXPANSOR DE 8BIT IO \(PCF8574\)](#)

Página 135

## Página 136

- skywodd - [Biblioteca Arduino PCF8574](#)
- [Folha de Dados - NXP](#)
- [Página do produto - TI](#)
- [Trabalhando com I2C](#)
- 

Biblioteca C ESP\_PCF8574

Uma classe chamada ESP\_PCF8574 foi escrita para usar as bibliotecas do Arduino. A biblioteca pode ser encontrado em [Github](#).

Ele fornece os seguintes métodos:

### **ESP\_PCF8574.begin**

Inicie o controle PCF8574.

```
void begin (uint8_t address, uint8_t sda, uint8_t clk)
```

O parâmetro de endereço é o endereço I2C do PCF8574... normalmente 0x20 - 0x27 .

Os parâmetros sda e clk são os números dos pinos usados para I2C SDA e CLK.

### **ESP\_PCF8574.getBit**

Recupere a entrada de um determinado bit.

```
bool getBit (uint8_t bit)
```

Se pensarmos nos 8 GPIOs fornecidos pelo PCF8574 como sendo 8 bits de dados, este método recupera o valor dos dados em um determinado pino de entrada.

**ESP\_PCF8574.getByte**

Recupere todos os 8 bits de entrada.

```
uint8_t getByte()
```

Se pensarmos nos 8 GPIOs fornecidos pelo PCF8574 como sendo 8 bits de dados, este método recupera o valor de todas as entradas.

**ESP\_PCF8574.setBit**

Defina o valor de um determinado pino de saída.

```
void setBit(uint8_t bit, valor bool)
```

Defina o valor de um determinado pino de saída.

**ESP\_PCF8574.setByte**

Página 136

**Página 137**

Defina o valor de todos os pinos de saída.

```
void setByte(valor uint8_t)
```

Defina o valor de todos os pinos de saída.

Biblioteca JavaScript PCF8574

Uma biblioteca JavaScript foi construída para fazer a interface do PCF8574 a partir de um JavaScript aplicativo.

Invocar do JavaScript é a própria simplicidade. Uma vez que o PCF8574 é apenas um simples I2C placa, usando a interface I2C é tudo o que é necessário.

Por exemplo:

```
var sda = NodeMCU.D1; // Amarelo
var scl = NodeMCU.D2; // Branco
I2C1.setup ({scl: scl, sda: sda});
I2C1.writeTo (0x20, valor);
```

Supondo que o endereço do nosso PCF8574 seja 0x20 .

**Trabalhando com I2C**

A interface I2C é uma tecnologia de interface serial para acessar dispositivos. Tem dois linhas de sinal chamadas SDA (Dados) e SCL (Relógio). O ESP8266 pode atuar como um mestre e os dispositivos conectados a jusante atuam como escravos. Até 127 escravos distintos são teoricamente anexável. Cada dispositivo escravo tem um endereço único e o mestre decide qual escravo receberá os dados ou terá permissão para falar em seguida.

Todos os escravos conectados usam uma conexão de "dreno aberto" para o barramento. Isso significa que quando eles se conectam, seu anexo é um circuito aberto ou aterrado como uma saída.

Por causa disso, é impossível que haja um conflito elétrico como seria impossível para um dispositivo declarar um sinal alto enquanto outro tentava declarar um sinal baixo sinal. A presença de um sinal lógico alto ocorre quando o dispositivo escravo atual vai circuito aberto. Isso significa que precisamos de resistores pull-up nas linhas de modo que quando não

um está ativamente afirmado um sinal baixo, eles são puxados para cima para um sinal alto lógico. UMA o valor do resistor de 4,7 KΩ é recomendado.

O início de uma transmissão é indicado quando o SCL é deixado alto e SDA é puxado para baixo.

Isso informa a todos os dispositivos escravos que um endereço está prestes a ser emitido. Quando o endereço é visto por todos os escravos, apenas um deles deve corresponder e os outros dispositivos ignore o pedido.

## Página 138

O endereço de um escravo segue a indicação de início inicial e é composto por 7 bits com o bit mais significativo primeiro. Após o endereço de 7 bits é um 8º final bit que indica se esta é uma solicitação de leitura ou gravação. Um valor de 1 indica uma leitura do escravo enquanto um valor de 0 indica uma gravação do mestre.

Imediatamente após os 8 bits de endereço, vem o bit de confirmação. Este pedaço **não** é transmitido do mestre para o escravo, mas em vez disso é transmitido do escravo para o mestre. Certifique-se de entender isso como ao olhar para diagramas que mostram dados no Fio SDA, esses diagramas normalmente não mostram a origem dos dados, apenas seus sequências. O tempo de retorno do último bit dos dados de endereço / direção de 8 bits enviados do mestre para o bit de confirmação enviado do escravo acontece sem perder todos os ciclos do relógio precisam ser rápidos. Um valor de 0 na confirmação indica que o escravo irá processar ou responder. Um valor de 1 na confirmação afirma que ninguém é respondendo ou o escravo não está presente.

Após esse quadro de endereçamento, vêm os quadros ou quadros de dados. Para um mestre escrever solicitação, o mestre enviará 8 bits de dados e esperará um único bit de confirmação. Para uma leitura do escravo, o escravo enviará 9 bits de dados (8 bits de dados e um reconhecimento).

O mestre finalmente enviará uma indicação de fim de comunicação (ou parada) que é uma transição para alto no relógio com NÃO transição correspondente para baixo e, em **seguida**, uma transição de baixo para alto na linha SDA.

Para usar I2C, primeiro transmitimos uma solicitação de início usando `i2c_master_start()`. Nós seguimos isso por o endereço e o sinalizador de leitura / gravação. Uma vez que um endereço tem 7 bits e o sinalizador de leitura / gravação é um bit, isso totaliza 8 bits e, portanto, podemos escrever um byte:

```
i2c_master_writeByte ((endereço << 1) | readOrWrite);
```

Em seguida, podemos ler e verificar o sinalizador de confirmação usando:

```
if (i2c_master_checkAck () == true) {  
    ...  
}
```

E a partir daqui, podemos encerrar ou executar a próxima parte de leitura ou gravação.

Quando desejamos encerrar, executamos `i2c_master_stop()`.

Podemos enviar uma consulta de endereço para cada um dos endereços de dispositivo possíveis e ver se obter um reconhecimento. Se o fizermos, teremos um dispositivo I2C nesse endereço. Esta pode ser usado para criar um mapa de dispositivos.

Aqui está um exemplo de aplicativo que faz exatamente isso:

```
#include <ets_sys.h>  
#include <osapi.h>  
#include <os_type.h>  
#include <gpio.h>
```

**Página 139**

```
#include <user_interface.h>
#include <espconn.h>
#include <mem.h>
#include "driver / uart.h"
#include "driver / i2c_master.h"

void user_rf_pre_init (void) {
}

os_timer_t scanTimer;

void scanTimerCB (void * pArg) {
    os_printf ("--- Examinando --- \n");
    uint8_t i;
    para (i = 1; i < 127; i++) {
        i2c_master_start ();
        i2c_master_writeByte (i << 1);

        if (i2c_master_checkAck ()) {
            os_printf ("Dispositivo encontrado em: 0x% 2x \n", i);
        }
        i2c_master_stop ();
    }
    os_printf ("Concluído! \n");
}

} // Fim do timerCallback

void init () {
    i2c_master_gpio_init ();
    os_timer_setfn (& scanTimer, scanTimerCB, NULL);
    os_timer_arm (& scanTimer, 10000, 1);
}

void user_init (void) {
    uart_init (BIT_RATE_115200, BIT_RATE_115200);
    system_init_done_cb (init);
}

} // Fim do user_init
```

Veja também:

- [I2C Bus](#)
- Sparkfun - [Tutorial: I2C](#)
- [APIs I2C Master](#)
- [wget http://<endereço IP> --quiet --output-document=-](http://<endereço IP> --quiet --output-document=-)

**Trabalhando com SPI - Interface Periférica Serial**

SPI é um protocolo serial usado para comunicação entre mestres e escravos. Todos os escravos conectar ao mesmo barramento, mas apenas o escravo com seu pino SS baixo tem permissão para transmitir. SPI é um protocolo full duplex. O que isso significa é que enquanto os dados estão sendo enviados do mestre para o escravo, o escravo está simultaneamente enviando dados de volta para o mestre. O pino MOSI contém os dados seriais do mestre para o escravo enquanto o pino MISO contém os dados do escravo para o mestre.

**Página 140**

Normalmente três pinos:

- MISO - M aster I n, S lave O ut - Enviando dados do escravo para o mestre
- MOSI - M aster O ut, S lave I n - Enviando dados para o escravo do mestre
- SCK (SCLK) - Relógio serial - sincroniza dados do mestre / escravo  
relação

Também há um sinal adicional:

- SS (CSN (Chip Select NOT), NSS) - S lave S elect - Usado para habilitar / desabilitar o escravo para que possa haver vários escravos. SS baixo significa que o escravo é o ativo escravo.

Uma vez que este é um protocolo serial e receberemos dados em bytes, precisamos estar cientes se os dados chegarão ou não ao LSB primeiro ou ao MSB primeiro. Haverá uma opção para controlar isso.

Para o relógio, estaremos travando dados e precisaremos saber quais arestas e as configurações são importantes. Haverá uma opção de modo de relógio para controlar isso. No SPI lá são dois atributos chamados fase e polaridade. Fase (CPHA) é se estamos travando dados em alta ou baixa e Polaridade (CPOL) é alta ou baixa significa que o relógio está ocioso.

CPOL = 0 significa que o relógio é o padrão baixo, CPOL = 1 significa que o relógio é o padrão alto.

Quando CPOL = 0 , a seguir estão os valores para CPHA

CPHA = 0 significa que os dados são capturados na borda ascendente do relógio, CPHA = 1 significa que os dados são capturados em borda de queda do relógio.

Quando CPOL = 1 , a seguir estão os valores para CPHA

CPHA = 0 significa que os dados são capturados na borda de descida do relógio, CPHA = 1 significa que os dados são capturados em borda ascendente do relógio.

O SPI envolve esses dois sinalizadores em quatro modos definidos e nomeados:

Modo	Polaridade do relógio - CPOL	Fase do relógio - CPHA
SPI_MODE0	0 (relógio padrão baixo)	0
SPI_MODE1	0 (relógio padrão baixo)	1
SPI_MODE2	1 (relógio padrão alto)	0
SPI_MODE3	1 (relógio padrão alto)	1

Também para o relógio, em que velocidade precisamos saber a que velocidade os dados devem estar mudou-se. Haverá uma opção de velocidade de controle de relógio para controlar isso.

### Hardware SPI

O ESP8266 tem suporte a SPI de hardware que é controlado pela configuração de registros e usando Macros fornecidas pelo SDK. Existem certos pinos físicos que são reservados para o hardware SPI. Estes são:

GPIO	NodeMCU	Nome	Função
GPIO12	D6	HMISO	MISSÔ
GPIO13	D7	HMOXI	MOSI

GPIO14	D5	HSCLK	CLK
GPIO15	D8	HCS	CS

Para começar, vamos pensar no relógio. Esta é a velocidade com que o ESP8266 alterna o fluxo de comunicação com o parceiro SPI. Podemos definir que seja o mesmo velocidade como o clock do processador (80 MHz) ao gravar em um registro periférico.

```
WRITE_PERL_REG(SPI_CLOCK(HSPI), SPI_CLK_EQU_SYSCLK)
```

Isso instrui o ESP8266 que a velocidade do relógio do SPI do hardware deve ser igual ao velocidade do relógio do sistema, que normalmente é de 80 MHz.

Se essa velocidade for muito rápida, podemos alterar a velocidade do clock com um divisor chamado SPI\_CLKDIV\_PRE. Isso divide a velocidade do clock por um valor. Se você quiser dividir por X em seguida, forneça um valor de X-1.

Por exemplo, para dividir a velocidade do clock por 10, forneça um valor de 9.

Agora temos uma velocidade de clock bruta. No entanto, há mais. Existe uma configuração chamada SPI\_CLKCNT\_N que define quantos desses tiques de relógio brutos devem constituir um SPI ciclo do relógio. Lembre-se de que um ciclo de clock do SPI começa alto, passa para baixo e então retorna ao alto novamente. O SPI\_CLKCNT\_N define quantos ciclos de relógio brutos correspondem ao ciclo de clock do SPI. Por exemplo, especificando um valor de 9 (o valor desejado é n + 1) significa que 10 ciclos de relógio bruto serão um ciclo de relógio SPI. Perceba que este é um divisor do ciclo de clock bruto.

Existem outras configurações que afetam o relógio e são chamadas de SPI\_CLKCNT\_H e SPI\_CLK\_CNT\_L. Juntos, eles definem o número de ciclos de clock que o clock é alto vs baixo. Isso molda o sinal do relógio. Os valores aqui são codificados para significar o número de ciclos de clock brutos em que o clock do SPI é alto vs baixo. A diferença entre eles dão o resultado. Por exemplo, definindo SPI\_CLKCNT\_H = 6 e SPI\_CLKCNT\_L = 1 resultados em uma diferença de 5, o que significa que 5 ciclos de relógio bruto serão altos e o relógio restante os ciclos (5) serão baixos.

## Página 142

Função	Bits	mascarar	Mudança
SPI_CLK_EQU_SYSCLK	31	N / D	SPI_CLK_EQU_SYSCLK
SPI_CLKDIV_PRE	30:18	SPI_CLKDIV_PRE	SPI_CLKDIV_PRE_S
SPI_CLKCNT_N	17:12	SPI_CLKCNT_N	SPI_CLKCNT_N_S
SPI_CLKCNT_H	11: 6	SPI_CLKCNT_H	SPI_CLKCNT_H_S
SPI_CLKCNT_L	5: 0	SPI_CLKCNT_L	SPI_CLKCNT_L_S

Existem três registros que controlam a entrada e saída de dados. Eles são chamados de SPI\_USER , SPI\_USER1 e SPI\_USER2 .

SPI\_USER contém os seguintes sinalizadores:

- SPI\_CS\_SETUP - Quando habilitado, a linha de seleção de chip é puxada para baixo alguns ciclos antes da transmissão, permitindo que o dispositivo SPI parceiro fique pronto para um transmissão.
- SPI\_CS\_HOLD - Quando habilitado, a linha de seleção de chip permanece baixa por alguns ciclos após a transmissão, permitindo que o dispositivo SPI do parceiro continue um pouco.

Função	Pedaço
--------	--------

SPI_USR_COMMAND	31
SPI_USR_ADDR	30
SPI_USR_DUMMY	29
SPI_USR_MISO	28
SPI_USR_MOSI	27
SPI_USR_MOSI_HIGHTPART	25
SPI_USR_MISO_HIGHTPART	24
SPI_SIO	16
SPI_FWRITE_QIO	15
SPI_FWRITE_DIO	14
SPI_FWRITE_QUAD	13
SPI_FWRITE_DUAL	12
SPI_WR_BYTE_ORDER	11
SPI_RD_BYTE_ORDER	10
SPI_CK_OUT_EDGE	7
SPI_CK_I_EDGE	6
SPI_CS_SETUP	5
SPI_CS_HOLD	4
SPI_FLASH_MODE	2

Página 142

---

**Página 143**

SPI\_USER1 contém quatro configurações para o número de bits em várias configurações SPI. Esses estão:

- SPI\_USR\_ADDR\_BITLEN - Quantos bits há no endereço SPI.
- SPI\_USR\_MOSI\_BITLEN - Quantos bits nos dados MOSI.
- SPI\_USR\_MISO\_BITLEN - Quantos bits nos dados MISO.
- SPI\_USR\_DUMMY\_CYCLELEN - Quantos bits nos ciclos fictícios.

Um registro adicional denominado SPI\_ADDR contém o endereço.

Função	Bits	mascarar	Mudança
SPI_USR_ADDR_BITLEN	31:26	SPI_USR_ADDR_BITLEN	SPI_USR_ADDR_BITLEN_S
SPI_USR_MOSI_BITLEN	25:17	SPI_USR_MOSI_BITLEN	SPI_USR_MOSI_BITLEN_S
SPI_USR_MISO_BITLEN	16: 8	SPI_USR_MISO_BITLEN	SPI_USR_MISO_BITLEN_S
SPI_USR_DUMMY_CYCLELEN	7: 0	SPI_USR_DUMMY_CYCLELEN	SPI_USR_DUMMY_CYCLELEN_S

Função	Bits	mascarar	Mudança
SPI_USR_COMMAND_BITLEN	31:28	SPI_USR_COMMAND_BITLEN	SPI_USR_COMMAND_BITLEN_S
SPI_USR_COMMAND_VALUE	15: 0	SPI_USR_COMMAND_VALUE	SPI_USR_COMMAND_VALUE_S

Os detalhes desses registros e macros que definem constantes podem ser encontrados no  
O arquivo spi\_register.h fornecido no diretório de inclusão de drivers.

Veja também:

- [Registros de relógio SPI de hardware](#)
- [Registros de Comando e](#)
- [Wikipedia - Barramento](#)

### O MetalPhreak / ESP8266 SPI Driver

Interagir com o SPI de hardware pode ser desafiador. Felizmente, um projeto de código aberto no GitHub forneceu uma solução fácil de usar para o problema. O projeto é conhecido como MetalPhreak / ESP8266\_SPI\_Driver que é composto de um arquivo fonte C e um par de arquivos de cabeçalho.

A fonte expõe as seguintes funções:

- void spi\_init (uint8 spi\_no)
- void spi\_init\_gpio (uint8 spi\_no, uint8 sysclk\_as\_spiclk)
- void spi\_clock (uint8 spi\_no,  
                  uint16 prediv,  
                  uint8 cntdiv)

Página 143

## Página 144

Para calcular a taxa de clock efetiva, pegue a frequência da CPU (80 MHz) e divida-a por preDiv. Isso dá uma frequência básica. Em seguida, dividimos isso por

- void spi\_tx\_byte\_order (uint8 spi\_no, uint8 byte\_order)
- void spi\_rx\_byte\_order (uint8 spi\_no, uint8 byte\_order)
- uint32 spi\_transaction (uint8 spi\_no,  
                  uint8 cmd\_bits,  
                  uint16 cmd\_data,  
                  uint32 addr\_bits,  
                  uint32 dout\_bits,  
                  uint8 dout\_data,  
                  uint32 din\_bits,  
                  uint32 dummy\_bits)

Além disso, as seguintes macros também são fornecidas:

- spi\_busy (spi\_no)
- spi\_txd (spi\_no, bits, dados)
- spi\_tx8 (spi\_no, dados)
- spi\_tx16 (spi\_no, dados)
- spi\_tx32 (spi\_no, dados)
- spi\_rxd (spi\_no, bits)
- spi\_rx8 (spi\_no)
- spi\_rx16 (spi\_no)
- spi\_rx32 (spi\_no)

Nas funções anteriores, SPI e HSPI são os valores permitidos para spi\_no .

Veja também:

- GitHub: [MetalPhreak / ESP8266\\_SPI\\_Driver](#)

## Trabalhando com serial

Existem dois UARTs no sistema, conhecidos como UART0 e UART1. UART0 tem seu próprio pinos TX e RX dedicados enquanto UART1 é multiplexado com GPIO2. UART1 é produzido

apenas e, portanto, tem apenas uma linha TX.

A interface serial para o ESP8266 pode ser inicializada com uma chamada para a função `uart_init()`.

Página 144

## Página 145

Por exemplo

```
uart_init(BIT_RATE_115200, BIT_RATE_115200);
```

Para escrever uma string na porta serial, podemos então usar `os_printf()`. Este tem o mesmo formato como um `printf`, mas grava na porta serial.

Para trabalhar com UART, você deve incluir `uart.c`, `uart.h` e `uart_register.h` arquivos de `samples / driver_lib`. Em seu aplicativo, você deve incluir `"driver / uart.h"`.

Para transmitir dados usando UART0, temos a função chamada `uart0_tx_buffer()` que aceita um ponteiro para dados e um comprimento e os transmite.

Dentro do SDK existe um buffer de transmissão. Porque a transmissão UART é tipicamente operação lenta, aplicativos que desejam transmitir dados têm seus dados armazenados no TX buffer que é então drenado pela transmissão ao longo do tempo. Os dados gravados no UART são garantido ser escrito na ordem em que foi fornecido. O buffer TX deve ficar cheio, nenhum dado novo pode ser aceito.

Da mesma forma, os dados recebidos pelo ESP8266 através do UART são colocados em um buffer de receção. O aplicativo em execução no ESP8266 deve receber os dados em tempo hábil. Se o buffer RX fica cheio, não há lugar para colocar novos dados e esses novos dados serão descartado. Em termos de UART, os buffers TX e RX são denominados "FIFO", que significa primeiro a entrar, primeiro a sair. Os buffers têm 128 bytes cada (128 bytes para TX e um segundo Buffer de 128 bytes para RX).

Para descobrir quantos bytes estão nas várias filas, temos que ir para um nível bem baixo.

Por exemplo, o número de bytes na fila TX é dado por:

```
(READ_PERI_REG(UART_STATUS(uart_no)) >> UART_TXFIFO_CNT_S) & UART_TXFIFO_CNT;
```

enquanto o número de bytes na fila RX é dado por:

```
(READ_PERI_REG(UART_STATUS(UART0)) >> UART_RXFIFO_CNT_S) & UART_RXFIFO_CNT;
```

Veja também:

- [Conectando-se ao ESP8266](#)
- [Conversores USB para UART](#)
- [Erro: fonte de referência não encontrada](#)
- [Erro: fonte de referência não encontrada](#)
- [Erro: fonte de referência não encontrada](#)
- [os\\_printf](#)

Página 145

**Página 146****ESP8266 Gerenciamento de tarefas**

Imagine que desejamos que uma tarefa seja executada para nós de forma assíncrona. O que podemos querer a fazer é postar que desejamos que isso aconteça e depois continuar com o nosso negócio. Quando nós são feitas e renunciamos ao controle de volta para o sistema operacional, assumimos que a tarefa será eventualmente começar a executar. Esta é a função fornecida pelas funções de tarefa do ESP8266. Existem duas funções que nos interessam. O primeiro é chamado `system_os_task()` que configura um processador de tarefas.

Quando desejamos postar que uma tarefa é elegível para iniciar, podemos usar a segunda função chamado `system_os_post()` que posta uma mensagem.

A função de tarefa que registramos será "invocada" em algum ponto após a postagem solicitação e receberá os parâmetros fornecidos no post. A prioridade identifica o prioridade relativa de dois postos que foram emitidos. Aquele com a maior prioridade irá execute primeiro.

É importante notar que apenas **três** prioridades são permitidas, que são 0, 1 e 2 com 0 tendo a prioridade mais baixa. Também é importante observar que só pode haver **um** manipulador para cada registro de tarefa por prioridade. Então, se executarmos `system_os_task()` duas vezes usando o mesma prioridade em ambos os casos, apenas o último é lembrado e será executado quando uma tarefa com essa prioridade é postada.

Com esse pano de fundo, qual é a finalidade desse conjunto de funções? Por que nos importaríamos sobre isso?

Nota: A biblioteca Arduino para ESP usa o conjunto de tarefas de prioridade 0.

Aqui está um exemplo de um gerenciador de tarefas simples.

```
void taskHandler(os_event_t* event) {
    switch (evento->sig) {
        caso 1:
            pausa;
        caso 2:
            pausa;
    }
}

os_event_t *taskQueue;
taskQueue = (os_event_t *) malloc(sizeof(os_event_t) * TASK_QUEUE_LEN);
system_os_task(taskHandler, USER_TASK_PRIO_1 taskQueue, TASK_QUEUE_LEN);

system_os_post(USER_TASK_PRIO_1, 1, (os_param_t) "Olá");
```

Veja também:

- [Erro: fonte de referência não encontrada](#)
- [Erro: fonte de referência não encontrada](#)

**Página 147****Timers e tempo**

Dentro do nosso código, podemos desejar atrasar por um período de tempo. Podemos usar o função `os_delay_us()` para suspender o processamento por um determinado período medido em microssegundos. Existem 1000 microssegundos em um milissegundo e 1000 milissegundos em um segundo.

Podemos configurar um cronômetro para ser chamado periodicamente com uma granularidade de retorno de chamada de milissegundos. Uma estrutura de dados chamada `os_timer_t` mantém o estado do cronômetro.

Podemos definir a função do usuário a ser chamada quando o cronômetro dispara usando o função `os_timer_setfn()`. Observe que só podemos definir a função de retorno de chamada quando o cronômetro está desarmado.

Quando estiver pronto, podemos armar o cronômetro para que comece a funcionar e dispare quando estiver pronto. Nós fazemos isso usando a função `os_timer_arm()`.

O sinalizador de repetição indica se o cronômetro deve reiniciar depois de disparar.

Podemos suspender ou cancelar o disparo do cronômetro usando `os_timer_disarm()`.

Aqui está um exemplo:

```
os_timer_t myTimer;

void timerCallback (void * pArg) {
    os_printf ("Assinale!");
} // Fim do timerCallback

void user_init (void) {
    uart_init (BIT_RATE_115200, BIT_RATE_115200);
    os_timer_setfn (& myTimer, timerCallback, NULL);
    os_timer_arm (& myTimer, 1000, 1);
} // Fim do user_init
```

Outro aspecto de trabalhar com o tempo é o cálculo e a medição do tempo. O função `system_get_time()` retorna um valor inteiro não assinado de 32 bits (unit32) que é o microssegundos desde que o dispositivo foi inicializado. Este valor será acumulado após 71 minutos.

E se precisarmos de uma granularidade de tempo menor que um microssegundo? Espero que você não preciso disso com frequência ... no entanto, algumas soluções, como trabalhar com os LEDs WS2812, na verdade, requerem precisão ultrafina.

Uma solução possível é descer para a programação em linguagem assembly. Existe um registo especial gerido pelo ESP8266 denominado "contagem" que mede ciclos de operação. O valor disso é incrementado cada vez que um ciclo operacional completa.

O valor deste registro pode ser recuperado com o seguinte código C:

Página 147

---

## Página 148

```
static inline uint32_t getCycleCount () {
    uint32_t ccount;
    __asm__ __volatile__ ("rsr% 0, conta": "=a" (conta conta));
    return ccount;
}
```

Este fragmento usa o assembler in-line para transformar uma declaração de linguagem assembly em sua instrução operacional correspondente.

Veja também:

- [Erro: fonte de referência não encontrada](#)
- [os\\_timer\\_arm](#)
- [os\\_timer\\_disarm](#)
- [os\\_timer\\_setfn](#)

### Trabalhando com a memória

Ao trabalhar em C, você deve pensar em termos de memória de computador. A quantidade de a RAM disponível provavelmente será inferior a 45 KB.

Podemos alocar memória usando `os_malloc()` ou `os_zalloc()`. A primeira função aloca e retorna a memória e o segundo faz exatamente o mesmo, mas zera a memória antes de retornar. Quando sua lógica não precisa mais da memória, ela pode retorná-la para o heap com `os_free()`. Para determinar quanto tamanho de heap está disponível, podemos chamar `system_get_free_heap_size()`. Assim que tivermos o ponteiro de memória, podemos começar a manipule-o por meio de uma série de comandos de memória. O comando `os_memset()` irá definir um bloco de memória com um valor específico. O `os_memcpy()` irá copiar um bloco de memória para um bloco diferente. A função `os_bzero()` irá definir os valores de um bloco de memória para zero.

A memória no ESP8266 é composta por vários componentes. Nós temos:

- dados
- rodata
- bss
- amontoar

Os valores destes podem ser encontrados através da função `system_print_meminfo()`.

Quando o ESP8266 precisa ler uma instrução da memória para executá-la, essa instrução pode vir de um de dois lugares. A instrução pode ser em flash memória (também chamada de irom) ou pode estar na RAM (também chamada de iram). Demora menos para o processador para recuperar a instrução da RAM do que do flash. Acredita-se que uma instrução de busca do flash leva quatro vezes mais do que a mesma instrução

Página 148

### Página 149

obtido da RAM. No entanto, no ESP8266 há muito menos RAM do que flash. O que isto significa é que é muito mais provável que você fique sem RAM antes de executar fora do flash. Ao escrever aplicativos normais, não devemos nos fixar em ter instruções em RAM em vez de flash para o benefício de desempenho. A execução as velocidades do ESP8266 são tão rápidas que, se o custo de recuperar uma instrução da RAM é incrivelmente rápido, em seguida, recuperar uma instrução do flash mais lento **ainda** é incrivelmente rápido.

No entanto, existem certas classes de instruções que podemos querer colocar na RAM em vez de flash. Exemplos disso são manipuladores de interrupção, onde o tempo gasto em estes devem ser sempre os mais curtos possíveis e também funcionar que gravam no flash.

Quando definimos funções C, podemos adicionar um atributo com o nome de `ICACHE_FLASH_ATTR`. O que isso faz é colocar esta função no endereço da memória flash espaço em oposição à RAM. Especificamente, sinalizando uma função com `ICACHE_FLASH_ATTR` marca-o como estando na seção "`.irom0.text`" do código.

**Observação :** de uma perspectiva técnica bruta, `ICACHE_FLASH_ATTR` é um `#define` que mapeia para:

```
_attribute__((seção(".irom0.text")))
```

Uma das áreas que ainda não discutimos é como a memória é preenchida e usada quando um ESP8266 é inicializado. Existem dois arquivos normalmente carregados em flash. O primeiro é no deslocamento 0x00000 e contém os dados que serão carregados na RAM pela ROM boot-loader baseado. Esses dados contêm uma série de seções e endereços na RAM de

onde os dados serão carregados. O segundo arquivo binário é comumente carregado no flash em 0x40000 . Ele contém os dados binários de uma seção chamada .irom0.text que contém código. O código de RAM carregado deve corresponder ao local onde esses dados armazenados em flash são endereçados.

Para aqueles interessados em detalhes de baixo nível, o formato da memória gravada no flash arquivos foram decodificados. Acredita-se que a baixa memória se pareça com:

```
struct rom_header {
    magia uint8;
    uint8 sect_count;
    uint8 flags1;
    uint8 flags2;
    uint32 entry_addr;
};
```

A propriedade mágica é uma constante de 0xe9 . O sect\_count contém o número de seções para carregar na memória ram. Isso não incluirá a seção irom.text . As bandeiras1 e flags2 são usados para indicar o tamanho do flash, a taxa de clock e o modo IO. Finalmente, entry\_addr é o ponto de entrada para iniciar a execução do código fornecido pelo usuário.

Imediatamente após o cabeçalho, estão as entradas da seção (deve haver sect\_count deles) onde cada entrada é:

Página 149

---

## Página 150

```
struct sect_header {
    endereço uint32;
    comprimento uint32;
}
```

Os endereços devem estar dentro do espaço de endereço .iram começando em 0x40100000 . Após cada um dos cabeçalhos é um valor de soma de verificação. Uma soma de verificação é calculada a partir de cada um dos seções e validado que corresponde ao que deveria estar presente.

Todo o flash também é mapeado para o espaço de endereço 0x40200000 .

O segundo arquivo contém os dados da seção irom.text . Por padrão, o espaço de endereço para esta seção é 0x40240000, o que significa que deve ser gravada para flash em 0x40000 .

Um excelente mapa da memória ESP8266 está sendo mantido no ESP8266 [Wiki](#).

A essência disso está aqui:

Endereço	Tamanho	Notas
0x0000 0000 <		Não pode ser lido.
0x2000 0000 <		Não mapeado
0x3FF0 0000 <		E / S mapeada em memória.
0x3FF1 0000 <		Não mapeado.
0x3FFE 8000 <80K - 81920 (0x14000)		RAM de dados do usuário (dram)
0x3FFF C000 <16K - 16384 (0x4000)		RAM de dados do sistema ETS.
0x4000 0000 <		ROM interna
0x4010 0000 <32K - 32768 (0x8000)		RAM de instrução (iram / sram)
0x4010 8000 <		Não mapeado ou desconhecido
0x4020 0000 <Máx. 1024K (1M) - 1048576 (0x100000)		SPI Flash
0x4030 0000 <		Não mapeado ou desconhecido

Agora vem uma discussão que demorei muito, muito tempo para compreender. É a relação entre a memória flash e o espaço de endereço do ESP8266. Se nós

examinar a tabela anterior, parece que vemos que a memória Flash está mapeada no Espaço de endereço ESP8266 no endereço 0x4020 0000 . Isso é fundamental e vital para entender.

Se leremos desse endereço em diante, na verdade estamos lendo da memória flash.

Vamos pensar no ESP-12 que possui 512K de memória flash. Em hexadecimal, é 0x8000 bytes. Isso significa uma faixa de endereço de 0x4020 0000 a 0x4027 FFFF . Agora vamos considere as ferramentas de flashing ESP, como “ esptool ”. Isso também aceita um endereço em qual carregar o flash ... mas como esse endereço se relaciona com o espaço de endereço em tempo de execução do

Página 150

## Página 151

ESP8266? A resposta é que se escrevermos para o endereço 0x0 0000 do flash, em tempo de execução aparecem em 0x4020 0000 . Então, na verdade, o que temos é o seguinte:

Agora respondemos a mais um quebra-cabeça. Quando vinculamos os arquivos objeto e produzimos uma imagem binária ... parte da função do vinculador é criar o layout de endereço final onde o executável finalmente residirá. Na prática, quando executamos o vinculador, fornecemos um arquivo de controle do vinculador denominado " eagle.app.v6.ld ". Este é fornecido pela Espressif. Se olharmos no arquivo *padrão* , encontramos o seguinte no início do arquivo:

```
MEMÓRIA
{
    dport0_0_seg:                      org = 0x3FF00000, len = 0x10
    dram0_0_seg:                        org = 0x3FFE8000, len = 0x14000
    iram1_0_seg:                        org = 0x40100000, len = 0x8000
    irom0_0_seg:                        org = 0x40240000, len = 0x32000
}
```

Agora ... olhe de perto ... porque levei uma eternidade para entender o que estou fazendo para te dizer. Observe o endereço onde o segmento irom0 começa. Começa em 0x40240000... isto é 256K **no** intervalo de endereços de 512K! Colocando de outra forma, nossos executáveis não podem usar o endereço inteiro com essas configurações padrão. Por que o Espressif definir isso? Acredita-se que a resposta seja porque eles desejam fornecer Over The Air (OTA) capacidade de atualização e deseja permitir que você tenha uma cópia em execução do seu aplicativo e uma nova cópia em voo seja carregada. Isso significa que efetivamente, 256 K para o atual versão e 256K para a nova versão que está sendo carregada. Se tentássemos usar o todo espaço de endereço e durante uma atualização algo deu errado, não temos nada para retroceder para.

---

**Página 152**

Se você, como eu, quiser usar todo o espaço de endereço, a resposta é simples. Mudar a entrada correspondente no arquivo "eagle.app.v6.ld".

Uma última ruga. Acredita-se que os últimos 16K de flash devam ser reservados para Armazenamento do Espressif SDK para itens como o último SSID e senha usados. Não suponha que você pode usar esse intervalo.

Por padrão, os seguintes itens em um arquivo ELF vão para o local flash 0x0 0000

- .texto
- .dados
- .rodata
- .iram0.text

As seções a seguir vão para o local do flash 0x4 0000

- .texto

Veja também:

- [os\\_memset](#)
- [os\\_memcpy](#)
- [os\\_memcmp](#)
- [os\\_malloc](#)
- [os\\_zalloc](#)
- [os\\_free](#)
- [Erro: fonte de referência não encontrada](#)
- [Erro: fonte de referência não encontrada](#)
- [O processo de inicialização ESP8266](#)
- [esptool.py](#)
- [esptool-ck](#)
- [gen\\_appbin.py](#)

---

**Trabalhando com memória flash**

A memória flash fornece um repositório não volátil de informações que sobrevivem a um poder ciclo do dispositivo.

Os dados contidos na memória flash são armazenados em unidades de setores com tamanho de 4096 bytes. Para escrever dados que podemos chamar de `spi_flash_write`. Para ler os dados, chamamos `spi_flash_read`.

Uma vez que a gravação em flash é realizada em unidades de 4096 bytes, não podemos alterar um único byte apenas sobreescrivendo-o, em vez disso, devemos recuperar todo o setor, apagar o setor e, em seguida, escreva de volta ao setor com o conteúdo alterado. Isso pode levar algum tempo para completa e por causa disso, podemos descobrir que uma falha é mais provável de ocorrer (por exemplo, um perda de potência). Se ocorrer uma falha depois de termos apagado um setor ou durante a reescrita de setor, deve ficar imediatamente claro que vamos resultar em uma corrupção de dados.

As leituras e gravações de dados devem ter 4 bytes alinhados dentro do flash.

O ESP8266 deve ser instruído sobre o tamanho da memória flash disponível.

Tentar usar endereços de memória flash que diferem do tamanho esperado de flash a memória disponível pode resultar em resultados inesperados.

Ao usar esptool.py , o sinalizador --flash\_size pode ser fornecido. Para esptool-ck, o sinalizador correspondente é -bz .

Veja também:

- [spi\\_flash\\_get\\_id](#)
- [spi\\_flash\\_erase\\_sector](#)
- [spi\\_flash\\_read](#)
- [spi\\_flash\\_set\\_read\\_func](#)
- [system\\_param\\_save\\_with\\_protect](#)
- Cesanta: [ESP8266\\_flash e alinhamento](#)
- [system\\_param\\_load](#)

### **Modulação por largura de pulso - PWM**

A ideia por trás da modulação por largura de pulso é que podemos pensar em pulsos regulares de saída sinais como informação de codificação em quanto tempo o sinal é mantido alto. Vamos imaginar que temos um período de 1HZ (uma coisa por segundo). Agora, vamos supor que aumentamos o

tensão de saída a um nível de 1 por  $\frac{1}{2}$  de segundo no início do período. Isso seria nos dê uma onda quadrada que começa alto, dura 500 milissegundos e depois cai baixo pelos próximos 500 milissegundos. Isso se repete no futuro. A duração que o

pulso é alto em relação ao período nos permite codificar um valor analógico em digital

sinais. Se o pulso for 100% alto para o período, o valor codificado seria 1,0.

Se o pulso for 100% baixo para o período, o valor codificado seria 0,0. Se o pulso está ligado por "n" milissegundos (onde n é menor que 1000), então o valor codificado seria  $n / 1000$ .

Normalmente, a duração de um período não é um segundo, mas muito, muito menor, permitindo-nos produza muitos valores diferentes muito rapidamente. A proporção do sinal ligado para o período é chamado de "ciclo de trabalho". Esta técnica de codificação é chamada de "Modulação de largura de pulso" ou "PWM".

Existem vários propósitos para o PWM. Alguns são codificadores de dados de saída. Um O propósito comumente visto é controlar o brilho de um LED. Se aplicarmos o máximo voltagem para um LED, é maximamente brilhante. Se aplicarmos  $\frac{1}{2}$  da tensão, é cerca de  $\frac{1}{2}$  da brilho. Ao aplicar um sinal PWM de período rápido à entrada de um LED, o ciclo de trabalho torna-se o brilho do LED. A maneira como isso funciona é com tensão plena ou sem tensão é aplicada ao LED, mas como o período é tão curto, a tensão "média" ao longo do tempo segue o ciclo de trabalho e mesmo que o LED esteja piscando, aceso ou apagado, é assim rápido que nossos olhos não conseguem detectar e tudo o que vemos é a mudança aparente de brilho.

Para o ESP8266, o período do PWM pode variar de 1 milissegundo a 10 milissegundos. Esta é uma frequência de 1KHz a 100Hz. A resolução do ciclo de trabalho é até 45 nanosegundos, que são 14 bits de dados de resolução. O dispositivo fornece suporte para até 8 canais PWM, onde cada canal pode ser associado ao seu próprio pino e ciclo de trabalho. O período é o mesmo para todos os canais PWM.

Para começar a usar o suporte PWM ESP8266, é necessária uma chamada para `pwm_init()` que configura quais pinos devem ser usados para PWM e para quais canais. Uma chamada para esta função também configura um período inicial e um ciclo de trabalho. Uma chamada para `pwm_start()` pode então ser feita para iniciar

as saídas PWM. O período de PWM como um todo e os ciclos de trabalho de cada canal pode ser alterado usando as funções `pwm_set_period()` e `pwm_set_duty()`.

As funções do ESP8266 PWM utilizam o temporizador de hardware. Como tal, você pode ter PWM suporte ou utilize o cronômetro de hardware para seus usos ... mas não ambos.

Para utilizar as funções ESP8266 SDK PWM, você deve vincular seu aplicativo com `libpwm.a`.

Veja também:

- Wikipedia: [Modulação de largura de pulso](#)
- [Amostra Espressif](#)
- [pwm\\_init](#)
- [pwm\\_start](#)
- [pwm\\_set\\_duty](#)
- [pwm\\_get\\_duty](#)
- [pwm\\_set\\_period](#)
- [pwm\\_get\\_period](#)

## Conversão analógica para digital

A conversão analógica para digital é a capacidade de ler um nível de tensão de um pino entre 0 e algum valor máximo e converter essa tensão analógica em uma representação digital.

A variação da tensão aplicada ao pino mudará o valor lido. O ESP8266 tem um conversor analógico para digital integrado com uma resolução de 1024 valores distintos. o que isso significa que 0 volts irá produzir um valor digital de 0, enquanto a tensão máxima irá produzir um valor digital de 1023 e as faixas de voltagem entre eles produzirão um valor digital correspondentemente escalado.

Para ler o valor digital da tensão analógica, a função chamada `system_adc_read()` deve ser chamado. O pino no ESP8266 físico a partir do qual a tensão é lida é chamado TOUT e não serve a nenhum outro propósito.

A faixa de entrada no pino é de 0 V a 1 V. Isso implica que a tensão de entrada para o ADC não pode ser a tensão máxima usada para alimentar o próprio ESP8266 (3,3 V). Então nós precisarão usar um circuito divisor de tensão.

Página 154

---

## Página 155

A fórmula para mapear isso é:

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in}$$

Como sabemos que  $V_{out}$  vai para 1V e  $V_{in}$  é 3V e escolhemos  $R_2$  como 10K, encontramos:

$$R_1 = \frac{R_2 \cdot V_{in}}{V_{out}} - R_2$$

e pelos nossos valores:

$$R_1 = \frac{10000 * 3,3}{1,0} - 10000 = 23000$$

Um resistor comum de 22K funcionará bem.

Aqui está um exemplo. O que este exemplo faz é imprimir o valor lido do ADC a cada segundo.

```
os_timer_t myTimer;
```

```
void timerCallback (void * pArg) {
    uint16 adcValue = system_adc_read ();
    os_pprintf ("adc =% d \n", adcValue);
} // Fim do timerCallback

void user_init (void) {
    uart_init (BIT_RATE_115200, BIT_RATE_115200);
    os_timer_setfn (& myTimer, timerCallback, NULL);
    os_timer_arm (& myTimer, 1000, 1);
} // Fim do user_init
```

Se construirmos em uma placa de ensaio um circuito que inclui um resistor dependente de luz, tal como o seguinte:

Página 155

## Página 156

Então, quando mudamos a quantidade de luz que incide sobre o resistor, podemos ver os valores muda conforme os dados são gravados no log de saída. Isso pode ser usado para desencadear uma ação (para exemplo) quando escurece.

Pergunta em aberto: Qual é a taxa de amostragem do ADC?

Veja também:

- [Erro: fonte de referência não encontrada](#)
- Wikipedia: [Divisor de tensão](#)

### Modos de hibernação

Se o dispositivo ESP8266 está constantemente ligado, ele está constantemente consumindo corrente. Se a fonte de energia é ilimitada, então isso não precisa ser necessariamente um problema, no entanto, quando funcionando com baterias ou outro suprimento finito, podemos precisar minimizar o consumo. Um maneira de conseguir isso é suspender a operação do dispositivo quando não estiver em uso. Quando com o aparelho suspenso, a noção é que o consumo será reduzido. Existem três modos de dormir definidos. Eles são chamados de sono moderno, sono leve e sono profundo.

Página 156

---

## Página 157

Observando a tabela a seguir, podemos ter uma noção das habilidades de cada um desses três modos:

Função	Modem	Luz	Profundo
Wi-fi	fora	fora	fora
Relógio do Sistema	sobre	fora	fora
Relógio de tempo real	sobre	sobre	sobre
CPU	sobre	pendente	fora
Consumo atual	15mA	0,5mA	20µA

O modo de espera do modem só pode ser usado quando o ESP8266 está no modo de estação conectado ao um ponto de acesso. A aplicação deste modo é quando o ESP8266 precisa ainda executa o trabalho, mas minimiza a quantidade de transmissões sem fio.

O modo de suspensão leve é o mesmo que a suspensão do modem, mas neste caso os relógios serão suspenso.

No modo de hibernação, o dispositivo está realmente hibernando. Nem as atividades da CPU nem do WiFi levam Lugar, colocar. O dispositivo está desligado para todos os efeitos ... com uma exceção ... ele pode despertar em um intervalo regular especificado.

Para entrar no modo de sono profundo, podemos chamar `system_deep_sleep()`. Isso pode ser fornecido com um tempo de suspensão. O dispositivo irá hibernar e após o intervalo ter decorrido, o dispositivo será reativado. Além de ter um cronômetro, também podemos acordar de uma profunda dormir alternando o valor de um sinal em um pino.

Podemos controlar em qual modo o dispositivo está chamando `wifi_set_sleep_type()`.

### Cronômetro de vigilância

O ESP8266 é um processador de thread único. Isso significa que ele só pode fazer uma coisa por tempo, pois não há threads paralelos que podem ser executados simultaneamente uns com os outros.

Uma implicação disso é que quando o sistema operacional dá o controle ao seu aplicativo, ele não consegue controle de volta até o momento em que você abandone-o explicitamente. No entanto, isso pode causar problemas. O ESP8266 é principalmente um dispositivo WiFi e TCP / IP que espera ser capaz para receber e transmitir dados, bem como responder a eventos assíncronos dentro de um prazo maneiras. Por exemplo, se o seu dispositivo ESP8266 estiver conectado a um ponto de acesso e o ponto de acesso deseja validar se você ainda está conectado, ele pode transmitir um pacote

para você e espera uma resposta. Você não tem controle sobre quando isso vai acontecer. Se seu próprio programa aplicativo tem controle sobre a execução no momento em que o pedido chegar, essa solicitação não será respondida até que você devolva o controle ao OS. Enquanto isso, o ponto de acesso pode estar esperando uma resposta dentro de alguns período de tempo predeterminado e, caso não receba resposta dentro desse intervalo, pode

---

## Página 158

suponha que você tenha se desconectado. Para evitar tais circunstâncias, seu aplicativo o código deve devolver o controle ao sistema operacional em tempo hábil. É recomendado que seu código retorna o controle dentro de 50 milissegundos de ganhar o controle. Se demorar mais, você corre o risco de solicitações para o seu dispositivo expirar.

Se o seu próprio código não retornar o controle ao sistema operacional, o sistema operacional deve assumir que as coisas estão dando errado. Como tal, tem um temporizador a que chamamos "cão de guarda". Quando o controle é dado ao seu próprio código, o cronômetro de watchdog começa a funcionar. Se você não voltou controle de volta ao sistema operacional no momento em que o cronômetro de watchdog chega a zero, é preciso em suas próprias mãos. Explicitamente, o que ele faz é reiniciar o dispositivo. Isso pode soar como uma ação muito drástica, mas o pensamento é que é melhor fazer isso e esperar que seja o que for foi bloqueado agora está desbloqueado do que apenas sentar lá "morto".

Os relatórios afirmam que o cronômetro de watchdog pode ter cerca de 1 segundo (1000 milissegundos). No entanto, em meus testes, descobri que o cronômetro dispara em cerca de 3,2 segundos (3200 ms).

Uma função chamada `system_soft_wdt_stop()` para o cronômetro de watchdog ... ou pelo menos um dos eles. Parece haver **dois** temporizadores. Um está no software, o outro no hardware. Esta função pára o temporizador do software. Ele pode ser reiniciado com `system_soft_wdt_restart()` ... no entanto, um segundo temporizador denominado temporizador de watchdog de hardware irá disparar após cerca de 8 segundos e não parece capaz de ser preso. Uma nova chamada introduzida no SDK 1.3 é chamado `system_soft_wdt_feed()`. Infelizmente, a documentação sobre isso é excessivamente pobre. Os melhores relatórios que temos até agora sobre o que ele faz é que, quando o chamamos, o tempo de watchdog é redefinido para seu ponto inicial e começa a diminuir novamente. Eu não sou bem certeza do valor disso, visto que já temos API para parar e reiniciar o cronômetro. Esperançosamente, no futuro, podemos obter conhecimento adicional para esclarecer qualquer mistério que podem estar espreitando por dentro.

Veja também:

- [system\\_soft\\_wdt\\_stop](#)
- [system\\_soft\\_wdt\\_restart](#)
- [Erro: fonte de referência não encontrada](#)

### Controle de cessão

Acabamos de descrever a noção de ter que devolver o controle ao sistema operacional em ordem para que ele desempenhe suas funções de manutenção da casa. A maneira como fazemos isso é simplesmente retornar do retorno de chamada que o sistema operacional nos invocou. Se pensarmos em como um ESP8266 programa funcionar, veremos que, para abrirmos mão do controle de volta ao sistema operacional, o OS deve ter nos chamado em primeiro lugar. Portanto, faz sentido para nós voltarmos controle em um ponto posterior. No entanto, se retornarmos o controle, (obviamente) perdemos todo o estado (variáveis) que existiam quando retornamos.

---

**Página 159**

Agora vamos introduzir um conceito chamado "cedência". A ideia por trás da rendição é que em vez de nosso aplicativo retornar o controle ao sistema operacional, o que podemos fazer é retornar ao sistema operacional enquanto, ao mesmo tempo, manter o contexto de onde estamos a execução atual. Quando o sistema operacional completa uma rodada de manutenção, o que ele pode então fazer é "retornar" para onde estávamos quando solicitamos a ocorrência de um rendimento.

É algo complicado de implementar, mas felizmente Ivan Gorkhotov conseguiu essa tarefa para nós e podemos alavancar o que ele já construiu.

Para usar isso:

1. Incluir cont.h
2. Crie um global de " cont\_t g\_cont \_\_attribute\_\_ ((alinhado (16))); "
3. Em user\_init chamado " cont\_init (& g\_cont); "
4. Registre um processador system\_task () .

Quando quisermos agendar algum código para execução, poste uma tarefa.

```
void esp_schedule () {
    system_os_post (TASK_PRIORITY, 0, 0);
}

static void taskHandler (os_event_t * events) {
    cont_run (& g_cont, someFunction);
    if (cont_check (& g_cont) != 0) {
        os_printf ("Overflow detectado \r \n");
        abortar();
    }
}
```

### **Segurança**

O ESP8266 tem a capacidade de armazenar a senha usada para se conectar ao ponto de acesso em memória. Isso significa que se alguém comprometer fisicamente o dispositivo (ou seja, roubar) então eles poderiam, em princípio, despejar a memória flash e recuperar sua senha.

Você pode optar por não armazenar a senha em cache no flash, mas em vez disso, ter seu aplicativos "decodificam" uma versão codificada que é salva na memória flash ... isso impediria uma recuperação óbvia por meio de uma captura de memória simples. A codificação esquema poderia ser um simples XOR contra um número mágico (codificado ou seu próprio Endereço MAC).

### **Mapeamento do Arduino**

Sem discussão, o Arduino se tornou o microprocessador de maior sucesso ambiente de programação atualizado. Existem toneladas e toneladas de esboços existentes em

---

**Página 160**

existência e não nos esqueçamos da riqueza das bibliotecas. Existem ferramentas e utilitários para compilar e executar esboços do Arduino em ESP8266s. E se em vez disso quiséssemos portar aqueles esboços do Arduino para o código ESP8266 nativo? Podemos encontrar mapeamentos entre o APIs do Arduino e as APIs ESP8266 correspondentes?

Arduino	ESP8266
digitalWrite (pino, valor)	GPIO_OUTPUT_SET (pino, valor)
digitalRead (pin)	GPIO_INPUT_GET (pin)
atraso (ms)	os_delay_us (ms * 1000) Nota: ms <= 65535
delayMicroseconds (us)	os_delay_us (nós)
millis ()	system_get_time () / 1000

De uma perspectiva funcional, aqui estão algumas comparações entre um Arduino e um ESP8266:

	ESP8266	Arduino (Uno)
GPIOs	17 (menos tipicamente exposto)	14 (20 incluindo analógico)
Entrada analógica	1	6
Canais PWM	8	6
Velocidade do relógio	80 MHz	16MHz
Processador	Tensilica	Atmel
SRAM	45 KB	2 KBytes
Instantâneo	512 KB ou mais (separado)	32 KB (no chip)
Tensão operacional	3,3 V	5V
Corrente máxima por E / S	12mA	40mA
UART (hardware)	1 ½	1
Networking	Construídas em	Separado
Documentação	Pobre	Excelente
Maturidade	Cedo	Maduro

Nota: Como o Arduino não tem rede nativa, não há outras comparações de rede capacidade foram incluídos acima. Lembre-se de que, neste momento, quando alguém está usando um ESP8266, as chances são altas porque você **precisa de** acesso à rede.

## Sistema de arquivos Spiffs

O SPI Flash File System (SPIFFS) é um mecanismo de sistema de arquivos destinado a dispositivos incorporados. Uma implementação é fornecida com o FreeRTOS ESP8266 SDK.

Página 160

---

## Página 161

Qual é o tamanho da página do flash ESP8266? Qual é o tamanho do bloco do flash ESP8266?

Quando uma chamada SPIFFS API é feita, uma resposta zero ou positiva indica sucesso enquanto um valor <0 indica um erro. A natureza do erro pode ser recuperada por meio do

Chamada SPIFFS\_errno () .

A implementação SPIFFS não acessa diretamente a memória flash. Em vez disso, um área funcional chamada de camada de abstração de hardware ("hal") fornece este serviço. UMA integração SPIFFS requer que três funções sejam criadas com o seguinte assinaturas:

```
s32_t (* spiffs_read) (u32_t addr, u32_t size, u8_t * dst)
s32_t (* spiffs_write) (u32_t addr, u32_t size, u8_t * src)
s32_t (* spiffs_erase) (u32_t addr, u32_t size)
```

Se forem bem-sucedidos, o código de retorno deve ser SPIFFS\_OK (0). Em um ESP8266, estes irão

mapear para as APIs flash SPI.

O sistema de arquivos SPIFFS pode ser de natureza hierárquica, de modo que contém ambos diretórios e arquivos, mas parece que na realidade não é. Existe apenas um diretório chamada de raiz. O diretório raiz é " / ". Para determinar os membros de um diretório, nós pode abrir um diretório para leitura com a API SPIFFS\_opendir () e, quando estivermos concluído, feche a operação de leitura com uma chamada de API SPIFFS\_closedir () . Podemos andar através das entradas do diretório com chamadas para SPIFFS\_readdir () .

Por exemplo:

```
spiffs_DIR spiffsDir;
SPIFFS_opendir (& fs, "/", & spiffsDir);
struct spiffs_dirent spiffsDirEnt;
while (SPIFFS_readdir (& spiffsDir, & spiffsDirEnt)! = 0) {
    printf ("Obteve uma entrada de diretório:% s \ n", spiffsDirEnt.name);
}
SPIFFS_closedir (& spiffsDir);
```

Para criar um arquivo, podemos usar a API SPIFFS\_open () fornecendo um SPIFFS\_CREAT bandeira.

Veja também:

- Github: [pellepl / spiffs](https://github.com/pellepl/spiffs)

## APIs TCP / IP de parceiros

Se o ESP8266 pode atuar como uma extremidade de uma conexão TCP / IP, outra coisa deve agir como o outro (é claro, não há nada para impedir que dois ESP8266s se comuniquem entre eles mesmos). Aqui, examinamos algumas tecnologias que permitem aos parceiros interagir com o ESP8266 através do protocolo TCP / IP.

Página 161

---

## Página 162

Para o protocolo TCP / IP, a API de programação desenvolvida originalmente para a plataforma Unix e escrito em C era chamado de "soquetes". A noção de um soquete é que logicamente representa um ponto final de uma conexão de rede. Um remetente de dados envia dados por meio de o soquete e o receptor de dados recebem dados por meio do soquete. O implementação do próprio "soquete" é fornecida pelas bibliotecas, mas a noção lógica de o soquete permanece. Você se verá trabalhando com uma "instância" de um soquete e você deve pensar nisso como um tipo de dados opaco que se refere a um link de comunicação.

Sockets continua sendo a API principal e está presente na maioria dos idiomas. Aqui nós discuta algumas das variantes para algumas das linguagens mais comuns.

## Soquetes TCP / IP

A API de sockets é uma interface de programação para trabalhar com rede TCP / IP. Isto é provavelmente a API mais familiar para programação de rede. Os fluxos de rede TCP / IP entram dois sabores... conexão orientada por TCP e datagrama orientada por UDP. O sockets API fornece padrões distintos de chamadas para ambos os estilos.

Para TCP, um servidor é construído por:

1. Criação de um soquete TCP
2. Associando uma porta local ao soquete
3. Configurando o soquete para modo de escuta

4. Aceitar uma nova conexão de um cliente
5. Receba e envie dados
6. Feche a conexão cliente / servidor
7. Voltando para a etapa 4

Para um cliente TCP, construímos por:

1. Criação de um soquete TCP
2. Conectando-se ao servidor TCP
3. Envio de dados / recebimento de dados
4. Feche a conexão

Agora, vamos dividi-los em fragmentos de código que podemos analisar com mais profundidade. O as definições de cabeçalho para a API de sockets podem ser encontradas em <lwip / sockets.h>.

Página 162

## Página 163

Tanto para o cliente quanto para o servidor, a tarefa de criar um soquete é a mesma. Isto é uma chamada de API para a função socket () .

```
sock int = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)
```

O retorno de socket () é um identificador inteiro usado para se referir ao socket.

Os soquetes têm muitos estados associados a eles, no entanto, esse estado é interno ao Implementação de TCP / IP e sockets e não precisa ser exposto à rede programador. Como tal, não há necessidade de expor esses dados ao programador. Podemos pensar em chamar socket () como pedir ao tempo de execução para criar e inicializar todos os dados necessário para uma comunicação de rede. Esses dados são propriedade do tempo de execução e nós recebem um "número de referência" ou identificador que atua como um proxy para os dados. Sempre que desejamos posteriormente realizar o trabalho nessa conexão de rede, passamos de volta identifique o que foi emitido anteriormente para nós e podemos correlacionar de volta à conexão. Isso isola e isola o programador para as entradas da implementação do TCP / IP e nos deixa com uma abstração útil.

Quando estamos criando um soquete do lado do servidor, queremos que ele escute a conexão de entrada solicitações de. Para fazer isso, precisamos dizer ao soquete qual número de porta TCP / IP deve ser ouvindo. Em um determinado dispositivo, apenas um aplicativo por vez pode usar qualquer determinado número de porta. Se quisermos associar um número de porta a um aplicativo, como nosso aplicativo de servidor, neste caso, realizamos uma tarefa chamada "vinculação" que vincula (ou atribui) o número da porta ao soquete que, por sua vez, pertence ao aplicativo.

```
struct sockaddr_in serverAddress;
serverAddress.sin_family = AF_INET;
serverAddress.sin_addr.s_addr = htonl (INADDR_ANY);
serverAddress.sin_port = htons (portNumber);
vincular (sock, (struct sockaddr *) & serverAddress, sizeof (serverAddress));
```

Com o soquete agora associado a um número de porta local, podemos solicitar que o hora de começar a escutar as conexões de entrada. Fazemos isso chamando a API listen (). Antes de chamar listen () , as conexões dos clientes seriam rejeitadas com um indicação ao cliente de que não havia nada no endereço de destino correspondente.

Assim que chamarmos `listen()`, o servidor começará a aceitar conexões de entrada do cliente. O API se parece com:

ouvir (soquete, lista de pendências)

O backlog é o número de solicitações de conexão que o tempo de execução atenderá e aceitar antes de serem entregues ao aplicativo para tratamento. A maneira de pensar sobre isso é imaginar que você é o aplicativo e só pode fazer uma coisa por Tempo. Por exemplo, você só pode falar com uma pessoa por vez ao telefone. Agora imagine que você tem uma secretária que cuida das chamadas recebidas. Quando chega uma chamada

Página 163

## Página 164

e você não estiver ocupado, a secretária desligará a ligação para você. Agora imagine que você é ocupado. Nesse momento, a secretária atende o telefone e pede ao chamador que espere. Quando você se liberta, ela lhe dá a chamada em espera. Agora, vamos supor que você ainda está ocupado quando outro cliente ligar. Ela também diz ao interlocutor para esperar. Estamos começando a construir um fila de chamadores. E é aqui que o conceito de backlog entra em ação. O acúmulo instrui o tempo de execução quantas chamadas podem ser recebidas e solicitadas a aguardar. Se mais ligações chegar do que nosso backlog permitirá, o tempo de execução rejeitará a chamada imediatamente. Não somente isso evita o consumo descontrolado de recursos no servidor, também pode ser usado como uma indicação ao chamador de que pode ser mais bem atendido tentando em outro lugar.

Agora, da perspectiva do servidor, estamos prontos para fazer algum trabalho. Um servidor o aplicativo agora pode bloquear a espera de conexões de clientes de entrada. O pensamento é que um o objetivo do aplicativo de servidor na vida é lidar com as solicitações do cliente e quando ele não tem uma solicitação de cliente ativa, não há nada a fazer a não ser esperar a chegada de uma solicitação. Embora esse seja certamente um modelo, não é necessariamente o único modelo ou mesmo o melhor modelo (em todos os casos). Normalmente gostamos que nossos processadores sejam "utilizados". Meios utilizados que embora tenha um trabalho produtivo que pode fazer, então deve fazê-lo. Se a única coisa nossa programa pode fazer é atender chamadas de clientes, então o modelo original faz sentido. Contudo, existem certos programas que, se eles não tiverem uma solicitação de cliente, imediatamente serviço, pode gastar tempo fazendo outra coisa que seja útil. Voltaremos para essa noção mais tarde. Por enquanto, veremos a chamada de função `accept()`. Quando aceitar () for chamado, uma de duas coisas acontecerá. Se não houver conexão do cliente imediatamente esperando por nós, então vamos bloquear até um momento no futuro quando uma conexão de cliente chega. Nesse momento vamos acordar e ser repassado a conexão para o cliente. Se por outro lado chamamos de `accept()` e já havia uma conexão do cliente esperando para nós, receberemos imediatamente essa conexão e seguiremos em frente. Em ambos os casos, chamamos `accept()` e retornamos uma conexão para um cliente. A distinção entre o casos é se temos ou não de esperar que uma conexão chegue.

A chamada de API se parece com:

```
struct sockaddr_in clientAddress;
socklen_t clientAddressLength = sizeof(clientAddress);
int clientSock = aceitar(sock, (struct sockaddr *) &clientAddress,
& clientAddressLength);
```

O retorno de `accept()` é um novo soquete que representa a conexão entre o cliente solicitante e o servidor. É vital perceber que isso é diferente do servidor socket que criamos anteriormente, que vinculamos à porta de escuta do nosso servidor. Esse soquete é ainda está vivo e bem e existe para continuar a servir outras conexões de cliente. O socket recém-retornado é a conexão para a conversa que foi iniciada por este cliente único. Como todas as conexões TCP, a conversação é simétrica e bidirecional.

**Página 165**

Isso significa que agora não existe mais a noção de cliente e servidor ... ambas as partes podem enviar e receber como gostariam a qualquer momento.

Veja também:

- Wikipedia - [Berkeley Sockets](#)
- [Guia de Beej para programação de rede](#)

**Manipulação de erros**

A maioria das APIs de sockets retorna um código de retorno int. Se este código for <0, então um erro foi ocorreu.

A natureza do erro pode ser encontrada usando o int global chamado " errno ". No entanto, em um ambiente multitarefa, trabalhar com globais não é recomendado. Nas tomadas área, podemos pedir a um soquete o último erro encontrado usando o seguinte código fragmento:

```
int esp8266_get_last_socket_errno(int socket) {
    ret int = 0;
    u32_t optlen = sizeof(ret);
    getsockopt(socket, SOL_SOCKET, SO_ERROR, &ret, &optlen);
    return ret;
}
```

Os significados dos erros podem ser comparados com constantes. Aqui está uma mesa de constantes usadas na implementação atual do FreeRTOS:

Símbolo	Valor	Descrição
EPERM	1	operação não permitida
ENOENT	2	Não existe tal arquivo ou diretório
ESRCH	3	Esse processo não existe
EINTR	4	Chamada de sistema interrompida
EIO	5	Erro de E / S
ENXIO	6	Esse dispositivo ou endereço não existe
E2BIG	7	Lista de Args muito longa
ENOEXEC	8	Erro de formato de execução
EBADF	9	Número de arquivo inválido
ECHILD	10	Nenhum processo filho
EAGAIN	11	Tente novamente
ENOMEM	12	Fora da memória
EACCES	13	Permissão negada
EFAULT	14	Endereço ruim
ENOTBLK	15	Dispositivo de bloqueio necessário
EBUSY	16	Dispositivo ou recurso ocupado

**Página 166**

EEXIST	17	o arquivo existe
EXDEV	18	Link entre dispositivos
ENODEV	19	Esse dispositivo não existe
ENOTDIR	20	Não é um diretório
EISDIR	21	É um diretório
EINVAL	22	Argumento inválido
ENFILE	23	Estouro de mesa de arquivo
EMFILE	24	Muitos arquivos abertos
ENOTTY	25	Não é uma máquina de escrever
ETXTBSY	26	Arquivo de texto ocupado
EFBIG	27	Arquivo muito grande
ENOSPC	28	Sem espaço disponível no dispositivo
ESPIPE	29	Busca ilegal
EROFS	30	Sistema de arquivos somente leitura
EMLINK	31	Muitos links
EPIPE	32	Cano quebrado
EDOM	33	Argumento matemático fora do domínio da função
ERANGE	34	Resultado matemático não representável
EDEADLK	35	Um impasse de recursos ocorreria
ENAMETOOLONG	36	Nome de arquivo muito longo
ENOLCK	37	Sem bloqueios de registro disponíveis
ENOSYS	38	Função não implementada
ENOTEMPTY	39	Diretório não vazio
ELOOP	40	Muitos links simbólicos encontrados
EWOULDBLOCK EAGAIN	41	A operação bloquearia
ENOMSG	42	Nenhuma mensagem do tipo desejado
EIDRM	43	Identificador removido
ECHRNG	44	Número do canal fora do alcance
EL2NSYNC	45	Nível 2 não sincronizado
EL3HLT	46	Nível 3 interrompido
EL3RST	47	Nível 3 redefinido
ELNRNG	48	Número do link fora do intervalo
EUNATCH	49	Driver de protocolo não anexado
ENOCSI	50	Nenhuma estrutura CSI disponível
EL2HLT	51	Nível 2 interrompido
EBADE	52	Troca inválida
EBADR	53	Descriptor de solicitação inválido

Página 166

---

**Página 167**

EXFULL	54	Troca cheia
ENOANO	55	Sem ânodo
EBADRQC	56	Código de requisição inválido
EBADSLT	57	Slot inválido
EBFONT	59	Formato de arquivo de fonte ruim
ENOSTR	60	O dispositivo não é um stream
ENODATA	61	Sem dados disponíveis

ETIME	62	Timer expirou
ENOSR	63	Recursos fora do stream
ENONET	64	A máquina não está na rede
ENOPKG	65	Pacote não instalado
EREMOTE	66	O objeto é remoto
ENOLINK	67	Link foi cortado
EADV	68	Erro de anúncio
ESRMNT	69	Erro de montagem
ECOMM	70	Erro de comunicação ao enviar
EPROTO	71	Erro de protocolo
EMULTIHOP	72	Tentativa de multihop
EDOTDOT	73	Erro específico de RFS
EBADMSG	74	Não é uma mensagem de dados
EOVERFLOW	75	Valor muito grande para o tipo de dados definido
ENOTUNIQ	76	Nome não exclusivo na rede
EBADFD	77	Descriptor de arquivo em mau estado
EREMCHG	78	Endereço remoto alterado
ELIBACC	79	Não é possível acessar uma biblioteca compartilhada necessária
ELIBBAD	80	Acessando uma biblioteca compartilhada corrompida
ELIBSCN	81	Seção .lib em a.out corrompida
ELIBMAX	82	Tentativa de vincular a muitas bibliotecas compartilhadas
ELIBEXEC	83	Não é possível executar uma biblioteca compartilhada diretamente
EILSEQ	84	Sequência ilegal de bytes
ERESTART	85	A chamada do sistema interrompida deve ser reiniciada
ESTRPIPE	86	Erro de tubo de streams
EUSERS	87	Muitos usuários
ENOTSOCK	88	Operação de soquete em não soquete
EDESTADDRREQ	89	Endereço de destino obrigatório
EMSGSIZE	90	Mensagem muito longa
EPROTÓTIPO	91	Protocolo tipo errado para socket

---

**Página 168**

ENOPROTOOPT	92	Protocolo não disponível
EPROTONOSUPPORT	93	Protocolo não suportado
ESOCKTNOSUPPORT	94	Tipo de soquete não compatível
EOPNOTSUPP	95	Operação não suportada no endpoint de transporte
EPFNOSUPPORT	96	Família de protocolo não suportada
EAFNOSUPPORT	97	Família de endereços não suportada pelo protocolo
EADDRINUSE	98	Endereço já em uso
EADDRNOTAVAIL	99	Não é possível identificar o endereço indicado
ENETDOWN	100	Rede está fora do ar
ENETUNREACH	101	Rede está inacessível
ENETRESET	102	A rede caiu na conexão devido à redefinição
ECONNABORTED	103	O software causou o aborto da conexão
ECONNRESET	104	Conexão redefinida por par
ENOBUFS	105	Nenhum espaço de buffer disponível
EISCONN	106	Endpoint de transporte já está conectado

ENOTCONN	107	O endpoint de transporte não está conectado
ESHUTDOWN	108	Não é possível enviar após desligamento do endpoint de transporte
ETOOMANYREFS	109	Muitas referências: não é possível unir
ETIMEDOUT	110	Tempo limite de conexão esgotado
ECONREFUSADO	111	Ligação recusada
EHOSTDOWN	112	O host está fora do ar
EHOSTUNREACH	113	Sem rota para hospedar
EALREADY	114	Operação já em andamento
EINPROGRESS	115	Operação agora em andamento
ESTALE	116	Identificador de arquivo NFS obsoleto
ECLEAN	117	A estrutura precisa de limpeza
ENOTNAM	118	Não é um arquivo de tipo nomeado XENIX
ENAVAIL	119	Sem semáforos XENIX disponíveis
EISNAM	120	É um arquivo de tipo nomeado
EREMOTEIO	121	Erro de E / S remota
EDQUOT	122	Cota excedida
ENOMEDIUM	123	Nenhuma mídia encontrada
TIPO DE EMEDIUM	124	Tipo de meio errado

### Soquetes - aceitar ()

Aceite uma solicitação recebida.

```
int aceitar (int s, struct sockaddr * addr, socklen_t * addrlen)
```

Página 168

---

## Página 169

Aqui, podemos bloquear a espera por uma solicitação de conexão de entrada do soquete do servidor. Voltaremos imediatamente se houver uma conexão de cliente aguardando aceitação. O endereço do cliente nos é devolvido juntamente com o seu comprimento.

Se tentarmos aceitar muitos sockets simultâneos, o ESP8266 pode retornar ENFILE para indicar que temos um estouro.

O retorno é a conexão de soquete aceita para o cliente ou -1 em caso de erro.

### Sockets - bind ()

Associe um soquete a um endereço.

```
vincular (int s, const struct sockaddr * nome, socklen_t namelen)
```

O parâmetro de nome é o endereço do soquete a ser vinculado ao soquete. O namelen fornece o comprimento do endereço. Se sin\_add.s\_addr for htonl (INADDR\_ANY) então nós estamos sendo um servidor ouvindo em qualquer endereço IP de entrada.

Um retorno de <0 em caso de erro.

Aqui está um exemplo de como nos definimos como um servidor:

```
struct sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_addr.s_addr = htonl (INADDR_ANY);
serverAddr.sin_port = htons (80);
```

```
int rc = bind (serverSocket, (struct sockaddr *) & serverAddr, sizeof (serverAddr));
```

### Soquetes - fechar ()

Feche o soquete correspondente.

`fechar (int s)`

Feche o soquete.

#### Sockets - closesocket ()

Feche o soquete correspondente.

`closesocket (int s)`

Feche o soquete.

#### Soquetes - conectar ()

Conecte-se a um servidor.

`conectar (int s, const struct sockaddr * partnerAddr, socklen_t addrlen)`

Página 169

## Página 170

Conekte o soquete a um parceiro. O endereço do parceiro é fornecido no campo `partnerAddr`. Esta é uma chamada iniciada pelo cliente e espera-se que se conecte a um servidor de escuta.

#### Sockets - fcntl ()

Execute funções de controle.

`fcntl (int s, int cmd, int val)`

Podemos definir funções de controle em soquetes aqui.

Comando	Valor	Descrição
<code>F_SETFL</code>	<code>O_NONBLOCK</code>	Defina o soquete como não bloqueado.

#### Sockets - freeaddrinfo ()

#### Sockets - getaddrinfo ()

#### Sockets - gethostbyname ()

#### Sockets - getpeername ()

Recupere o endereço associado ao parceiro / par ao qual o soquete está conectado.

```
getpeername (int s,
            struct sockaddr * peerAddr,
            socklen_t * namelen)
```

Observe que `namelen` deve ser preparado com o tamanho do buffer de endereço disponível.

#### Sockets - getsockname ()

Recupere o endereço local atual ao qual o soquete está vinculado.

```
getsockname (int s,
            nome do struct sockaddr *,
            socklen_t * namelen)
```

Observe que `namelen` deve ser preparado com o tamanho do buffer de endereço disponível.

**Sockets - getsockopt ()**

```
getsockopt (int s,
           nível interno,
           int optname,
```

Página 170

**Página 171**

```
void * optval,
socklen_t * optlen)
```

Um exemplo importante de uso desta função é recuperar o último erro associado com o soquete. O seguinte fragmento de código ilustra isso:

```
int espx_last_socket_errno (int socket) {
    ret int = 0;
    u32_t optlen = sizeof (ret);
    getsockopt (socket, SOL_SOCKET, SO_ERROR, & ret, & optlen);
    return ret;
}
```

**Sockets - htonl ()**

Converte um inteiro longo formatado de host em ordem de bytes de rede.

**Soquetes - htons ()**

Converte um inteiro curto formatado de host em ordem de bytes de rede.

**Sockets - inet\_ntop ()****Soquetes - inet\_pton ()****Sockets - ioctlsocket ()**

```
ioctl (int s, long cmd, void * argp)
```

**Sockets - ouvir ()**

Comece a ouvir as conexões de entrada.

```
ouvir (int s, int backlog)
```

Se estivermos vinculados a um servidor, começaremos a escutar as solicitações de conexão de entrada em o soquete. O parâmetro backlog define quantos sockets podemos controlar antes de aceitá-los.

Um valor de retorno <0 significa um erro.

**Sockets - ler ()**

Receba dados de um parceiro.

```
ssize_t lido (int s, void * mem, size_t len)
```

---

**Página 172**

Semelhante à função recv () .

Veja também:

- [Sockets - recv\(\)](#)
- [Sockets - recvfrom\(\)](#)

### Sockets - recv ()

Receba dados de um parceiro.

```
ssize_t recv (int s,
             void * mem,
             size_t len,
             sinalizadores internos)
```

Esta função retorna o número de bytes realmente recebidos. Um valor de -1 indica um erro. Um valor zero indica que o parceiro fechou a conexão.

Os sinalizadores são a combinação booleana de:

- MSG\_CMSG\_CLOEXEC
- MSG\_DONTWAIT – Indica que não queremos bloquear a espera de dados. Se houver nenhum dado imediatamente disponível para recebermos, retornamos -1 para indicar um erro e o código de erro é "EAGAIN".
- MSG\_ERRQUEUE
- MSG\_OOB
- MSG\_PEEK
- MSG\_TRUNC
- MSG\_WAITALL

Veja também:

- [Sockets - ler\(\)](#)
- [Sockets - recvfrom\(\)](#)
- [man\(2\)-recv](#)

### Sockets - recvfrom ()

```
recvfrom (int s,
          void * mem,
          size_t len,
          sinalizadores int,
          struct sockaddr * from,
          socklen_t * fromlen)
```

Veja também:

- [Sockets - ler\(\)](#)

**Página 173**

- [Sockets - recv\(\)](#)

### Soquetes - selecione ()

Verifique se há dados disponíveis para leitura ou gravação.

```
select (int maxfdp1,
        fd_set * readset,
        fd_set * Writeteet,
        fd_set * exceptset,
        struct timeval * timeout)
```

Em alguns sistemas Linux, para usar selecionado, incluiria <sys / select.h> . Em ESP-IDF que não está presente, mas não parece ser necessário.

### Sockets - enviar ()

Envie um conjunto de bytes pelo soquete para o parceiro.

```
ssize_t send (int s, const void * dataptr, size_t size, int flags)
```

Os dados apontados por dataptr para bytes de tamanho são transmitidos.

Veja também:

- [homem \(2\) - enviar](#)

### Sockets - sendto ()

Envie dados para um parceiro UDP.

```
sendto (int s,
        const void * dataptr,
        size_t size,
        sinalizadores int,
        const struct sockaddr * to,
        socklen_t tolen)
```

### Sockets - setsockopt ()

```
setsockopt (int s,
            nível interno,
            int optname,
            const void * optval,
            socklen_t optlen)
```

As opções que devem estar disponíveis são:

- TCP\_NODELAY - Desativa o algoritmo de Nagle.
- SO\_KEEPALIVE - Habilita ping de vivacidade .

### Sockets - shutdown ()

Desligue partes de um soquete.

```
desligamento (int s, int como)
```

Desligue todo ou parte do soquete.

O soquete é desligado com base no parâmetro como, que pode ser um dos seguintes:

- SHUT\_RD - Nenhum outro recebimento é permitido.
- SHUT\_WR - Nenhuma outra gravação é permitida.
- SHUT\_RDWR - Nenhuma leitura ou gravação adicional é permitida.

**Soquetes - soquete ()**

Crie um novo soquete para o domínio, tipo e protocolo específicos.

socket (domínio interno, tipo interno, protocolo interno)

O domínio pode ser um dos seguintes:

- AF\_INET - TCP / IP
- Outras ...

O tipo pode ser um dos seguintes:

- SOCK\_STREAM
- SOCK\_DGRAM
- SOCK\_RAW

O protocolo pode ser um dos seguintes:

- IPPROTO\_IP
- IPPROTO\_TCP
- IPPROTO\_UDP

Um padrão de uso comum é:

sock int = socket (AF\_INET, SOCK\_STREAM, IPPROTO\_TCP)

Retorna um novo descritor de soquete ou um valor <0 em caso de erro.

**Sockets - escrever ()**

escrever (int s, const void \* dataptr, size\_t size)

Página 174

**Página 175****Estruturas de dados de soquete**

Sockets - struct sockaddr

Sockets - struct sockaddr\_in

- sin\_family - AF\_INET
- sin\_port
- struct in\_addr sin\_addr - Esta estrutura tem um membro chamado s\_addr que é um Endereço de IP. Valores especiais têm significados especiais. Por exemplo INADDR\_ANY é qualquer endereço.

**Java Sockets**

A API de sockets é a API padrão de fato para programação em TCP / IP. Minhas linguagem de programação de escolha é Java e tem suporte total para sockets. O que isso significa que posso escrever um aplicativo baseado em Java que utiliza sockets para comunicar com o ESP \*. Posso enviar e receber dados facilmente.

Em Java, existem duas classes principais que representam sockets, são java.net.Socket que representa um aplicativo cliente que formará uma conexão e

a segunda classe é `java.net.ServerSocket` que representa um servidor que está ouvindo em um soquete aguardando uma conexão do cliente. Já que o ESP \* pode ser um cliente ou um servidor, ambas as classes Java entrarão em jogo.

Para conectar a um ESP \* rodando como um servidor, precisamos saber o endereço IP do dispositivo e o número da porta em que está escutando. Depois de conhecê-los, podemos criar uma instância do cliente Java com:

```
Socket clientSocket = novo Socket (ipAddress, porta);
```

Isso formará uma conexão com o ESP \*. Agora podemos pedir um `InputStream` de para receber dados do parceiro e um `OutputStream` para o qual podemos gravar dados.

```
InputStream is = clientSocket.getInputStream ();
OutputStream os = clientSocket.getOutputStream ();
```

Quando terminarmos com a conexão, devemos chamar `close ()` para fechar o lado Java da conexão:

```
clientSocket.close ();
```

Realmente é tão simples quanto isso. Aqui está um exemplo de aplicativo:

```
pacote kolban;
```

```
import java.io.OutputStream;
import java.net.Socket;
```

Página 175

## Página 176

```
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.DefaultParser;
import org.apache.commons.cli.Options;

public class SocketClient {
    private String hostname;
    porta privada interna;

    public static void main (String [] args) {
        Opções opções = novas opções ();
        options.addOption ("h", true, "hostname");
        options.addOption ("p", true, "port");
        Analisador CommandLineParser = novo DefaultParser ();
        tentar {
            CommandLine cmd = parser.parse (opções, argumentos);

            Cliente SocketClient = novo SocketClient ();
            client.hostname = cmd.getOptionValue ("h");
            client.port = Integer.parseInt (cmd.getOptionValue ("p"));
            client.run ();
        } catch (Exceção e) {
            e.printStackTrace ();
        }
    }

    public void run () {
        tentar {
            int SIZE = 65000;
            dados de byte [] = novo byte [SIZE];
            para (int i = 0; i < SIZE; i ++) {
                dados [i] = 'X';
            }
            Soquete s1 = novo soquete (nome do host, porta);
            OutputStream os = s1.getOutputStream ();
            os.write (dados);
            s1.close ();
            System.out.println ("Dados enviados!");
        } catch (Exceção e) {
            e.printStackTrace ();
        }
    }
}
```

```

        }
    }
} // Fim da aula
// Fim do arquivo

```

Configurar um aplicativo Java como um servidor de soquete é tão fácil quanto. Desta vez nós criamos uma instância da classe `SocketServer` usando:

```
SocketServer serverSocket = novo SocketServer (porta)
```

A porta fornecida é o número da porta na máquina na qual o JVM está executando esse será o ponto final das solicitações de conexão do cliente remoto. Assim que tivermos um

Página 176

## Página 177

Instância `ServerSocket`, precisamos esperar por uma conexão de cliente de entrada. Nós fazemos isso usando o método de API de bloqueio chamado `accept()`.

```
Socket partnerSocket = serverSocket.accept ();
```

Esta chamada é bloqueada até que uma conexão do cliente chegue. O `partnerSocket` retornado é o soquete conectado ao parceiro, que pode ser usado da mesma maneira que antes discutido para conexões de cliente. Isso significa que podemos solicitar o `InputStream` e Objetos `OutputStream` para ler e gravar de e para o parceiro. Uma vez que Java é um multi linguagem encadeada, uma vez que acordamos de `accept()` podemos passar o recebido socket do parceiro para um novo thread e repita a chamada `accept()` para outro paralelo conexões. Lembre-se de fechar () todas as conexões de soquete de parceiro que você receber quando você terminou com eles.

Até agora, falamos sobre conexões orientadas a TCP, onde uma vez que uma conexão é aberto, ele permanece aberto até que seja fechado, período durante o qual qualquer extremidade pode enviar ou receber independentemente do outro. Agora vamos olhar para os datagramas que usam o protocolo UDP.

A classe principal por trás disso é chamada `DatagramSocket`. Ao contrário do TCP, o `DatagramSocket` classe é usada tanto para clientes quanto para servidores.

Primeiro, vejamos um cliente. Se quisermos escrever um cliente UDP Java, criaremos um instância de um `DatagramSocket` usando:

```
DatagramSocket clientSocket = new DatagramSocket ();
```

Em seguida, iremos "conectar" ao parceiro UDP remoto. Precisamos saber o endereço IP e a porta que o parceiro está ouvindo. Embora a API seja chamada de "conectar", nós precisa perceber que nenhuma conexão é formada. Os datagramas não têm conexão, então o que que estamos realmente fazendo é associar nosso soquete de cliente com o soquete de parceiro no outra extremidade para que, **quando** realmente desejamos enviar dados, saibamos para onde enviá-los.

```
clientSocket.connect (ipAddress, port);
```

Agora estamos prontos para enviar um datagrama usando o método `send()`:

```
Dados DatagramPacket = novo DatagramPacket (novo byte [100], 100);
clientSocket.send (dados);
```

Para escrever um ouvinte UDP que ouve datagramas de entrada, podemos usar o seguinte:

```
DatagramSocket serverSocket = novo DatagramSocket (porta);
```

A porta aqui é o número da porta na mesma máquina que a JVM que será usada para escute as conexões UDP de entrada.

Para aguardar a chegada de um datagrama, ligue para `receive()`.

```
Dados DatagramPacket = novo DatagramPacket (novo byte [100], 100);
clientSocket.receive (dados);
```

---

**Página 178**

Se você for usar as APIs Java Socket, leia o JavaDoc completamente para estes classes existem muitos recursos e opções que não foram listados aqui.

Veja também:

- [Tutorial de Java: tudo sobre soquetes](#)
- [JDK 8 JavaDoc](#)

### **WebSockets**

WebSockets é uma API e um protocolo introduzido em HTML5. Simplificando, se nós imagine um servidor HTTP esperando por solicitações HTTP de entrada, podemos converter um solicitação atual em uma conexão de soquete entre o servidor e o navegador de modo que qualquer uma das extremidades pode enviar dados a serem recebidos por seu parceiro.

Aqui, vemos uma solicitação bruta para atualizar uma conexão HTTP para um WebSocket conexão:

```
GET / HTTP / 1.1
Host: 192.168.1.10
Conexão: Upgrade
Pragma: sem cache
Cache-Control: sem cache
Upgrade: websocket
Origem: Arquivo://
Versão Sec-WebSocket: 13
Agente do usuário: Mozilla / 5.0 (X11; Linux x86_64) AppleWebKit / 537.36 (KHTML, como Gecko)
Chrome / 46.0.2490.86 Safari / 537.36
Aceitar-Codificação: gzip, deflate, sdch
Idioma de aceitação: en-US, en; q = 0.8
Sec-WebSocket-Key: saim6TzFH + zVb4qY2nrh0Q ==
Extensões Sec-WebSocket: permessage-deflate; client_max_window_bits
```

Veja também:

- [eb](#)
- 
- 

### **Um aplicativo de navegador WebSocket**

A maior probabilidade é que você esteja executando o ESP8266 como um servidor WebSocket.

Isso implicaria que você terá aplicativos hospedados no navegador que serão conectando-se a um servidor WebSocket como clientes e, a partir daí, você provavelmente estará escrevendo algum código de cliente WebSocket ... senão mais, para testar a unidade de seu servidor. Por causa disso, vamos agora passar algum tempo falando sobre o que está envolvido em escrever um Aplicativo cliente WebSocket.

Vamos supor que escreveremos JavaScript hospedado no navegador. Nós começamos por criar uma instância de um objeto WebSocket passando a URL para o WebSocket servidor:

```
var ws = new WebSocket ("ws://<somehost>[:<someport>]");
```

A API WebSocket é principalmente orientada a eventos e existem vários tipos de eventos de interesse para nós:

- open - Invocado quando a conexão com o servidor WebSocket foi estabelecido e agora estamos prontos para enviar ou receber dados. Podemos definir isso com a propriedade "onopen" do WebSocket como uma referência de função.
- mensagem - Receba uma mensagem do servidor. Podemos definir isso com o propriedade "onmessage" do WebSocket como uma referência de função.
- erro - Receba uma indicação de que um erro foi detectado. Podemos definir isso com a propriedade "onerror" do WebSocket como uma referência de função.
- fechar - Recebe uma indicação de que um pedido para fechar a conexão foi detectou. Podemos definir isso com a propriedade "onclose" do WebSocket como um referência de função.

Os manipuladores de eventos podem ser registrados com um mecanismo "on<Event>" ou com um chamada addEventListener().

Existem dois métodos definidos em um objeto WebSocket. Esses são:

- enviar - enviar dados para um servidor WebSocket
- fechar - Fechar a conexão com um servidor WebSocket. O método close() leva dois parâmetros:
  - código de fechamento - um código de fechamento inteiro que descreve o motivo do fechamento.
    - 1000 - CLOSE\_NORMAL
    - 1001 - CLOSE\_GOING\_AWAY
  - mensagem de status - Uma string que descreve o motivo do fechamento.

Finalmente, existem alguns atributos:

- readyState - O estado da conexão WebSocket. Os valores incluem:
  - WebSocket.CONNECTING
  - WebSocket.OPEN
  - WebSocket.CLOSING
  - WebSocket.CLOSED

- bufferedAmount - Quantidade de dados em buffer com transmissão pendente para o Servidor WebSocket
- protocolo - O protocolo selecionado do servidor WebSocket em uso.

### FreeRTOS WebSocket

O FreeRTOS SDK distribui uma implementação do projeto de código aberto noPoll.

Para usar o noPoll em seu aplicativo, você deve incluir " nopoll.h ".

O contexto é muito importante e deve ser criado no início e eliminado no fim. Por exemplo:

```
noPollCtx * ctx = nopoll_ctx_new ();
// Faça alguma coisa ...
nopoll_ctx_unref ();
```

Depois de criar um contexto, podemos nos registrar como um servidor:

```
noPollCon * listener = nopoll_listener_new (ctx, "0.0.0.0", "1234");
nopoll_ctx_set_on_msg (ctx, listener_on_message_handler, NULL);
nopoll_loop_wait (ctx, 0);
```

Quando uma mensagem é recebida, a função registrada (listener\_on\_message\_handler) será invocado:

```
void listener_on_message_handler (
    noPollCtx * ctx,
    noPollConn * conn,
    noPollMsg * msg,
    noPollPtr * userData) {
    // Faça alguma coisa
    nopoll_conn_send_text (conn, "Obrigado", 5);
}
```

Um exemplo de aplicativo WebSocket é fornecido pela Espressif no SDK do FreeRTOS para ESP8266. O exemplo pode ser encontrado em / examples / websocket\_demo.

Veja também:

- 
- 
- 

### Mongoose WebSocket

Usando as bibliotecas Mongoose de Cesanta, podemos configurar um servidor WebSocket. Depois de definir criar uma ligação para solicitações de rede de entrada que podemos chamar

`mg_set_protocol_http_websocket ()`. Isso irá anexar um manipulador de eventos à rede nível de protocolo para lidar com eventos associados a WebSockets. Especificamente, estes são os seguintes eventos nos quais estamos interessados:

Página 180

---

## Página 181

- MG\_EV\_WEBSOCKET\_HANDSHAKE\_REQUEST
- MG\_EV\_WEBSOCKET\_HANDSHAKE\_DONE
- MG\_EV\_WEBSOCKET\_FRAME

Quando um MG\_EV\_WEBSOCKET\_HANDSHAKE\_REQUEST é recebido, os dados contêm um Solicitação de HTTP como uma estrutura http\_message .

A estrutura http\_message contém:

- mensagem
- método
- uri
- proto
- resp\_code
- resp\_status\_msg

- query\_string
- header\_names
- header\_values
- corpo

Quando um MG\_EV\_WEBSOCKET\_FRAME é recebido, os dados contêm uma referência a uma estrutura websocket\_message . A estrutura websocket\_message contém:

- dados - os dados passados do parceiro.
- size - O tamanho dos dados passados.
- flags - Sinalizadores (desconhecido).

## Servidores web

Um servidor da Web é um componente de software que escuta as solicitações de HTTP recebidas de Navegadores da web. Existem muitas implementações de servidores da Web que podem ser executados em um Ambiente ESP8266.

### Mangusto

O Mongoose fornece uma API que pode ser usada para construir um servidor HTTP rico e poderoso.

Em um nível alto, chamamos mg\_mgr\_init () para inicializar o ambiente. Em seguida, ligamos a um manipulador usando mg\_bind () . Finalmente, verificamos o funcionamento do servidor usando mg\_mgr\_poll () .

Página 181

---

## Página 182

```
void setupMongoose () {
    struct mg_mgr mgr;
    printf ("Iniciando a configuração do mangusto \n");
    mg_mgr_init (& mgr, NULL);
    printf ("Iniciado com sucesso \n");
    mg_bind (& mgr, "80", evHandler);
    printf ("Vinculado com sucesso \n");
    enquanto (1) {
        mg_mgr_poll (& mgr, 1000);
    }
}
```

Quando uma solicitação chega de um navegador, consideraremos que um evento que é entregue para um manipulador de eventos. A assinatura de um manipulador de eventos é:

```
void eventHandler (
    struct mg_connection * nc,
    int ev,
    void * evData)
```

Onde:

- nc - A conexão que recebeu o evento.
- ev - O tipo de evento que acionou o retorno de chamada.
- evData - Dados associados ao evento.

Até agora, receberemos retornos de chamada para conexões de soquete. Se quisermos, podemos agora registrar que desejamos analisar os dados recebidos como uma solicitação HTTP. Nós fazemos isso por fazendo uma chamada para mg\_set\_protocol\_http\_websocket () .

Quando configurado como um servidor HTTP, uma solicitação de entrada do navegador aparecerá com o evento tipo de MG\_EV\_HTTP\_REQUEST. O evData passado será uma instância de struct

http\_message a partir do qual a natureza da solicitação pode ser determinada.

Veja também:

- Github: [cesanta / mangusto](https://github.com/cesanta/mangusto)
- [Mongoose Developer Center](#)

## Programação usando Eclipse

Eclipse é uma estrutura de software livre popular usada principalmente para hospedar aplicativos ferramentas de desenvolvimento. Embora seja principalmente voltado para a construção de aplicativos Java, ele também tem suporte de primeira classe C e C++.

Um projeto para construir aplicativos ESP8266 usando Eclipse pode ser encontrado aqui:

Página 182

---

## Página 183

- <http://www.esp8266.com/viewtopic.php?f=9&t=820>

Não inclua espaços em nenhuma das partes do caminho que apontem para a área de trabalho. Aqui estão algumas notas sobre a instalação deste projeto ... no entanto, sempre leia a documentação acompanhamento do projeto.

Baixe o Espressif-ESP8266-DevKit-vxxx-x86 . Este é um grande download de aprox.  
125 MBytes.

Execute o instalador. Ele irá pedir a sua escolha de idioma de instalação.

Em seguida, vem a tela inicial:

Em seguida, vem o contrato de licença:

Página 183

---

**Página 184**

Agora, a seleção de quais componentes instalar:

---

**Página 185**

Finalmente, uma caixa de diálogo de confirmação para revisar o que você selecionou.

O resultado disso será uma nova estrutura de diretório em C:\Espressif\ .

Existem outras dependências de que você precisa, as quais estão listadas no link acima.

Esses incluem:

- Um ambiente de execução Java. Eu uso o Java 8 mais recente da Oracle.
- Ambiente Eclipse com ferramentas de desenvolvimento C / C ++. Eu uso a última versão de "Marte".
- MinGW - ferramentas e utilitários Unix que são executados no Windows.
- Auxiliar de instalação MinGW - Um cache e uma lista dos pacotes MinGW que precisam a ser instalado para operação correta.

Os Makefiles fornecidos com o pacote são essenciais. Eles foram criados para fornecer as compilações mais fáceis. Os alvos contidos nos Makefiles incluem:

- all - Compila todo o código, mas não pisca.
- limpar - Limpa todas as compilações anteriores.
- flash - Compile o código se necessário e depois faça flash.
- flashboot
- flashinit

Página 185

---

**Página 186**

- flashonefile

Existem alguns sinalizadores que são usados com o Makefile que você pode editar. Esses incluem:

[https://translate.googleusercontent.com/translate\\_f](https://translate.googleusercontent.com/translate_f)

157/370

- **VERBOSE = 1** - Habilita o detalhamento que inclui informações de depuração. Especificamente os comandos de compilação são mostrados.

Veja também:

- [Eclipse.org](#)
- [Eclipse C / C ++ Development Tooling \(CDT\)](#)
- [Tópico principal do fórum](#)

## Instalando o Terminal Serial Eclipse

Embora existam muitos terminais seriais excelentes disponíveis como Windows autônomo aplicativos, uma alternativa é o Terminal Eclipse que também possui suporte serial. Esta permite que um terminal serial apareça como uma visualização dentro do IDE Eclipse. Não vem instalado por padrão, mas as etapas para adicionar não são complexas.

Primeiro inicie o Eclipse (eu uso a versão de Marte).

Vá para Ajuda> Instalar novo software .

Selecione o repositório de download do eclipse.

Selecione Mobile and Device development> TM Terminal .

Passe pelas seções a seguir e, quando solicitado a reiniciar, aceite sim.

Ainda não estamos prontos para usá-lo, devemos adicionar suporte para porta serial no Eclipse.

Volte para Ajuda> Instalar novo software e adicione um novo repositório

Página 187

---

## Página 188

O URL do repositório é:

- <http://archive.eclipse.org/tm/update/rxtx/>

Agora podemos selecionar a biblioteca de suporte em tempo de execução da porta serial:

Página 188

---

**Página 189**

Siga as demais telas de navegação e reinicie o Eclipse quando solicitado.

Agora temos o suporte para terminal instalado e estamos prontos para usá-lo. Do Windows> Mostrar Exibir> Outros , encontraremos uma nova categoria chamada " Terminal ".

Página 189

---

## Página 190

Abrir isso adiciona uma visualização Terminal à nossa perspectiva. Existe um botão que permite para abrir uma nova instância de terminal que é mostrada na imagem a seguir:

Clicar aqui abre a caixa de diálogo que nos pergunta o tipo de terminal e as propriedades. Para nossos propósitos, desejamos escolher um terminal serial. Não se esqueça de definir também a porta e taxa de transmissão para corresponder ao que seu ESP8266 usa.

Página 190

---

**Página 191**

Após clicar em OK, após alguns segundos veremos que estamos conectados e um novo ícone de desconexão aparece:

E agora o terminal está ativo. Para meus propósitos, eu conecto este terminal ao UART1 de o ESP8266 para depuração, deixando o UART0 para atualizar novas cópias do meu aplicativo. Aqui está um exemplo da aparência de minha janela típica:

Página 191

---

**Página 192**

[https://translate.googleusercontent.com/translate\\_f](https://translate.googleusercontent.com/translate_f)

162/370

Você pode inverter as cores para produzir uma visualização de branco sobre preto que muitos usuários preferir.

### **Desenvolvimento web usando Eclipse**

O Eclipse também fornece um ambiente de desenvolvimento da web de primeira classe para escrever e testar aplicativos da web, incluindo páginas HTML. É sugerido que o Eclipse Web Developer Tools ser instalado.

Página 192

---

**Página 193**

### **Programação usando o Arduino IDE**

Muito antes de haver um ESP8266, havia o Arduino. De vital importância contribuição para a comunidade de hardware de código aberto e o ponto de entrada para a maioria de amadores no mundo dos circuitos e processadores construídos em casa.

Uma das principais atrações do Arduino é sua relativa baixa complexidade, permitindo

todos a capacidade de construir algo de forma rápida e fácil. O Desenvolvimento Integrado Ambiente (IDE) para o Arduino sempre foi gratuito para download em a Internet. Se um programador profissional se sentasse com ele, eles seriam chocados com suas aparentes capacidades limitadas. No entanto, o subconjunto de funções que ele fornece em comparação com um IDE "completo", pode cobrir 90% do que se deseja alcançar. Combine isso com a interface intuitiva e o IDE do Arduino é uma força a ser reconhecida com.

Aqui está a aparência de um programa simples no IDE do Arduino:

No jargão do Arduino, um aplicativo é denominado "esboço". Pessoalmente, não sou fã disso frase, mas tenho certeza de que uma pesquisa foi feita para descobrir que este é o nome menos intimidante

Página 193

---

## Página 194

para o que de outra forma seria chamado de programa de linguagem C e que intimidaria o menor número de pessoas.

O IDE possui um botão denominado " Verificar " que, ao ser clicado, compila o programa. De claro, isso também terá o efeito colateral de verificar se o programa compila de forma limpa ... mas compilação é o que ela faz. Um segundo botão é chamado de " Upload " que, quando clicado, o que ele faz é implantar o aplicativo no Arduino.

Além de fornecer um editor de linguagem C mais ferramentas para compilar e implantar, o Arduino IDE fornece bibliotecas pré-fornecidas de rotinas C que "escondem" complexas detalhes de implementação que, de outra forma, podem ser necessários ao programar para o Placas Arduino. Por exemplo, a programação UART, sem dúvida, teria que definir registros, lidar com interrupções e muito mais. Em vez de fazer com que os usuários pobres tenham que aprender essas APIs técnicas. o pessoal do Arduino forneceu bibliotecas de alto nível que poderiam ser chamadas dos esboços com interfaces mais limpas que escondem o "gorp" mecânico que acontece sob as cobertas. Essa noção é a chave ... como essas bibliotecas, tanto quanto qualquer coisa

caso contrário, forneça o ambiente para os programadores do Arduino.

Por mais interessante que seja esta história, você pode estar se perguntando como isso se relaciona com o nosso ESP8266 história? Bem, um monte de indivíduos talentosos construíram um projeto de código aberto em Github que fornece um "plug-in" ou "extensão" para a ferramenta Arduino IDE (lembre-se de que o IDE do Arduino é gratuito). O que esta extensão faz é permitir que alguém escreva esboços em o IDE do Arduino que aproveita as interfaces da biblioteca do Arduino que, ao compilar e tempo de implantação, gere o código que será executado no ESP8266. O que isso efetivamente significa que podemos usar o IDE do Arduino e construir aplicativos ESP8266 com o mínimo de barulho.

### **Implicações do suporte do Arduino IDE**

A capacidade de tratar um ESP8266 como se fosse "como" um Arduino é uma noção de que eu ainda não foi capaz de absorver totalmente. ESP8266 é uma CPU Tensilica ao contrário do Arduino que é uma CPU ATmega. A Espressif criou uma API dedicada e arquitetada no forma de seu SDK para APIs ESP8266 diretamente expostas. As bibliotecas do Arduino para ESP8266 parece mapear sua intenção para essas APIs expostas. Por essas razões e semelhante, pode-se argumentar que o suporte do Arduino é uma fachada desnecessária em cima de um ambiente perfeitamente bom e impondo um modelo de tecnologia "alienígena" no topo do ESP8266 funções nativas, estamos mascarando o acesso a níveis mais baixos de conhecimento e função. Além disso, pensar no ESP8266 como se fosse um Arduino pode levar a problemas de design. Por exemplo, o ESP8266 precisa de controle regular para lidar com WiFi e outras ações internas. Isso entra em conflito com o modelo Arduino, onde o programador pode fazer o que quiser dentro da função de loop pelo tempo que quiser.

Página 194

---

### **Página 195**

O outro lado é que a curva de aprendizado para fazer algo rodar em um Arduino foi mostrado ser extremamente baixo. Não demora muito para acender uma luz piscando em um placa de ensaio. Com essa linha de pensamento, por que os usuários do ESP8266 devem ser penalizados por ter que instalar e aprender cadeias de ferramentas mais complexas e sintaxe para conseguir o mesmo resultado com mais ferramentas e técnicas orientadas para ESP8266? O nome do jogo deve ser para permitir que as pessoas mexam em CPUs e sensores sem ter que ter diplomas universitários em ciência da computação ou engenharia elétrica e se o preço paga para chegar lá é inserir uma ilusão "simples de usar", então por que não? Se eu construir um jornal avião e jogá-lo pela janela ... Posso ter prazer com isso. Um foguete da NASA o cientista não deve zombar das minhas atividades ou da falta de conhecimento de aerodinâmica ... o papel dobrado fez seu trabalho e eu alcancei meu objetivo. No entanto, se meu trabalho fosse colocar um homem a lua, a capacidade de visualizar as realidades da tecnologia no nível "realista" torna-se extremamente importante.

### **Instalando o Arduino IDE com suporte ESP8266**

Para montar este ambiente, deve-se baixar uma versão atual do IDE Arduino.

Isso será cerca de 140 Mbytes.

Baixo a versão do arquivo ZIP e extraia seu conteúdo.

Em seguida, lançamos o Arduino IDE e abrimos a caixa de diálogo Preferências :

Página 195

---

**Página 196**

No campo URLs adicionais do gerenciador de placas, insira o URL do pacote ESP8266  
qual é:

[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)

Página 196

---

**Página 197**

Selecione o gerenciador de placas no menu Ferramentas> Placa :

---

**Página 198**

Instale o suporte ESP8266:

Página 198

---

**Página 199**

Isso entrará em contato com a Internet e fará o download dos artefatos necessários para ESP8266 Apoio, suporte.

Depois de concluído, nas seleções da placa IDE do Arduino, você encontrará o campo " Genérico Módulo ESP8266 ":

Página 199

---

**Página 200**

Agora estamos prontos para começar a construir, compilar e executar esboços.

Um esboço simples e de amostra que recomendo para teste é:

```
void setup () {
    Serial1.begin (115200);
}

void loop () {
    Serial1.println ("Olá! - millis () =" + String (millis ()));
}
```

Quando executado, um loop de mensagens aparecerá na saída do UART1 dizendo olá e o número de milissegundos desde a última inicialização. Tanto quanto qualquer coisa, isso irá validar que o

Página 200

## Página 201

ambiente foi configurado corretamente, você pode compilar um programa e essa implantação para o ESP8266 é bem-sucedido.

Veja também:

- Github: [esp8266 / Arduino](https://github.com/esp8266/Arduino)
- [IDE Arduino](https://www.arduino.cc/en/Main/Software)

### Dicas para trabalhar no ambiente Arduino

Lembre-se de que o ambiente do Arduino é composto por duas coisas. Primeiro, um aplicativo real que você instala em sua máquina fornecendo o IDE do Arduino. Em segundo lugar, um conjunto de bibliotecas que modelar aqueles disponíveis para um dispositivo Arduino real que são mapeados para ESP \*. Capacidades. Com isso em mente, aqui estão algumas dicas e sugestões que considero úteis quando escrever esboços do Arduino para um ambiente ESP \*.

### Inicializar classes globais na configuração ()

Dentro de um esboço do Arduino, temos uma função pré-fornecida chamada `setup()` que é chamada apenas uma vez durante a inicialização do ESP8266. Dentro desta função, você executa uma vez funções de inicialização. Em C++, temos a capacidade de criar instâncias de classe globalmente.

Por exemplo:

```
MyClass myClass (123);
```

```
void setup () {
    // Algun código aqui
}
```

em vez disso, use o seguinte:

```
MyClass * myClass;
```

```
void setup () {
    minhaClasse = nova MinhaClasse (123);
    // Algun código aqui
}
```

É claro que isso muda o tipo de dados da sua variável. Deixou de ser uma instância de MyClass para ser um ponteiro para uma instância de MyClass, o que significa que você pode ter para mudar outros aspectos do seu programa ... mas a razão para isso é que no primeiro caso, o construtor de sua instância MyClass foi executado fora do setup () e não podemos dizer em que estado o ambiente poderia estar naquele ponto. Dentro da configuração () código, temos uma expectativa razoável do contexto do ambiente.

### Invocando Espressif SDK API a partir de um esboço

Página 201

---

## Página 202

Não há nada que o impeça de invocar a API Espressif SDK de dentro do seu retrato falado. Você deve incluir todos os arquivos de inclusão necessários. Aqui está um exemplo de incluindo " user\_interface.h ".

```
extern "C" {
    #include "user_interface.h"
}
```

Observe o colchete com a construção C ++ que faz com que o conteúdo apareça como embora estivesse sendo definido em um programa C.

### Manipulação de exceção

Quando uma exceção é detectada no código, o código é interrompido. Normalmente vemos o seguinte conectado à porta serial quando isso acontecer:

```
data de 8 de janeiro de 2013, primeira causa: 2, modo de inicialização: (1,7)
data de 8 de janeiro de 2013, primeira causa: 4, modo de inicialização: (1,7)
wdt reset
```

Infelizmente, isso não nos diz absolutamente nada sobre a localização ou a causa do problema.

## O sistema de arquivos SPIFFS

### O comando mkspiffs

Foi disponibilizada uma ferramenta que cria um binário do sistema de arquivos " spiffs " a partir de um diretório estrutura encontrada no disco. O comando é chamado de " mkspiffs " e tem seu próprio github projeto.

A sintaxe completa do comando é:

```
mkspiffs {-c <pack_dir> |-l | -i} [-b <number>] [-p <number>] [-s <number>] [-]
[-version] [-h] <image_file>
```

Onde os parâmetros são:

- -c <pack\_dir> ou --create <pack\_dir> - Cria um arquivo de imagem spiffs a partir de exame de um diretório a ser compactado na imagem do spiffs.
- -l ou --list - Lista o conteúdo de um arquivo de imagem existente.
- -i ou --visualize - Visualize a imagem spiffs.
- -b <número> ou --block <número> - O tamanho dos blocos do sistema de arquivos em bytes.
- -p <número> ou --page <número> - O tamanho da página do sistema de arquivos em bytes.
- -s <número> ou --size <número> - O tamanho da imagem do sistema de arquivos em bytes.
- - ou --ignore \_rest - Ignora os argumentos restantes.

Página 202

## Página 203

- --version - Exibe as informações da versão.
- -h ou --help - Exibe informações de uso / ajuda
- <image\_file> - O arquivo que contém (ou já contém) a imagem spiffs.

Veja também:

- Github: [igrf / mkspiffs](https://github.com/igrf/mkspiffs)

### A arquitetura de suporte do Arduino IDE

O Arduino IDE para ESP8266 usa o conceito de um "gerenciador de placa". O pensamento por trás disso, com o número crescente de placas relacionadas ao Arduino por aí, todos com diferentes capacidades e sutilezas, o ato de adicionar suporte para um novo tipo de dispositivo (placa) deve ser genérico e mais fácil. Para esse fim, o suporte foi adicionado em 1.6.4 e posteriores para o arquivo JSON do gerenciador da placa. Este arquivo descreve o conteúdo de uma nova placa e onde "obter" as peças necessárias para a construção de aplicativos.

Para o IDE do Arduino ESP8266, o arquivo JSON da placa pode ser encontrado em:

[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)

Se examinarmos o conteúdo deste arquivo em conjunto com a especificação do Arduino Formato de arquivo de pacote IDE, aprendemos muitas coisas interessantes.

Aqui está o arquivo de 03/08/2015 ...

```
{
  "pacotes": [
    {
      "nome": "esp8266",
      "mantenedor": "Comunidade ESP8266",
      "websiteURL": "https://github.com/esp8266/Arduino",
      "email": "ivan@esp8266.com",
      "ajuda": {
        "online": "http://arduino.esp8266.com/versions/1.6.5-947-g39819f0 / doc / reference.html"
      }
    },
    {
      "plataformas": [
        {
          "nome": "esp8266",
          "arquitetura": "esp8266",
          "versão": "1.6.5-947-g39819f0",
          "categoria": "ESP8266",
          "url": "http://arduino.esp8266.com/versions/1.6.5-947-g39819f0/esp8266-1.6.5-947-g39819f0.zip",
          "archiveFileName": "esp8266-1.6.5-947-g39819f0.zip",
          "checksum": "SHA-"
        }
      ]
    }
  ]
}
```

```
256: 79a395801a94e77f4855f3921b9cc127d679d961ec207e7fb89f90754123d66a",
  "tamanho": "2295584",
  "ajuda": {
    "online": "http://arduino.esp8266.com/versions/1.6.5-947-
g39819f0 / doc / reference.html "
```

**Página 204**

```
},
"Pranchas": [
  {
    "name": "Módulo ESP8266 Genérico"
  },
  {
    "name": "Olimex MOD-WIFI-ESP8266 (-DEV)"
  },
  {
    "name": "NodeMCU 0.9 (Módulo ESP-12)"
  },
  {
    "name": "NodeMCU 1.0 (Módulo ESP-12E)"
  },
  {
    "name": "Adafruit HUZZAH ESP8266 (ESP-12)"
  },
  {
    "name": "SweetPea ESP-210"
  }
],
"toolsDependencies": [
  {
    "packager": "esp8266",
    "nome": "esptool",
    "versão": "0.4.5"
  },
  {
    "packager": "esp8266",
    "nome": "xtensa-lx106-elf-gcc",
    "versão": "1.20.0-26-gb404fb9"
  }
],
"Ferramentas": [
  {
    "nome": "esptool",
    "versão": "0.4.5",
    "sistemas": [
      {
        "host": "i686-mingw32",
        "url": "https://github.com/igrr/esptool-ck/releases/download/0.4.5/esptool-
        .4.5-win32.zip",
        "archiveFileName": "esptool-0.4.5-win32.zip",
        "checksum": "SHA-
256: 1b0a7d254e74942d820a09281aa5dc2af1c8314ae5ce1a5abb0653d0580e531b",
        "tamanho": "17408"
      },
      {
        "host": "x86_64-apple-darwin",
        "url": "https://github.com/igrr/esptool-ck/releases/download/0.4.5/esptool-
        0.4.5-osx.tar.gz",
        "archiveFileName": "esptool-0.4.5-osx.tar.gz",
        "checksum": "SHA-
256: 924d31c64f4bb9f748c70806dafbabb15e5eb80afcdde33715f3ec884be1652d",
        "tamanho": "11359"
      }
    ]
  }
]
```

**Página 205**

```
{
  "host": "i386-apple-darwin",
  "url": "http://arduino.esp8266.com/esptool-0.4.5-l-gfaa5794-osx.tar.gz",
  "archiveFileName": "esptool-0.4.5-l-gfaa5794-osx.tar.gz",
  "checksum": "SHA-
256: 722142071f6cf4d8c02dea42497a747e06abf583d86137a6a256b7db71dc61f6",
  "tamanho": "20751"
},
{
  "host": "x86_64-pc-linux-gnu",
  "url": "https://github.com/igrr/esptool-ck/releases/download/0.4.5/esptool-
0.4.5-linux64.tar.gz",
  "archiveFileName": "esptool-0.4.5-linux64.tar.gz",
  "checksum": "SHA-
256: 4ce799e13fb89f8a8f08a08db77dc3b1362c4486306fe1b3801dee80cf3203",
  "tamanho": "12789"
},
{
  "host": "i686-pc-linux-gnu",
  "url": "https://github.com/igrr/esptool-ck/releases/download/0.4.5/esptool-
0.4.5-linux32.tar.gz",
  "archiveFileName": "esptool-0.4.5-linux32.tar.gz",
  "checksum": "SHA-
256: 4aa81b97a470641771cf371e5d470ac92d3b177adbe8263c4aae66e607b67755",
  "tamanho": "12044"
}
],
},
{
  "nome": "xtensa-lx106-elf-gcc",
  "versão": "1.20.0-26-gb404fb9",
  "sistemas": [
    {
      "host": "i686-mingw32",
      "url": "http://arduino.esp8266.com/win32-xtensa-lx106-elf-gb404fb9.tar.gz",
      "archiveFileName": "win32-xtensa-lx106-elf-gb404fb9.tar.gz",
      "checksum": "SHA-
56: 1561ec85cc58cab35cc48bfb0d0087809f89c043112a2c36b54251a13bf781f",
      "tamanho": "153807368"
    },
    {
      "host": "x86_64-apple-darwin",
      "url": "http://arduino.esp8266.com/osx-xtensa-lx106-elf-gb404fb9-2.tar.gz",
      "archiveFileName": "osx-xtensa-lx106-elf-gb404fb9-2.tar.gz",
      "checksum": "SHA-
256: 0cf150193997bd1355e0f49d3d49711730035257bc1ae1eaaad619e56b9e4e6",
      "tamanho": "35385382"
    },
    {
      "host": "i386-apple-darwin",
      "url": "http://arduino.esp8266.com/osx-xtensa-lx106-elf-gb404fb9-2.tar.gz",
      "archiveFileName": "osx-xtensa-lx106-elf-gb404fb9-2.tar.gz",
      "checksum": "SHA-
256: 0cf150193997bd1355e0f49d3d49711730035257bc1ae1eaaad619e56b9e4e6",
      "tamanho": "35385382"
    }
  ]
}
```

Página 205

**Página 206**

```
,
  },
  {
    "host": "x86_64-pc-linux-gnu",
    "url": "http://arduino.esp8266.com/linux64-xtensa-lx106-elf-
gb404fb9.tar.gz",
    "archiveFileName": "linux64-xtensa-lx106-elf-gb404fb9.tar.gz",
    "checksum": "SHA-
```

```

256: 46f057f relevantb320889a26167daf325038912096d09940b2a95489db92431473b7",
    "tamanho": "30262903"
},
{
    "host": "i686-pe-linux-gnu",
    "url": "http://arduino.esp8266.com/linux32-xtensa-lx106-elf.tar.gz",
    "archiveFileName": "linux32-xtensa-lx106-elf.tar.gz",
    "checksum": "SHA-
256: b24817819f0078fb05895a640e806e0aca9aa96b47b80d2390ac8e2d9ddc955a",
    "tamanho": "32734156"
}
]
}]
}
}
}

```

Resolvendo isso, temos um pacote neste arquivo que possui as seguintes seções:

- name - esp8266 - O nome do próprio pacote
- mantenedor - Comunidade ESP8266 - Quem mantém o pacote
- websiteURL - Onde encontrar mais informações sobre este pacote
- e-mail - para quem enviar e-mail para saber mais
- ajuda - Onde obter ajuda online
- plataformas - o conjunto de plataformas em que esta placa é executada
- ferramentas - Os detalhes das ferramentas necessárias

Para as plataformas, descrevemos os detalhes de cada plataforma ... atualmente, há apenas 1:

- nome - esp8266
- arquitetura - esp8266
- versão - O ID da versão deste pacote / plataforma
- categoria - ESP8266
- url - Onde baixar esta plataforma
- archiveFileName - O nome do arquivo

Página 206

## Página 207

- checksum - Um hash que pode ser usado contra o arquivo para ver se ele foi adulterado com
- size - O tamanho em bytes do arquivo
- ajuda - Onde ler a documentação para esta plataforma
- placas - Uma lista de placas associadas à plataforma.
- dependências de ferramentas - Os nomes de ferramentas / componentes adicionais que são necessários

Para as ferramentas, esta é uma lista de ferramentas necessárias para o pacote. Cada ferramenta possui o seguinte:

- nome - O nome lógico da ferramenta
- versão - a versão da ferramenta

- sistemas - uma lista de entradas que definem onde baixar a ferramenta para uma variedade de plataformas incluindo Windows, Linux e OSx.

Com essas informações e uma cópia do arquivo, você poderá ver como alguns dos peças se encaixam.

Quando um pacote é instalado, ele é criado no diretório:

```
C:\Users\<User>\AppData\Roaming\Arduino15\packages
```

Para nossa história ESP8266, o pacote é esp8266 e, portanto, todos os arquivos podem ser encontrados em:

```
C:\Users\<User>\AppData\Roaming\Arduino15\packages
```

vamos chamar isso de raiz.

Abaixo da raiz, encontraremos dois diretórios:

- hardware
- Ferramentas

O diretório de ferramentas contém a raiz de nossas ferramentas necessárias para a execução ... essas são as Compilador C e a ferramenta de upload.

A pasta de hardware contém o resto de nossas informações.

Especificamente as seguintes pastas:

- bootloaders - um mistério ...
- núcleos - Cabeçalho principal e arquivos de origem fornecendo o código sempre vinculado ao esboços. Este é o conjunto principal de wrappers para as bibliotecas do Arduino.
- ferramentas - O Espressif SDK

Página 207

## Página 208

- bibliotecas - As bibliotecas padrão para nosso pacote
- variantes - arquivos de cabeçalho que diferem de acordo com a variante da placa selecionada

E os seguintes arquivos:

- placas
- plataforma
- programadores

Dentro do IDE do Arduino, podemos ativar as configurações detalhadas, o que resulta em detalhes sendo registrados durante a compilação ou upload. A partir deles, podemos aprender mais sobre o que acontece.

Se examinarmos uma instrução de compilação típica, encontraremos o seguinte.

```
xtensa-lx106-elf-gcc
-D_ets_
-DICACHE_FLASH
-U_STRICT_ANSI_
-Itools / sdk // incluem
-c
-g
-x assembler-with-cpp
-MMD
-DF_CPU = 80000000L
-DARDUINO = 10605
-DARDUINO_ESP8266_ESP01
```

```
-DARDUINO_ARCH_ESP8266
-DESP8266
-Icores \ esp8266
-Ivariants \ generic \ core \ esp8266 \ cont.S
-o cont. Então

xtensa-lx106-elf-gcc
-D_ets_
-DICACHE_FLASH
-U_STRICT_ANSI_
-fflags / sdk // incluem
-c
-Os
-g
-Wpointer-arith
-Declaração de função implícita
-Wl, -EL
-fno-inline-functions
-nostdlib
-mlongcalls
-mtext-section-literals
-falign-functions = 4
-MMD
-std = gnu99
```

Página 208

## Página 209

```
-DF_CPU = 80000000L
-DARDUINO = 10605
-DARDUINO_ESP8266_ESP01
-DARDUINO_ARCH_ESP8266
-DESP8266
-Icores \ esp8266
-Ivariantes \ genérico
núcleos \ esp8266 \ cont_util.c
-o cont_util.co
```

O conteúdo do diretório principal são os artefatos vinculados ao seu Arduino esboços.

spiffs	
abi.cpp	
Arduino.h	Arquivo de inclusão primário para aplicativos.
binary.h	Definições binárias para o intervalo 0-255 até 8 bits.
cbuf.h	Tampão circular.
Client.h	
cont.h	
cont.S	
cont_util.c	
core_esp8266_eboot_command.c	
core_esp8266_flash_utils.c	
core_esp8266_i2s.c	
core_esp8266_main.cpp	O principal ponto de entrada no aplicativo ESP.
core_esp8266_noniso.c	
core_esp8266_phy.c	
core_esp8266_postmortem.c	
core_esp8266_si2c.c	
core_esp8266_sigma_delta.c.unused	
core_esp8266_timer.c	
core_esp8266_wiring.c	
core_esp8266_wiring_analog.c	

```

core_esp8266_wiring_digital.c
core_esp8266_wiring_pulse.c
core_esp8266_wiring_pwm.c
core_esp8266_wiring_shift.c
debug.cpp
debug.h
eboot_command.h
Esp.cpp

```

Página 209

**Página 210**

```

Esp.h
esp8266_peri.h
flash_utils.h
HardwareSerial.cpp
HardwareSerial.h
i2s.h
IPAddress.cpp
IPAddress.h
libc_replacements.c
pgmspace.cpp
pgmspace.h
Print.cpp
Print.h
Printable.h
Server.h
sigma_delta.h
stdlib_noniso.h
Stream.cpp
Stream.h
Tone.cpp
twi.h
Udp.h
Updater.cpp
Updater.h
user_config.h
Wcharacter.h
wiring_private.h
Wmath.cpp
Wstring.cpp
Wstring.h

```

Quando olhamos como um aplicativo é carregado, vemos um comando semelhante ao Segue:

```
esptool.exe -vv -cd ck -cb 115200 -cp COM11 -ca 0x00000 -cf ESP_I2CScanner.cpp.bin
```

---

**Página 211****Construindo aplicativos ESP Arduino usando o IDE Eclipse**

Agora nossas cabeças vão doer muito ... não há maneira fácil de passar por isso ... mas a história é importante e os resultados são ótimos.

Até agora, vimos que podemos construir programas ESP usando um compilador C e o Espressif SDK. Também vimos que podemos construir esses programas dentro de um Eclipse ambiente... também em relação ao Espressif SDK. Acabamos de examinar a noção de construir programas usando o Arduino IDE, que fornece mapeamentos para muitos dos Bibliotecas Arduino implementadas para ESP. Agora vamos voltar a usar o Eclipse, mas desta vez como uma alternativa ao IDE do Arduino, mas ainda usando as bibliotecas do Arduino para construir Programas ESP "com sabor Arduino".

A chave para esta história é o excelente conjunto de ferramentas Open Source Eclipse para a construção do Arduino encontrado aqui:

Este conjunto de plug-ins para a estrutura do Eclipse aproveita um ambiente Arduino existente como o que acabamos de construir, que inclui o suporte ESP. Os plug-ins interroguem a configuração do IDE do Arduino e forneçam as ferramentas de construção e edição usadas lá.

Antes de prosseguir, é vital que você faça com que o IDE do Arduino ESP funcione seguindo todas as instruções necessárias para construir soluções usando esse ambiente. Isso é um pré-requisito para fazer o ambiente Eclipse funcionar.

Temos duas opções para fazer o ambiente Eclipse funcionar. O primeiro é para baixe um ambiente Eclipse totalmente preparado que inclui as ferramentas de desenvolvimento C e os plug-ins para suporte ao Arduino. Este é o mais fácil ... no entanto, também é provável que você tem uma estrutura Eclipse existente já instalada que pode desejar reutilizar ou ampliar. O Eclipse deve ser um ambiente extensível que fornece uma estrutura em que plug-ins adicionais podem ser adicionados conforme necessário. Nesse padrão, podemos baixe a estrutura Eclipse mais recente (Mars) e, em seguida, inclua os plug-ins relevantes.

Aqui está um exemplo de preparação usando um download do Eclipse pré-construído.

Primeiro baixe uma compilação noturna atual ... e extraia seu conteúdo. Observe que o mantenedor está distribuindo arquivos no formato tar.gz. eu uso [7-Zip](#) para descomprimir. O o tamanho do download é de cerca de 170 MB, portanto, certifique-se de fazer o download mais cedo ou mais tarde. Certifique-se de baixar a versão correta do ambiente Eclipse que corresponde à versão do Java que você instalou. Por exemplo, se você tiver 32 bits Java instalado, não baixe a versão de 64 bits do Eclipse. Se você fizer e tentar iniciar o Eclipse, você obterá erros que terá que examinar apenas para encontrar enterrados em logs que existe uma incompatibilidade. Se você cometer o erro, a mensagem que você recebe pode ser parecido com:

Como você pode ver, não é fácil perceber o que deu errado.

Quando estiver pronto para lançar, inicie o programa chamado "eclipseArduinoIDE".

Se tudo tiver corrido bem, veremos o seguinte:

Página 212

---

**Página 213**

Agora é hora de configurar o ambiente Eclipse para aprender sobre nosso Arduino meio Ambiente. As receitas a seguir são usadas para superar alguns bugs, portanto, podem mudar ao longo do tempo ...

Primeiro, precisamos dizer ao Eclipse onde ele pode encontrar nosso ambiente Arduino.

Abra Preferências e selecione Arduino :

Página 213

---

**Página 214**

Precisamos alterar algumas das configurações.

Para o caminho do Arduino IDE, aponte para o diretório raiz onde o seu Arduino IDE está instalado.

Este deve ser o diretório que contém o seguinte:

Página 214

---

## Página 215

A próxima parte é um pouco mais complicada. Precisamos fornecer valores para " Biblioteca privada caminho "e" Caminho de hardware privado ". Esses diretórios são os diretórios para o Pacote Arduino ESP e **NÃO** o Arduino nativo. Você os encontrará no seguinte diretórios:

C:\ Usuários \<Seu ID do usuário>\ AppData\ Roaming\ Arduino15\ packages\ esp8266\ hardware\ esp8266\ <Seu Versão>\ bibliotecas

C:\ Users\ <Your Userid> AppData\ Roaming\ Arduino15\ packages\ esp8266\ hardware

Eu também recomendo que você altere seu " Build before upload " para " Yes " neste momento.

Depois de entrar, sua tela pode ficar parecida com.

Página 215

---

**Página 216**

Ao aplicar as alterações feitas aqui, você pode ver o seguinte aviso algumas vezes:

Clique em " Sim " para ignorar os avisos e seguir em frente.

Em seguida, temos que adicionar " \*.ino " como um novo tipo de arquivo C ++ ... fazemos isso novamente nas preferências:

Página 216

---

## Página 217

Na caixa de diálogo resultante, adicione o seguinte:

Estamos chegando perto. Agora temos alguns ajustes finais a fazer. Encontre o arquivo denominado " platform.txt " que está localizado em:

C:\ Usuários \<Seu ID do usuário>\ AppData\ Roaming\ Arduino15\ packages\ esp8266\ hardware\ esp8266\ <Version>\ platform .TXT

Edite este arquivo com seu editor de texto e encontre a linha que diz:

```
tools.esptool.upload.pattern = "{path} / {cmd}" {upload.verbose} -cd {upload.resetmethod}  
-cb {upload.speed} -cp "{serial.port}" -ca 0x00000 -cf "{build.path} /  
{build.project_name} .bin "
```

e mude para:

```
tools.esptool.upload.pattern = "{path} / {cmd}" -vv -cd {upload.resetmethod} -cb  
{upload.speed} -cp "{serial.port}" -ca 0x00000 -cf "{build.path} /  
{build.project_name} .bin "
```

( Mudamos " {upload.verbose} " para " -vv ")

Página 217

---

**Página 218**

Salve o arquivo.

Agora estamos prontos para construir um uso do nosso ambiente.

Na janela principal, crie um novo " Sketch ".

Observe que também podemos fazer isso por meio do padrão Eclipse> Novo Projeto

A primeira página do assistente de criação de projeto é o nome que desejamos dar ao nosso projeto:

Página 218

---

**Página 219**

A seguir, fornecemos algumas configurações básicas. Tome seu tempo com isso. O que provavelmente diferente para você é a "Porta: " que é a porta serial usada para fazer o flash do seu dispositivo:

Na conclusão do assistente (supondo que você tenha considerado o restante das opções como padrão)  
... você terá um projeto que se parece com:

Página 219

---

**Página 220**

Edite o arquivo de origem Test1.ino C ++ e adicione seu código.

Aqui você verá o seu pagamento. Agora você está editando sua fonte no profissional Editor C / C ++ que faz parte do Eclipse. Isso inclui assistência de entrada, verificação de sintaxe e destacando (e mais).

Antes de compilar seu projeto, você precisa alterar as configurações específicas do projeto para diga ao projeto onde encontrar seu programa make . No meu ambiente estou usando " mingw32-make ". Você pode ver onde fazer as alterações na captura de tela a seguir.

Nota: Tem que haver uma maneira melhor do que esta ... mas eu queria pegar essa receita antes do que segurar todos enquanto eu mexia com esta pequena pepita:

Página 220

---

**Página 221**

E, finalmente, podemos compilar nosso programa.

Clique no ícone " Verificar ":

Agora a compilação acontecerá. Para a sua primeira construção deste projeto, **todas** as fontes o código de todas as bibliotecas será construído. As compilações futuras apenas compilarião o que mudou. Na minha máquina, a construção levou 51 segundos:

Página 221

---

## Página 222

No entanto, quando agora eu edito meu arquivo de projeto ( Test1.ino ) e recompilo, a reconstrução apenas leva 4 segundos, pois apenas os arquivos que foram alterados precisam ser recompilados.

Após uma construção, um novo diretório chamado " Release " pode ser encontrado, que contém todos os artefatos que foram compilados. Se você quiser forçar uma reconstrução de tudo, simplesmente exclua " Liberar ".

Página 222

---

## Página 223

Agora que você construiu o programa, pode carregá-lo com o ícone de upload:

Isso fará o upload do executável. Infelizmente, não há muito para ver durante o upload acontece, então sente-se e seja paciente. Se você tiver um conector USB → UART conectado ao UART1 do ESP8266, você vai ver o andamento do upload ... mas isso não é essencial (embora eu recomende).

E ... é isso. Agora você está construindo aplicativos ESP8266 usando bibliotecas Arduino em um ambiente Eclipse.

Se desejarmos adicionar uma das bibliotecas fornecidas, podemos selecionar a biblioteca a ser incluída com:

Arduino> Adicionar uma biblioteca ao projeto selecionado

A partir daí, podemos selecionar uma das bibliotecas:

---

**Página 224**

Podemos adicionar bibliotecas externas que existam em arquivos de origem no sistema de arquivos. De  
No menu Arduino, selecione " Adicionar uma pasta de origem ao projeto selecionado ".

Uma caixa de diálogo será apresentada, na qual é possível selecionar o diretório a ser incluído na construção.

---

**Página 225**

Depois de adicionado, torna-se um diretório vinculado no projeto e o conteúdo do diretório será compilado e vinculado.

Observação: a receita anterior foi baseada no Arduino IDE 1.6.5-r2, o estável versão do Arduino para ESP8266 a partir de 2015-07-23 e a compilação de 2015-08-05 do Arduino Eclipse.

Veja também:

- Github: [esp8266 / Arduino](#)
- [Arduino - Eclipse](#)

#### Razões para considerar o uso do Eclipse em vez do Arduino IDE

Como mencionado anteriormente, não há dúvida de que o IDE Arduino é muito mais amigável e consumível que o ambiente Eclipse profissional para pessoas novas na área. Não parece haver nada que não seja possível construir usando o IDE do Arduino isso significaria que seria necessário mudar para o Eclipse. Então, por que então alguém considera usar o Eclipse?

Página 225

---

**Página 226**

Existe uma compensação entre facilidade de uso e riqueza de funções. Por exemplo, Eclipse foi construído em assistência de sintaxe, verificação de erros, referências cruzadas de código, refatoração e muito mais. Nenhuma dessas coisas é "essencial", mas qualquer uma delas pode ser considerado para tornar um trabalho de programação mais fácil se e quando necessário. Se eu precisar renomear uma variável, no Arduino IDE eu tenho que encontrar e substituir manualmente cada ocorrência. Dentro Eclipse, posso refatorar a variável usando um assistente integrado e o IDE faz o trabalho para Eu. Como outro exemplo, se eu não consigo lembrar a sintaxe de um método, no Arduino IDE I iria para a web e procuraria enquanto no Eclipse eu poderia digitar o nome do método

e passo o mouse sobre ele e as ferramentas vão me mostrar as opções possíveis para o parâmetros.

#### Notas sobre o uso do pacote Eclipse Arduino

- Você **não** criar projetos Eclipse que têm espaços em seus nomes. Isso confunde o compilador.
- Se você estiver compilando em uma máquina multi-core, você pode causar as compilações de os arquivos de origem para progredir em paralelo usando, adicionando o " --jobs <num> " parâmetro para seu comando make . Isso fará com que o make execute alguns número de empregos em paralelo. Por exemplo, se você tem uma máquina de 4 núcleos, definindo num ter 4 anos pode ser um bom começo. Este sinalizador funciona com a ferramenta GNU make .
- Às vezes, desejamos escrever esboços que funcionem tanto em um Arduino real quanto em um ESP8266, mas o código deve ser um pouco diferente. Podemos incluir código para ambos

Página 226

---

#### Página 227

arquiteturas usando os #defines chamados ARDUINO\_ARCH\_ESP8266 e / ou esp8266 . Usando isso, podemos #if / #endif seções com base na arquitetura que estamos compilar contra.

- A partir de 2015-08-08, tentativa de usar a função SDK nativa no Arduino Eclipse ambiente não funciona se seus arquivos de origem são " \*.ino ". Parece que " user\_include.h " é incluído automaticamente com o nome da função C ++ mutilado em efeito.
- Recomendo renomear qualquer arquivo " \*.ino " em seu projeto para " \*.cpp ".

#### Bibliotecas Arduino ESP

##### A biblioteca WiFi

O Arduino tem uma biblioteca WiFi para uso com seu escudo WiFi. Uma biblioteca com um similar interface foi fornecida para o ambiente Arduino para o ESP8266.

Para usar a biblioteca WiFi ESP8266, você deve incluir seu cabeçalho:

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

Para ser uma estação e se conectar a um ponto de acesso, execute uma chamada para WiFi.begin (ssid, senha) . Agora precisamos pesquisar WiFi.status () . Quando isso voltar WL\_CONNECTED, então estamos conectados à rede.

Para configurar um ponto de acesso, chamaríamos WiFi.softAP () fornecendo o SSID e informações de senha.

Aqui está um exemplo de como nos conectamos como uma estação:

```
WiFi.mode(WIFI_STA);
WiFi.begin(SSID, SENHA);
if (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial1.println("Falha");
    Retorna;
}
WiFi.printDiag(Serial1);
// Agora estamos conectados como uma estação
```

Veja também:

- [WiFiClient](#)
- [WiFiServer](#)
- [Biblioteca Arduino WiFi](#)

WiFi.begin

Inicie uma conexão WiFi como uma estação.

```
int begin (
    const char * ssid,
```

Página 227

## Página 228

```
const char * passPhrase = NULL,
canal int32_t = 0,
uint8_t bssid [6] = NULL)

int begin (
    char * ssid,
    char * passPhrase = NULL,
    canal int32_t = 0,
    uint8_t bssid [6] = NULL)
```

Comece uma conexão WiFi como uma estação. O parâmetro ssid é obrigatório, mas os outros pode ser deixado como padrão. O valor de retorno é o nosso status de conexão atual.

```
WiFi.beginSmartConfig
bool beginSmartConfig ()
```

```
WiFi.beginWPSConfig
bool beginWPSConfig ()
```

WiFi.BSSID

Recupere o BSSID atual.

```
uint8_t BSSID ()
uint8_t * BSSID (uint8_t networkItem)
```

Recupere o BSSID atual.

WiFi.BSSIDstr

Recupere o BSSID atual como uma representação de string.

```
String BSSIDstr ()
String BSSIDstr (uint8_t networkItem)
```

Recupere o BSSID atual como uma representação de string.

Canal wi-fi

Recupere o canal atual.

```
canal int32_t ()
canal int32_t (uint8_t networkItem)
```

Recupere o canal atual.

Página 228

## Página 229

WiFi.config

Defina a configuração de conexão sem fio.

```
void config (IPAddress local_ip, IPAddress gateway, IPAddress subnet)
void config (IPAddress local_ip, IPAddress gateway, IPAddress subnet, IPAddress dns)
```

Defina a configuração do WiFi usando parâmetros estáticos. Isso desativa o DHCP.

WiFi.disconnect

Desconecte-se de um ponto de acesso.

```
desconectar int (bool wifiOff = false)
```

Desconecte-se do ponto de acesso atual.

WiFi.encryptionType

Retorna o tipo de criptografia do ponto de acesso WiFi verificado.

```
uint8_t encryptionType (uint8_t networkItem)
```

Retorna o tipo de criptografia do ponto de acesso WiFi verificado.

Os valores são um dos:

- ENC\_TYPE\_NONE
- ENC\_TYPE\_WEP
- ENC\_TYPE\_TKIP
- ENC\_TYPE\_CCMP
- ENC\_TYPE\_AUTO

WiFi.gatewayIP

Obtenha o endereço IP do gateway da estação.

```
IPAddress gatewayIP ()
```

Recupere o endereço IP do gateway da estação.

WiFi.getNetworkInfo

Recupere todos os detalhes do item de rede verificado especificado.

```
bool getNetworkInfo (uint8_t networkItem,
    String e_ssid,
    uint8_t & encryptionType,
    int32_t & RSSI,
    uint8_t * & BSSID,
```

Página 229

## Página 230

```
    uint32_t & channel,
    bool & isHidden)
```

Recupere todos os detalhes do item de rede verificado especificado.

WiFi.hostByName

Procure um host por um nome.

```
int hostByName (const char * hostName, IPAddress & result)
```

Procure um host pelo nome e obtenha seu endereço IP. Esta função retorna 1 em caso de sucesso e 0 em caso de falha.

WiFi.hostname

Recupere e defina o nome do host usado por esta estação.

```
String hostname ()
bool hostname (char * hostName)
bool hostname (const char * hostName)
bool hostname (String hostName)
```

WiFi.isHidden

Determine se o item de rede verificado está sinalizado como oculto.

```
bool isHidden (uint8_t networkItem)
```

Determine se o item de rede verificado está sinalizado como oculto.

WiFi.localIP

Obtenha o endereço IP da estação.

```
Endereço IP localIP ()
```

Obtenha o endereço IP da estação. Há um endereço IP separado se o ESP for um ponto de acesso.

Veja também:

- [WiFi.softAPIP](#)

WiFi.macAddress

Obtenha o endereço MAC da interface da estação.

```
uint_t * macAddress (uint8_t * mac)
String macAddress ()
```

Obtenha o endereço MAC da interface da estação.

---

**Página 231**

WiFi.mode

Defina o modo de operação.

modo vazio (modo WiFiMode)

Defina o modo de operação do WiFi. Este é um dos seguintes:

- WIFI\_OFF - Desligue o WiFi
- WIFI\_STA - Seja uma estação WiFi
- WIFI\_AP - Seja um ponto de acesso WiFi
- WIFI\_AP\_STA - ser uma estação WiFi e um ponto de acesso WiFi

Veja também:

- [Definindo o modo de operação](#)

WiFi.printDiag

Registre o estado da conexão sem fio.

void printDiag (impressão e destino)

Registre o estado da conexão sem fio. Podemos passar Serial ou Serial1 como um argumento para registrar os dados na porta serial. Um exemplo de saída é mostrado a seguir:

```
Modo: STA
Modo PHY: N
Canal: 7
AP id: 0
Status: 5
Conexão automática: 0
SSID (7): seuSSID
Senha (8): sua senha
Conjunto BSSID: 0
```

Observe que o valor do status é o resultado de uma chamada wifi\_station\_get\_connect\_status () .

WiFi.RSSI

Recupere o valor RSSI (Received Signal Strength Indicator) da rede digitalizada item.

int32\_t RSSI (uint8\_t networkItem)

Recupere o valor RSSI do item de rede varrido.

WiFi.scanComplete

Determine o status de uma solicitação de varredura anterior.

[Página 231](#)

---

**Página 232**

int8\_t scanComplete ()

Se o resultado for  $\geq 0$ , este é o número de pontos de acesso WiFi encontrados. Caso contrário, o valor é menor que 0 e os códigos são:

- SCAN\_RUNNING - Uma varredura está em andamento.
- SCAN\_FAILED - Uma varredura falhou.

Veja também:

- [WiFi.scanDelete](#)
- [WiFi.scanNetworks](#)

WiFi.scanDelete

Exclua os resultados de uma verificação anterior.

`void scanDelete ()`

Exclua os resultados de uma verificação anterior. Uma solicitação para verificar a rede resulta no alocação de memória. Esta chamada libera essa memória.

Veja também:

- [WiFi.scanComplete](#)
- [WiFi.scanNetworks](#)

WiFi.scanNetworks

Faça a varredura dos pontos de acesso no ambiente.

`int8_t scanNetworks (bool async = false)`

Faça a varredura dos pontos de acesso no ambiente. Podemos realizar isso de forma síncrona ou assíncrono. Em uma chamada síncrona, o resultado é o número de pontos de acesso encontrados.

Veja também:

- [WiFi.scanComplete](#)
- [WiFi.scanDelete](#)

WiFi.smartConfigDone

`bool smartConfigDone ()`

WiFi.softAP

Configure um ponto de acesso.

```
void softAP (const char * ssid)
void softAP (const char * ssid,
            const char * passPhrase,
            canal interno = 1,
            int ssid_hidden = 0)
```

Página 232

## Página 233

O ssid é usado para anunciar nossa rede. A senha é a senha de uma estação deve fornecer para ter autorização de acesso.

WiFi.softAPConfig

`void softAPConfig (IPAddress local_ip, IPAddress gateway, IPAddress sub-rede)`

WiFi.softAPDisconnect

`int softAPdisconnect (bool wifiOff = false)`

WiFi.softAPmacAddress

Obtenha o endereço MAC da interface do ponto de acesso.

```
uint8_t * softAPmacAddress (uint8_t * mac)
```

Obtenha o endereço MAC da interface do ponto de acesso.

WiFi.softAPIP

Obtenha o endereço IP da interface do ponto de acesso.

```
IPAddress softAPIP ()
```

Retorne o endereço IP da interface do ponto de acesso. Há um IP separado para o estação.

Veja também:

- [WiFi.localIP](#)

WiFi.SSID

Recupere o SSID.

```
char * SSID ()
```

```
const char * SSID (uint8_t networkItem)
```

Aqui, recuperamos o SSID da estação atual ou o SSID do ID de rede verificado.

WiFi.status

Recupere o status atual do WiFi.

```
wl_status_t status ()
```

O status retornado será um dos seguintes:

- WL\_IDLE\_STATUS (0)

Página 233

## Página 234

- WL\_NO\_SSID\_AVAIL (1)
- WL\_SCAN\_COMPLETED (2)
- WL\_CONNECTED (3)
- WL\_CONNECT\_FAILED (4)
- WL\_CONNECTION\_LOST (5)
- WL\_DISCONNECTED (6)

WiFi.stopSmartConfig

```
void stopSmartConfig ()
```

WiFi.subnetMask

```
IPAddress subnetMask ()
```

WiFi.waitForConnectResult

Aguarde até que a conexão WiFi seja formada ou falhe.

```
uint8_t waitForConnectResult ()
```

Se formos uma estação, então bloqueie esperando que nos conectemos ou falhemos. O retorno código é o status. Especificamente, esta função observa o status para ver quando se torna algo diferente de WL\_DISCONNECTED . Talvez uma forma mais positiva desta função seria:

```
enquanto (WiFi.status () != WL_CONNECTED) {
    atraso (500);
    Serial.print (".");
}
```

### **WiFiClient**

Esta biblioteca fornece conexões TCP a um parceiro. Uma classe separada fornece UDP comunicações.

Para usar esta biblioteca, você deve incluir " ESP8266WiFi.h ".

Criamos uma instância desta classe e, em seguida, nos conectamos a um parceiro usando o comando connect () método.

WiFiClient

WiFiClient.available

Retorna a quantidade de dados disponíveis para serem lidos.l15summe

Página 234

## **Página 235**

int disponível 0

Retorna a quantidade de dados disponíveis para serem lidos.

WiFiClient.connect

Conecte-se ao host fornecido na porta fornecida usando TCP.

```
int connect (const char * host, uint16_t port)
int connect (IPAddress ip, uint16_t port)
```

Conecte-se ao host fornecido na porta fornecida usando TCP. Esta função retorna 0 em um falha.

WiFiClient.connected

Determine se estamos conectados a um parceiro.

uint8\_t conectado ()

Retorna verdadeiro se conectado e falso caso contrário.

WiFiClient.flush

void flush ()

WiFiClient.getNoDelay

bool getNoDelay ()

WiFiClient.peek

int peek ()

WiFiClient.read

Leia os dados do parceiro.

```
int read()
int read(uint8_t * buf, size_t size)
```

Leia os dados do parceiro. Essas funções leem um único byte ou uma sequência de bytes do parceiro.

WiFiClient.remoteIP

Recupere o endereço IP remoto da conexão.

```
IPAddress remoteIP()
```

Página 235

## Página 236

Recupere o endereço IP remoto da conexão.

WiFiClient.remotePort

Retorne a porta remota que está sendo usada em uma conexão existente.

```
uint16_t remotePort()
```

Retorne a porta remota que está sendo usada em uma conexão existente.

WiFiClient.setLocalPortStart

Defina a porta inicial para alocar portas locais para conexões.

```
void setLocalPortStart(uint16_t port)
```

Defina a porta inicial para alocar portas locais para conexões.

WiFiClient.setNoDelay

```
void setNoDelay(bool nodelay)
```

WiFiClient.status

```
uint8_t status()
```

WiFiClient.stop

Desconecte um cliente.

```
void stop()
```

Desconecte um cliente.

WiFiClient.stopAll

Pare todas as conexões formadas por este cliente WiFi.

```
void stopAll()
```

WiFiClient.write

Escreva dados para o parceiro.

```
size_t write(uint8_t b)
size_t write(const uint8_t * buf, size_t size)
size_t write(T & source, size_t unitSize);
```

Página 236

---

## Página 237

Escreva dados para o parceiro. A primeira função escreve um byte, enquanto a segunda função escreve uma série de caracteres.

### WiFiServer

WiFiServer

Crie uma instância de um servidor escutando na porta fornecida.

WiFiServer (porta uint16\_t)

Crie uma instância de um servidor escutando na porta fornecida. Interessante, parece que depois de criarmos uma instância de servidor em um ESP8266, não há como interromper sua execução.

WiFiServer.available

Recupere um objeto WiFiClient que pode ser usado para comunicações.

WiFiClient disponível (status byte \*)

Recupere o WiFiClient correspondente.

Veja também:

- [WiFiClient](#)

WiFiServer.begin

Comece a ouvir as conexões de entrada.

void begin ()

Comece a ouvir as conexões de entrada. Até que este método seja chamado, o ESP8266 não aceita conexões de entrada. Curiosamente, uma vez chamado, não há óbvio maneira de parar de ouvir. A porta usada para as conexões de entrada é a fornecida quando o objeto WiFiServer foi construído.

WiFiServer.getNoDelay

WiFiServer.hasClient

Retorna verdadeiro se tivermos um cliente conectado.

bool hasClient ()

Página 237

---

## Página 238

```
WiFiServer.setNoDelay
```

```
WiFiServer.status
```

```
WiFiServer.write
```

**AVISO!!** Este método não foi implementado.

```
size_t write (uint8_t b)
size_t write (const uint8_t * buffer, size_t size)
```

Embora presente na interface, esse método ainda não foi implementado.

### Endereço de IP

Uma representação de um endereço IP. Esta classe tem algumas substituições de operador:

[i] - Pega o enésimo byte do endereço. Eu deveria ter 0-3.

### ESP8266WebServer

A classe ESP8266WebServer fornece a implementação principal de um servidor HTTP.

Este é um software que responde às solicitações do navegador. Para usar esta classe, criamos uma instância de um objeto ESP8266WebServer especificando o número da porta TCP na qual ouvirá. A porta padrão para navegadores é a porta 80, portanto, é uma boa escolha.

Uma vez que o objeto foi criado, definimos uma ou mais funções de callback que serão chamadas quando uma conexão do navegador é recebida. A função chamada `on()` é usada para registrar os. Essas funções de retorno de chamada são digitadas no caminho de URL solicitado pelo navegador. Por exemplo, se nosso ESP8266 está sendo executado no endereço IP 192.168.1.2 e o URL do navegador é:

```
http://192.168.1.2/index.html
```

O caminho do URL será "`/index.html`".

Se quisermos enviar uma resposta a uma solicitação nesse URL, registraremos um retorno de chamada função usando esse caminho como uma chave.

Por exemplo:

```
myServer.on ("/ index.html", myFunction);
```

Onde `myFunction` é uma função C com a assinatura:

```
void myFunction ()
```

A função de retorno de chamada, quando chamada, pode usar o objeto ESP8266WebServer para executar uma chamada de método `send()` para enviar uma resposta.

Página 238

---

### Página 239

Se uma solicitação do navegador chega para um caminho de URL que não é explicitamente tratado, uma chamada para uma função de retorno de chamada registrada com o método `onNotFound()` é chamada. Isso pode servir como um pega-tudo para processamento.

Quando um URL contém propriedades de consulta no formato "`x = y`", o número, nomes e valores dessas propriedades estão disponíveis nas funções `args()`, `argName()` e `arg()`. Observação essa codificação de URL não é compatível atualmente, então os dados ainda não podem conter URL inválido personagens.

Quando uma solicitação de um navegador é recebida, deseja-se enviar de volta uma resposta e o maneira de conseguir isso é por meio de uma invocação do método `send()`. Isso leva o código de resposta para o navegador (200 para OK), o tipo de codificação MIME e a carga útil de os dados como parâmetros.

Ao enviar uma solicitação de um navegador para um servidor Web, antecipe um HTTP extra Solicitação GET que deseja recuperar um arquivo chamado "favicon.ico" que é usado para especificar um ícone que representa o site que está sendo acessado. Para lidar com isso, podemos desejar adicione uma função de manipulador parecida com a seguinte:

```
webServer.on ("/favicon.ico", [] () {
    webServer.send (404, "text / plain", "");
});
```

Isso registra um manipulador para o arquivo de ícone e envia de volta uma resposta 404 (não encontrada) para o navegador. Observe o uso de uma função anônima in-line em C++. Sua escolha para usar este estilo de codificação é seu. Pessoalmente, prefiro funções declaradas explicitamente.

Aqui está um exemplo de um aplicativo de servidor da Web:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char * ssid = "mySsid";
const char * password = "myPassword +";

ESP8266WebServer webServer (80);

void logDetails () {

void testHandler () {
    Serial1.println ("testHandler");
    logDetails ();
    webServer.send (200, "text / plain", "Aqui está nossa resposta:" + String (millis ()));
}

void notFoundHandler () {
    Serial1.println ("Manipulador não encontrado");
    logDetails ();
}
```

Página 239

---

## Página 240

```
String methodToString (método HTTPMethod) {
    switch (método) {

        case HTTP_GET:
            retornar "GET";
        case HTTP_POST:
            retornar "POST";
        case HTTP_PUT:
            retornar "PUT";
        case HTTP_PATCH:
            return "PATCH";
        case HTTP_DELETE:
            retornar "DELETE";
    }
    retornar "Desconhecido";
}

void logDetails () {
    Serial1.println ("URL é:" + webServer.uri ());
    Serial1.println ("O método HTTP a pedido era:" +
methodToString (webServer.method ()));
```

```

// Imprime quantas propriedades recebemos e depois imprime seus nomes
// e valores.
Serial1.println ("Número de propriedades da consulta:" + String (webServer.args ()));
int i;
para (i = 0; i < webServer.args (); i++) {
    Serial1.println ("-" + webServer.argName (i) + "=" + webServer.arg (i));
}
}

void setup () {
    Serial1.begin (115200);
    Serial1.println ("Iniciando ...");
    WiFi.begin (ssid, senha);
    // Aguarde a conexão
    enquanto (WiFi.status () != WL_CONNECTED) {
        atraso (500);
        Serial1.print (".");
    }
    Serial1.println ("");
    Serial1.print ("Conectado a");
    Serial1.println (ssid);
    Serial1.print ("endereço IP:");
    Serial1.println (WiFi.localIP ());
    webServer.on ("/ test", testHandler);
    webServer.on ("/ favicon.ico", [] () {
        webServer.send (404, "text / plain", "");
    });

    webServer.onNotFound (notFoundHandler);
    webServer.begin ();
    Serial1.println ("Iniciamos o servidor da Web");
}

```

Página 240

## Página 241

```

void loop () {
    webServer.handleClient ();
}

```

Veja também:

- Wikipedia: [Favicon](#)
- [ESP8266WebServer.on](#)
- [ESP8266WebServer.send](#)

ESP8266WebServer

Construa uma instância de um objeto WebServer.

ESP8266WebServer :: ESP8266WebServer (porta interna)

Construa uma instância da classe. O número da porta fornecido é a porta que será ouvida as solicitações de entrada do navegador.

ESP8266WebServer.arg

Recupere o valor do argumento.

String arg (índice interno)

Para uma propriedade passada em uma string de consulta, aqui podemos recuperar o valor correspondente.

ESP8266WebServer.argName

Recupere o nome nome do argumento.

String argName (índice interno)

Para uma propriedade passada em uma string de consulta, aqui podemos recuperar o nome correspondente.

ESP8266WebServer.args

Recupere o número de propriedades transmitidas em uma string de consulta fornecida com o navegador inquerir.

```
int args ()
```

ESP8266WebServer.begin

Comece a escutar as conexões de entrada do navegador.

```
void begin ()
```

ESP8266WebServer.client

Cliente WiFiClient ()

Página 241

---

## Página 242

ESP8266WebServer.handleClient

Lidar com uma solicitação do cliente (navegador).

```
void handleClient ()
```

Esta é a função que deve ser chamada periodicamente para processar o navegador de entrada solicitações de. Por exemplo, esta é a função que é colocada no corpo do loop () .

ESP8266WebServer.hasArg

Retorne verdadeiro se a propriedade nomeada foi fornecida.

```
bool hasArg (const char * name)
```

O parâmetro name é o nome da propriedade que pode ter sido fornecida como um propriedade em uma string de consulta.

ESP8266WebServer.method

Recupere o método fornecido pela solicitação do navegador original.

```
Método HTTPMethod ()
```

O HTTPMethod pode ser um dos seguintes:

- HTTP\_GET
- HTTP\_POST
- HTTP\_PUT
- HTTP\_PATCH
- HTTP\_DELETE

ESP8266WebServer.on

Registre retornos de chamada para processar solicitações do navegador.

```
void on (const char * uri,
```

```
    ESP8266WebServer :: THandlerFunction handler)
```

```
void ESP8266WebServer :: on (const char * uri,
```

```
    Método HTTPMethod,
```

```
    ESP8266WebServer :: THandlerFunction handler)
```

Registre uma função de retorno de chamada para um URI e método. A primeira variante da função irá lidar com um URI correspondente para todos os métodos, enquanto o segundo nos permite lidar com retornos de chamada para a mesma parte do URI, mas métodos HTTP diferentes.

A função de manipulador tem uma assinatura de:

```
void (manipulador *) ()
```

Página 242

## Página 243

`ESP8266WebServer.onFileUpload`

`ESP8266WebServer.onNotFound`

Registre um retorno de chamada quando nenhum manipulador específico existir.

```
void onNotFound (THandlerFunction fn)
```

Se nenhum retorno de chamada foi explicitamente registrado para uma solicitação de URL de entrada, este retorno de chamada função será chamada como um pega-tudo.

`ESP8266WebServer.send`

Envie uma resposta ao navegador.

```
void send (int code, const char * contentType, const String & content)
```

Envie dados para o navegador. Este é o principal ponto de entrada da resposta.

O parâmetro `code` é o código de resposta HTTP. O valor do código de 200 significa OK.

O parâmetro `contentType` é o conteúdo MIME da carga útil. O conteúdo parâmetro é o conteúdo real a ser enviado.

`ESP8266WebServer.sendContent`

Envie uma string para o navegador. Esta é uma interface de nível inferior e usando `send()` método é a maneira correta de enviar dados de aplicativos.

```
void sendContent (const String & conteúdo)
```

Envie uma string para o navegador. A string passada no conteúdo é usada para transmitir os dados.

`ESP8266WebServer.sendHeader`

Envie um cabeçalho HTTP.

```
void sendHeader (const String & nome, const String & valor, bool primeiro)
```

Adicione um cabeçalho HTTP ao fluxo de saída enviado ao navegador. O parâmetro de nome é o nome do cabeçalho enquanto `value` é o valor do cabeçalho. O primeiro parâmetro diz se o cabeçalho será adicionado ou não no início da lista de cabeçalhos ou no final.

`ESP8266WebServer.setContentLength`

Defina a duração do conteúdo a ser enviado.

```
void setContentLength (size_t contentLength)
```

---

**Página 244**

ESP8266WebServer.streamFile

Transmita o conteúdo de um arquivo.

`size_t streamFile (T & file, const String & contentType)`

ESP8266WebServer.upload

`HTTPUpload e upload ()`

ESP8266WebServer.uri

Recupere o URI atual fornecido pelo navegador.

`String uri ()`

Este é o valor principal em um manipulador de retorno de chamada de solicitação, onde podemos determinar o caminho URI relativo.

### Biblioteca ESP8266mDNS

Anuncie-nos via Multicast DNS. A biblioteca chamada " ESP8266mDNS " deve ser adicionada ao projeto e o include chamado " ESP8266mDNS.h " deve ser incluído.

Examinando esta biblioteca, parece **não** usar as funções do ESP8266 SDK para mDNS.

Isso parece estranho.

Veja também:

- [Sistemas de nomes de domínio multicast](#)

MDNS.addService

`void addService (char * service, char * proto, uint16_t port);`

`void addService (const char * service, const char * proto, uint16_t port)`

`void addService (String serviço, String proto, porta uint16_t)`

MDNS.begin

Comece a responder às solicitações mDNS.

`bool begin (const char * hostName);`

`bool begin (const char * hostName, IPAddress ip, uint32_t ttl = 120)`

Observe que a versão da função com mais do que o nome do host é implementada por ignorando os outros parâmetros. A função retorna verdadeiro em caso de sucesso.

MDNS.update

`void update ();`

---

**Página 245**

### I2C - Fio

A classe Wire fornece suporte I2C. Para usar esta classe, importe " Wire.h " para

seu esboço. Quando usamos esta classe, uma instância global chamada " Wire " é disponibilizada para nós. Um fio é chamado de SCL, que fornece o relógio, enquanto o outro fio é chamado SDA e é o barramento de dados. No Arduino, a biblioteca suporta ser um mestre ou um escravo, entretanto, na implementação atual, apenas ser mestre é suportado.

Para usar essa classe, primeiro definimos quais pinos devem ser usados e, em seguida, iniciamos o serviço.

```
Wire.begin (SDApin, SCLpin);
```

Para enviar dados, começamos uma transmissão usando beginTransmission () :

```
Wire.beginTransmission (deviceAddress);
```

agora podemos escrever alguns dados ...

```
Wire.write (valor);
```

e, finalmente, conclua a transmissão:

```
Wire.endTransmission ();
```

se quisermos receber dados do escravo, podemos chamar requestFrom () :

```
Wire.requestFrom (deviceAddress, size, true);
```

e os dados podem ser lidos usando as funções available () e read () .

Veja também:

- [Trabalhando com I2C](#)
- [Arduino - Wire Library](#)

Wire.available

Determine o número de bytes disponíveis para leitura.

int disponível (vazio)

Determine o número de bytes disponíveis para leitura.

Veja também:

- [Wire.read](#)
- [Wire.requestFrom](#)

Wire.begin

Initialize a biblioteca de fios.

```
void begin (int SDApin, int SCLpin)
void begin ()
void begin (uint8_t address)
void begin (endereço interno)
```

## Página 246

Initialize a biblioteca de fios. Quando um endereço é fornecido, somos escravos, caso contrário, somos um mestre. Também podemos especificar os pinos a serem usados para SDA e SCL. Se formos um mestre e nenhum pino é fornecido, usaremos os pinos padrão.

**AVISO!!** - Parece que **NÃO** há suporte para realmente ser um escravo e apenas um mestre é suportado neste momento.

**AVISO!!** - No código atual, o parâmetro de endereço é ignorado !!

Veja também:

- [Wire.pins](#)

Wire.beginTransmission

Começando um bloco de transmissão para um escravo.

```
void beginTransmission (endereço uint8_t)
void beginTransmission (endereço interno)
```

Comece a ideia de enviar uma transmissão para um dispositivo escravo com o endereço fornecido.

Outras chamadas para write () enfileiram os dados a serem transmitidos, os quais são finalmente executados com um chamada para endTransmission () .

Wire.endTransmission

Fim do agrupamento de uma transmissão.

```
uint8_t endTransmission (void) // O padrão é sendStop = true
uint8_t endTransmission (uint8_t sendStop)
```

Termine o agrupamento de uma transmissão e execute a transmissão real. Os códigos de retorno estão:

- 0 - transmitido corretamente
- 2 - NACK recebido na transmissão do endereço
- 3 - NACK recebido na transmissão de dados
- 4 - linha ocupada

Wire.flush

Descarte todos os dados não lidos ou não gravados.

```
void TwoWire :: flush (void)
```

Descarte todos os dados não lidos ou não gravados. Uma chamada para available () retornará 0 e uma chamada para endTransmission () não transmitirá dados.

Página 246

## Página 247

Wire.onReceive

Um retorno de chamada quando estamos no papel de um escravo e recebemos uma transmissão de um mestre.

```
void onReceive (void (* function) (int numBytes))
```

Um retorno de chamada quando o escravo recebe uma transmissão de um mestre.

**AVISO!!** - Esta função não está implementada.

Wire.onReceiveService

Não implementado.

```
void onReceiveService (uint8_t * inBytes, int numBytes)
```

Não implementado.

**AVISO!!** - Esta função não está implementada.

Wire.onRequest

Um retorno de chamada invocado quando estamos no papel de um escravo e um mestre solicita dados de nós.

```
void onRequest (void (* function) (void))
```

Um retorno de chamada invocado quando estamos no papel de um escravo e um mestre solicita dados de nós.

**AVISO!!** - Esta função não está implementada.

Wire.onRequestService

Não implementado.

void onRequestService (void)

Não implementado.

**AVISO!!** - Esta função não está implementada.

Wire.peek

Dê uma olhada no próximo byte.

int peek (vazio)

Dê uma espiada no próximo byte, se houver algum disponível. Um retorno de -1 se não houver byte disponível.

Wire.pins

**AVISO!!** - Esta função foi substituída por begin (sda, scl) .

---

## Página 248

Defina os pinos padrão para SDA e SCL.

pinos vazios (int sda, int scl)

Defina os pinos padrão para SDA e SCL.

Veja também:

- [wire.begin](#)

Wire.read

Leia um único byte.

int read (void)

Leia um único byte do ônibus. Um valor de -1 é retornado se não houver byte disponível.

Veja também:

- [wire.available](#)
- [wire.requestFrom](#)

Wire.requestFrom

Solicite dados de um escravo.

```
size_t requestFrom (endereço uint8_t, size_t size, bool sendStop)
uint8_t requestFrom (endereço uint8_t, quantidade uint8_t, uint8_t sendStop)
uint8_t requestFrom (endereço uint8_t, quantidade uint8_t)
uint8_t requestFrom (endereço interno, quantidade interna)
uint8_t requestFrom (endereço interno, quantidade interna, envio interno interno)
```

Solicite dados de um escravo. Este método deve ser chamado quando estamos desempenhando o papel de um mestre. O parâmetro de endereço define o endereço do escravo para o dispositivo que deve responder. Se sendStop for true, uma mensagem de parada é transmitida liberando o I2C autocarro. Se sendStop for false, uma mensagem de reinicialização é transmitida evitando outro barramento mestre de assumir o controle.

O parâmetro de quantidade indica quantos bytes desejamos receber.

O valor de retorno é o número de bytes que foram recebidos.

Veja também:

- [Wire.read](#)
- [Wire.available](#)

Wire.setClock

Defina a frequência do relógio.

`void setClock (frequência uint32_t)`

Defina a frequência do relógio. Sempre chame `setClock ()` APÓS uma chamada para `começar ()`.

Página 248

## Página 249

Wire.write

Grave um ou mais bytes no escravo.

`size_t write (dados uint8_t)`  
`size_t write (const uint8_t * data, size_t quantity)`

Grave um ou mais bytes no escravo.

### Biblioteca de ticker

Esta biblioteca configura funções de retorno de chamada que são chamadas após um período de tempo. Para usar isto biblioteca você deve incluir " `Ticker.h` ". Por exemplo:

```
#include <Ticker.h>
```

```
void timerCB () {
    Serial1.println ("Assinale ...");
}

void setup ()
{
    Serial1.begin (115200);
    ticker.attach (5, timerCB);
    Serial1.println ("Ticker anexado");
}
```

`Ticker`

Uma instância de um objeto Ticker. Normalmente, isso é criado como um global, como:

```
Ticker myTicker;
```

anexar

Anexe uma função de retorno de chamada ao ticker.

```
void attach (float segundos,
            callback_t callback)
void attach (float segundos,
            void (* callback) (TArg),
            TArg arg)
```

Anexe uma função de retorno de chamada ao ticker de forma que o retorno de chamada seja invocado a cada período de segundos. Observe que segundos são flutuantes, então podemos especificar valores como 0,1 para indicar um retorno de chamada a cada 1/10 de segundo (100 milissegundos).

O `callback_t` é definido como:

```
void (* callback_t) (void)
```

**Página 250**

attach\_ms

Anexe uma função de retorno de chamada ao ticker.

```
void attach_ms (uint32_t milissegundos,
                callback_t callback)
void attach_ms (uint32_t milissegundos,
                void (*callback) (TArg), TArg arg)
```

Anexe uma função de retorno de chamada ao ticker de forma que o retorno de chamada seja invocado a cada período de milissegundos. Apenas um anexo pode ser feito em um cronômetro.

separar

Retire um relógio do cronômetro.

```
void detach ()
```

Desanexe uma função de retorno de chamada do cronômetro. Nenhum retorno de chamada mais ocorrerá.

uma vez

Anexe uma função de retorno de chamada ao cronômetro para um disparo único.

```
vazio uma vez (segundos flutuantes,
                callback_t callback)
vazio uma vez (segundos flutuantes,
                void (*callback) (TArg),
                TArg arg)
```

Anexe uma função de retorno de chamada ao cronômetro para um disparo único. Observe que segundos é uma flutuação então podemos especificar valores como 0,1 para indicar um retorno de chamada a cada 1/10 de segundo (100 milissegundos).

once\_ms

Anexe uma função de retorno de chamada ao cronômetro para um disparo único.

```
void once_ms (uint32_t milissegundos,
                callback_t callback)
void once_ms (uint32_t milissegundos,
                void (*callback) (TArg),
                TArg arg)
```

Anexe uma função de retorno de chamada ao cronômetro para um disparo único.

### Biblioteca EEPROM

Esta biblioteca nos permite armazenar e recuperar dados de armazenamento que persistem em um reinicialização do dispositivo. Um objeto singleton denominado EEPROM é pré-fornecido para uso.

**EEPROM.begin**

Comece o processo de escrita ou leitura da EEPROM. O tamanho é a quantidade de armazenamento com o qual desejamos trabalhar.

```
void begin (size_t size)
```

**EEPROM.commit**

As alterações nos dados são comprometidas com a EEPROM. Um retorno verdadeiro indica sucesso enquanto um retorno de false indica uma falha.

```
bool commit ()
```

**EEPROM.end**

Confirma as alterações nos dados e, em seguida, libera qualquer armazenamento local. Sem mais leituras ou as gravações devem ser tentadas até depois da próxima chamada begin () .

```
void end ()
```

**EEPROM.get**

Leia uma estrutura de dados do armazenamento.

```
T & get (endereço interno, T & t)
```

**EEPROM.getDataPtr**

Recupere um ponteiro para o armazenamento que iremos ler ou escrever.

```
uint8_t * getDataPtr ()
```

**EEPROM.put**

Coloque uma estrutura de dados para armazenamento.

```
T & put const (endereço int, T & t const)
```

**EEPROM.read**

Leia um byte do armazenamento.

```
uint8_t read (int address)
```

**EEPROM.write**

Grave um byte no armazenamento.

```
void write (int address, uint8_t value)
```

**SPIFFS**

FS é a biblioteca do sistema de arquivos que oferece a capacidade de ler e gravar arquivos de dentro o ambiente Arduino ESP. Mas espere ... ler e gravar arquivos para onde? Não há "dirige" em um ESP8266. Os dados dos arquivos são lidos e gravados em uma área do flash memória e, uma vez que o tamanho do flash é relativamente pequeno (4 MB ou mais no máximo), isso é um limite superior do tamanho máximo dos arquivos cumulativos ... no entanto, isso ainda é mais do que

o suficiente para muitos padrões de uso, como salvar estado, logs ou informações de configuração.

SPIFFS.begin

Comece a trabalhar com o sistema de arquivos SPIFFS.

bool begin ()

Retorna verdadeiro em caso de sucesso e falso em caso contrário.

SPIFFS.open

Abra o arquivo nomeado.

Arquivo aberto (caminho const char \*, modo const char \*)

Arquivo aberto (const String e caminho, modo const char \*)

O modo define como desejamos acessar o arquivo. As opções são:

- r - Leia o arquivo. O arquivo deve existir.
- w - Grava no arquivo. Truncar o arquivo se ele existir.
- a - Anexar ao arquivo.
- r + - Lê e grava o arquivo.
- w + - Ler e gravar o arquivo.
- a + - Ler e gravar o arquivo.

Veja também:

- [File.close](#)

SPIFFS.openDir

Abra um diretório.

Dir openDir (const char \* path)

Dir openDir (const String e caminho)

SPIFFS.remove

Remova / exclua um arquivo do sistema de arquivos.

bool remove (const char \* path)

bool remove (const String e caminho)

Página 252

## Página 253

SPIFFS.rename

Renomeie um arquivo.

bool renomear (const char \* pathFrom, const char \* pathTo)

bool renomear (const String & pathFrom, const String & pathTo)

Arquivo disponível

Retorna o número de bytes que estão disponíveis no arquivo a partir da posição atual do arquivo ao seu tamanho máximo.

int disponível 0

File.close

Feche um arquivo aberto anteriormente.

void close ()

Nenhuma leitura ou escrita adicional deve ser realizada.

File.flush

**Limpe o arquivo.**

void flush ()

Nome do arquivo

**Recupere o nome do arquivo.**

const char \* name ()

File.peek

Espie o próximo byte de dados no arquivo sem consumi-lo.

int peek ()

File.position

**Recupere a posição atual do ponteiro do arquivo.**

size\_t position ()

File.read

Leia os dados do arquivo.

Página 253

## Página 254

int read ()

size\_t lido (uint8\_t \* buf, size\_t size)

Leia um único byte de dados ou um buffer de dados do arquivo.

File.seek

Altera a posição atual do ponteiro do arquivo.

bool seek (uint32\_t pos, modo SeekMode)

O modo pode ser um dos seguintes:

- SeekSet - altera a posição do ponteiro do arquivo para o valor absoluto.
- SeekCur - altera a posição do ponteiro do arquivo para ser relativa à posição atual.
- SeekEnd - altera a posição do ponteiro do arquivo para ser relativa ao final do arquivo.

Tamanho do arquivo

**Recupere o tamanho máximo do arquivo.**

size\_t size ()

File.write

**Grave dados no arquivo.**

size\_t write (uint8\_t c)

size\_t write (uint8\_t \* buf, size\_t size)

Grave um único byte ou um buffer de bytes no arquivo no ponteiro do arquivo atual

posição.

Dir.fileName

Recupere o nome do arquivo.

`String fileName ()`

Dir.next

`bool next ()`

Dir.open

Arquivo aberto (modo const char \*)

Arquivo aberto (string e caminho, modo const char \*)

Dir.openDir

`Dir openDir (const char * path)`

`Dir openDir (String e caminho)`

Página 254

## Página 255

Dir.remove

Dir.rename

### Biblioteca ESP

Uma classe foi fornecida chamada ESP que fornece ESP8266 específico do ambiente funções. Você deve perceber que o uso dessas funções resultará em seus aplicativos não ser portátil para o Arduino (se for esse o desejo).

ESP.deepSleep

`void deepSleep (uint32_t time_us, modo WakeMode)`

ESP.eraseConfig

`bool eraseConfig ()`

ESP.getBootMode

`uint8_t getBootMode ()`

ESP.getBootVersion

`uint8_t getBootVersion ()`

ESP.getChipId

`uint32_t getChipId ()`

ESP.getCpuFreqMHz

`uint8_t getCpuFreqMHz ()`

ESP.getCycleCount

`uint32_t getCycleCount ()`

```
ESP.getFlashChipId  
uint32_t getFlashChipId ()
```

```
ESP.getFlashChipMode  
FlashMode_t getFlashChipMode ()
```

Página 255

---

## Página 256

```
ESP.getFlashChipRealSize  
uint32_t getFlashChipRealSize ()
```

```
ESP.getFlashChipSize  
uint32_t getFlashChipSize ()
```

```
ESP.getFlashChipSizeByChipId  
uint32_t getFlashChipSizeByChipId ()
```

```
ESP.getFlashChipSpeed  
uint32_t getFlashChipSpeed ()
```

```
ESP.getFreeHeap  
uint32_t getFreeHeap ()
```

```
ESP.getFreeSketchSpace  
uint32_t getFreeSketchSpace ()
```

```
ESP.getResetInfo  
String getResetInfo ()
```

```
ESP.getResetInfoPtr  
struct rst_info * getResetInfoPtr ()
```

```
ESP.getSdkVersion  
Recupere a representação de string do SDK que está sendo usado.  
const char * getSdkVersion ()
```

```
ESP.getSketchSize  
uint32_t getSketchSize ()
```

---

**Página 257**

```
ESP.getVcc  
uint16_t getVcc ()
```

```
ESP.reset  
void EspClass :: reset ()
```

```
ESP.restart  
void EspClass :: restart ()
```

```
ESP.updateSketch  
bool updateSketch (Stream & in,  
tamanho uint32_t,  
bool restartOnFail,  
bool restartOnSuccess)
```

```
ESP.wdtDisable  
void wdtDisable ()
```

```
ESP.wdtEnable  
void wdtEnable (uint32_t timeout_ms)
```

```
ESP.wdtFeed  
void wdtFeed ()
```

---

**Página 258**

Embora se acredite que esta biblioteca possa ser idêntica à biblioteca Arduino String, eu creio que é tão essencial entender que vou listar os métodos novamente.

### Fragmento

Construtor

```
String (const char * cstr = "");
String (const String & str)
String (char c)
String (unsigned char, unsigned char base = 10)
String (int, unsigned char base = 10)
String (long, unsigned char base = 10)
String (longo sem sinal, base de caractere sem sinal = 10)
String (float, unsigned char decimalPlaces = 2)
String (double, unsigned char decimalPlaces = 2)
```

Crie uma instância da classe String semeada com vários inicializadores de tipo de dados.

String.c\_str

Recupere uma representação de string C.

```
const char * c_str ()
```

String.reserve

reserva de char unsigned (unsigned int size)

String.length

Retorne o comprimento da string.

```
comprimento int sem sinal ()
```

Retorne o comprimento da string.

String.concat

```
concat de char unsigned (const String & str)
concat de char unsigned (const char * cstr)
concat de char unsigned (char c)
Unsigned char concat (unsigned char c)
concat de char unsigned (int num)
unsigned char concat (unsigned int num)
concat de char unsigned (long num)
Unsigned char concat (unsigned long num)
concat de char unsigned (float num)
concat de char unsigned (duplo num)
```

String.equalsIgnoreCase

```
char unsigned equalsIgnoreCase (const String & s) const;
```

String.startsWith

Determine se esta string começa ou não com outra string.

Página 258

### Página 259

```
unsigned char startsWith (const String & prefix)
unsigned char startsWith (const String & prefix, unsigned int offset)
```

String.endsWith

```
unsigned char endsWith (const String e sufixo)
```

String.charAt

```
char charAt (índice int não assinado)
```

String.setCharAt

```
void setCharAt (índice int não assinado, caractere c)
```

String.getBytes

```
void getBytes (unsigned char * buf, unsigned int bufsize, unsigned int index = 0)
```

```
String.toCharArray
void toCharArray (char * buf, unsigned int bufsize, unsigned int index = 0)
```

String.indexOf

Encontre a posição de uma string ou caractere dentro da string atual.

```
int indexOf (char ch)
int indexOf (char ch, unsigned int fromIndex)
int indexOf (const String & str)
int indexOf (const String & str, unsigned int fromIndex)
```

Encontre a posição de uma string ou caractere dentro da string atual. Se a partida não for encontrado, -1 é retornado, caso contrário, a posição de início da partida é retornada.

```
String.lastIndexOf
int lastIndexOf (char ch)
int lastIndexOf (char ch, unsigned int fromIndex)
int lastIndexOf (const String & str)
int lastIndexOf (const String & str, unsigned int fromIndex)
```

String.substring

Recupere uma substring de dentro da string atual.

```
String.substring (sem sinal int beginIndex)
String.substring (sem sinal int beginIndex, sem sinal int endIndex)
```

Recupere uma substring de dentro da string atual.

```
String.replace
void replace (char find, char replace)
void replace (const String & find, const String & replace)
```

```
String.remove
void remove (indice int não assinado)
void remove (unsigned int index, unsigned int count)
```

```
String.toLowerCase
void toLowerCase (void)
```

```
String.toUpperCase
void toUpperCase (void)
```

Página 259

## Página 260

```
String.trim
vazio trim (vazio)
```

```
String.toInt
long toInt (vazio)
```

```
String.toFloat
flutuar para flutuar (vazio)
```

## Programação com JavaScript

O JavaScript é uma linguagem interpretada de alto nível. Alguns de seus principais construtos são digitação livre, orientação a objetos, suporte a funções lambda, suporte a fechamentos e, a maioria importante, tornou-se a linguagem da web. Se alguém está escrevendo um navegador hospedado aplicação, então é certo que será escrito em JavaScript. Mas e não ambientes do navegador? Por um tempo, o JavaScript estava comendo no lado do servidor código por meio de projetos como Node.js. Como uma linguagem para executar o código do servidor, tem um conjunto significativo de recursos para realizar esse recurso. Especificamente, ele apóia um evento paradigma de arquitetura dirigida. Em JavaScript, podemos registrar funções a serem chamadas de volta quando os eventos são detectados. Esses retornos de chamada podem ser definidos como funções simples em linha sobre o que fazer. Nestes exemplos inventados, ilustramos isso:

```
httpServer.on ("/ path1", function () {  
    // Faça algo para / path1  
    httpServer.send (resposta);  
});  
  
ou  
  
socket.accept (port, function (newSocket) {  
    newSocket.on ("receber", função (dados) {  
        imprimir ("Recebemos novos dados:" + dados);  
        newSocket.send ("Pegamos os dados", function () {  
            newSocket.close ();  
        });  
    });  
});
```

E se pudermos implementar um bom modelo de JavaScript, ele mapeia de maneira excelente para o ESP8266 modelo do mundo que é impulsionado por eventos por meio de callbacks. Este mapeamento não será fácil ... mas os planos estão em andamento.

Espruino é um projeto de código aberto para fornecer um tempo de execução de JavaScript para dispositivos. Foi implementado para os processadores ARM Cortex M3 / M4 e outros.

A questão agora é ... pode ser usado para o ESP8266? Um projeto ativo está tentando para fazer exatamente isso.

Veja também:

Página 260

---

## Página 261

- [Espruino](#)

### Smart.js

Smart.js é uma implementação de JavaScript para uma variedade de plataformas embarcadas. Isto fornece suporte nativo para a maioria das funções de I / O comuns, incluindo GPIO, SPI, I2C, PWM. Possui suporte de rede através de servidor WiFi e HTTP. Suporta um arquivo sistema.

---

**Página 262**

Para configurar o WiFi, execute:

```
wifi.setup (ssid, senha)
```

Para listar arquivos... execute File.list ('.')

Para determinar qual versão, execute a versão

Veja também:

- Github: [cesanta / smart.js](#)
- [Página inicial do Smart.js](#)
- [Smart.js Developer Center](#)

### Smart.js GPIO

Smart.js tem alguns recursos sofisticados de GPIO. Antes de ler ou escrever de um pin, devemos primeiro declarar o modo de acesso. Podemos fazer isso com GPIO.setmode (pin, modo, pullup) . O parâmetro pin é o pino que estamos definindo, mode é o modo I / O

que é codificado como:

- 0 - entrada e saída

Página 262

## Página 263

- 1 - Entrada
- 2 - Saída
- 3 - Interrupções

O parâmetro pullup define quaisquer características de pullup:

- 0 - flutuante
- 1 - pull-up
- 2 - pull-down

Uma vez que o modo tenha sido definido, podemos ler de um pino via GPIO.read (pin) ou escrever em um pin com GPIO.write (pin, value) .

Se quisermos usar interrupções, podemos registrar uma função a ser chamada quando um valor de um pino alterar. Isso é definido com GPIO.setisr (pin, type, function) . O parâmetro de tipo é uma codificação do tipo de mudança de sinal que acionará a interrupção.

- 0 - Interrupções desabilitadas
- 1 - Borda positiva
- 2 - Borda negativa
- 3 - Qualquer borda
- 4 - Baixo
- 5 - alto
- 6 - Botão

A função chamada possui a função de assinatura (pino, nível) .

Relacionado ao GPIO está a noção de PWM. Podemos definir um PWM usando PWM.set (pin, período, dever) . Tanto o período quanto a taxa são fornecidos em microssegundos.

### Configurando um servidor HTTP

Smart.js fornece uma implementação de um servidor Http. Uma classe pré-fornecida chamada "Http" está disponível. Primeiro, criamos uma instância de um servidor Http usando o método createServer () .

```
var httpServ = Http.createServer (função (req, res) {});
```

A criação de um servidor não o inicia imediatamente ouvindo, em vez de ouvir () deve ser chamado o método no servidor Http.

```
httpServ.listen (80);
```

Página 263

**Página 264**

Quando um navegador se conecta ao servidor, a função será chamada com parâmetros de um objeto de solicitação e resposta.

O objeto de solicitação contém as seguintes propriedades:

- url - A parte relativa do URL da solicitação

O objeto de resposta contém os seguintes métodos:

- writeHead (status, cabeçalhos)
- escrever (dados)
- fim()

**Depurando**

Ao executar programas JavaScript Smart.js, podemos desejar fazer alguma depuração. Nós pode realizar o registro usando print () ou Debug.print () .

Do ponto de vista da memória e de outros recursos, o comando GC.stat () retorna um objeto que contém detalhes sobre os recursos disponíveis. Isso inclui:

- own\_max
- possuído
- astsize
- funcncell
- sem função
- propncell
- propnfree
- objncell
- objfree
- struse
- strres
- jsfree
- jssize
- sysfree
- 

**Página 265****Espruino**

Espruino é uma implementação de código aberto de JavaScript que roda em uma variedade de plataformas incluindo o ESP8266. A página da web para suporte ESP8266 pode ser encontrada aqui:

<http://www.espruino.com/EspruinoESP8266>

Uma compilação do Espruino para o ESP8266 está disponível como parte do conjunto mestre de compilações disponível aqui:

<http://www.espruino.com/Download>

Aqui está uma receita para baixar e instalar uma cópia do Espruino em um ESP8266 usando o Linux como base.

```
$ mkdir espruino
$ cd espruino
$ wget http://www.espruino.com/files/espruino_1v87.zip
$ unzip espruino_1v87.zip
$ cd espruino_1v87_esp8266
$ esptool.py --port /dev/ttyUSB0 --baud 115200 write_flash \
--flash_freq 80m --flash_mode qio --flash_size 32m \
0x0000 "boot_v1.4 (b1).bin" \
0x1000 espruino_esp8266_user1.bin \
0x37E000 blank.bin
```

Quando fiz isso, o resultado foi o seguinte:

```
esptool.py v1.2-dev
Conectando ...
Executando o esboço do pisca-pisca Cesanta ...
Parâmetros de flash definidos para 0x004f
Escrevendo 4096 @ 0x0 ... 4096 (100%)
Escreveu 4.096 bytes em 0x0 em 0,4 segundos (88,7 kbit/s) ...
Escrevendo 466944 @ 0x1000 ... 466944 (100%)
Escreveu 466944 bytes em 0x1000 em 41,0 segundos (91,0 kbit/s) ...
Escrevendo 4096 @ 0x37e000 ... 4096 (100%)
Escreveu 4.096 bytes em 0x37e000 em 0,4 segundos (88,6 kbit/s) ...
Deixando...
```

Se esptool.py não estiver instalado, ele pode ser adicionado com:

```
$ sudo apt-get install python-pip
$ pip install esptool
```

#### Edição e implantação de código

O Espruino tem um editor de código elegante e uma ferramenta de implantação baseada no Chrome. Isto nos permite escrever nosso JavaScript e carregá-lo no tempo de execução do Espruino. Nós também podemos usar o método dump() para perguntar ao Espruino o que ele instalou como um programa. Isso pode ser copiado de volta para um editor.

---

## Página 266

O editor da web Espruino não está apenas disponível para execução no Chrome, ele está disponível como um aplicativo que pode ser instalado localmente. Supondo que você tenha uma cópia local do Node.js instalado, podemos executar:

```
$ sudo npm install -g espruino-web-ide
$ espruino-web-ide
```

Isso lançará uma cópia local das ferramentas de desenvolvimento.

#### Trabalhando com variáveis

Um objeto chamado "global" pode ser usado para listar os dados globais no escopo.

#### Inicializando Espruino

Quando o Espruino for inicializado, ele continuará de onde estava no código quando "save()" estava

executado. O efeito de executar "save ()" é fazer com que o estado do programa seja gravado no flash e carregado na próxima inicialização. Se precisarmos realizar a inicialização em boot, usamos o manipulador de eventos init E.on () . Por exemplo:

```
E.on ("init", function () {
    // Codifique aqui ...
});
```

Um método chamado " reset () " irá redefinir o estado do Espruino sem realizar um reinicialização do hardware.

### Acesso wi-fi

Um módulo denominado " Wifi " contém a maioria das funções de acesso sem fio. Para usar isso nós trazer o módulo com:

```
var wi-fi = requer ("Wifi");
```

Para começar a escutar como um ponto de acesso, podemos usar o método startAP () . Por exemplo:

```
var wi-fi = requer ("Wifi");
wifi.startAP ("ESP8266", {
    "authMode": "aberto"
}, função (err) {
    console.log ("AP agora iniciado:" + err);
});
```

Podemos interrogar o status do Ponto de Acesso usando getAPDetails () . Por exemplo:

```
wifi.getAPDetails (function (data) {
    // processar dados
});
```

Página 266

## Página 267

A estrutura retornada pode ser semelhante a:

```
{
    "status": "ativado",
    "authMode": "abrir",
    "oculto": falso, "maxConn": 4,
    "ssid": "ESP8266",
    "senha": "",
    "savedSsid": null,
    "estações": [
        {
            "ip": "192.168.4.2",
            "mac": "00: 36: 76: 21: 97: a3"
        }
    ]
}
```

Observe que "estações" é uma matriz de estações conectadas ao ponto de acesso.

Se quisermos saber o endereço IP do ponto de acesso, podemos chamar wifi.getAPIP () . Para exemplo:

```
wifi.getAPIP (function (data) {
    console.log (dados);
});
```

Um exemplo do que pode ser retornado seria:

```
{
    "ip": "192.168.4.1",
    "máscara de rede": "255.255.255.0",
    "gw": "192.168.4.1",
    "mac": "1a: fe: 34: f5: 2e: ec"
```

```
}
```

A alternativa a ser um ponto de acesso é ser uma estação. Se escolhermos essa opção em seguida, podemos nos conectar a uma rede WiFi existente como participante da estação. Nós fazemos isso chamando wifi.connect(). Isso leva a identidade da rede para se conectar, opções e uma função de retorno de chamada. Por exemplo:

```
wifi.connect("mySsid", {
    senha: "myPassword",
}, função (err) {
    console.log ("Conexão concluída ... O erro é:" + err);
});
```

Observe que as opções e as funções de retorno de chamada são opcionais.

### **Escrevendo aplicativos de soquete de rede usando Espruino**

A biblioteca de rede fornecida pelo ambiente Espruino pode ser acessada usando:

```
var net = require ("net")
```

Página 267

## **Página 268**

Isso retorna um objeto de rede que expõe as operações necessárias para interagir com o rede. Os métodos expostos por este objeto incluem:

- createServer - Cria uma instância de servidor
- conectar - conectar a um parceiro

```
net.connect ("http://1.2.3.4:80", função (conexão) {
    // Conexão é um objeto de conexão com um parceiro.
});
```

O objeto de conexão (que é chamado de "Socket" pelo Espruino) é sempre uma referência para uma conexão específica entre o ESP8266 e um parceiro na rede. O objeto de conexão tem os seguintes métodos:

- disponível - O número de bytes disponíveis para leitura. Isso sempre retornará 0 se um ouvinte de dados foi registrado.
- ler - Lê algum número de bytes.
- escrever - Grava algum número de bytes.
- fim - encerra a conexão.
- on ("dados") - Registra um retorno de chamada a ser invocado quando os dados se tornam disponível. Escrevendo aplicativos HTTP de rede usando Espruino

### Escrevendo um cliente REST usando Espruino

Um cliente REST é aquele que envia solicitações REST que são basicamente solicitações HTTP. Em um nível geral, o exemplo a seguir ilustra como conseguir isso:

```
var http = requer ("http");
var getRequest = http.get ({
    host: "192.168.1.9",
    porta: 1817,
    caminho: "/ sendmessage? mensagem = 1"
}, função (resposta) {
    // Trate a resposta aqui
});
});
```

Existem dois parâmetros principais para o método `get()`. O primeiro é um JavaScript de opções objeto que configura os detalhes da solicitação REST. Este objeto inclui:

- host - O endereço IP do destino da solicitação REST
- porta - O número da porta para a qual a solicitação será enviada
- caminho - a parte relativa do URL
- método - desconhecido

Página 268

## Página 269

- cabeçalhos - objeto JavaScript que representa os cabeçalhos a serem enviados.

O segundo parâmetro é uma função de retorno de chamada que será invocada quando a resposta a uma solicitação REST é recebida. A função de retorno de chamada tem a assinatura:

`retorno de chamada (resposta)`

em que `resposta` é uma instância do objeto `Espruino HTTPCRS`. Este objeto tem os seguintes métodos:

- disponível - Retorna o número de bytes disponíveis para leitura.
- evento fechado - Chamado quando a conexão é fechada.
- dados do evento - Chamado quando há dados disponíveis.
- cachimbo - Faça algo com um cachimbo.
- ler - ler os dados disponíveis.

Além disso, o objeto de resposta tem uma propriedade chamada `cabeçalhos` que são os HTTP cabeçalhos de resposta retornados do servidor. Isso incluirá as propriedades usuais que podem incluir:

- Tipo de conteúdo
- Data
- Conexão
- Comprimento do conteúdo

O retorno da chamada de função `http.get()` é um objeto do tipo `HTTPCRQ`.

Escrevendo um servidor web usando o Espruino

Também podemos configurar nosso ambiente para ser um servidor da Web que pode processar dados de entrada. Solicitações de cliente HTTP. Para fazer isso, precisamos acessar a classe `http` usando

`requer ("http")`. Esta classe possui um método para criar uma instância de um servidor web. O método possui a assinatura:

`createServer (handlerFunction)`

A função de retorno de chamada do manipulador é invocada quando chega cada novo pedido do cliente. Na maioria dos casos, o cliente será um navegador, mas poderia facilmente ser outro aplicativo fazendo solicitação REST. O manipulador de retorno de chamada usa dois parâmetros como entrada. O primeiro é um objeto que representa a solicitação (`httpSRq`) e o segundo é um objeto representando a resposta (`httpSRs`)

---

**Página 270**

O método `createServer` retorna uma instância de um objeto de servidor HTTP (`httpSrv`). Nós pode então iniciar aquele servidor HTTP escutando com uma chamada para `escutar()` que pega a porta número em que o servidor deve escutar.

Por exemplo:

```
var http = requer ("http");
var httpServer = http.createServer (função (solicitação, resposta) { ... });
httpServer.listen (80);
```

Assim que o servidor começar a escutar, as solicitações HTTP de entrada farão com que o manipulador função a ser invocada passando em um objeto que representa a solicitação e um objeto que pode ser usado para formular a resposta.

O objeto de solicitação passado para a função de retorno de chamada tem as seguintes propriedades:

- `url` - O caminho local da solicitação de entrada.
- `método` - O método HTTP da solicitação de entrada.
- `cabeçalhos` - os cabeçalhos contidos nos dados.

Uma vez que o parâmetro `url` pode ser composto de opções de consulta, pode ser bom ser capaz de extraia aqueles. Aqui está um fragmento que fará exatamente isso:

```
var partsOfUrl = request.url.split ("?");
if (partsOfUrl.length > 1) {
    var options = partsOfUrl [1] .split ('&');
    var optionsObj = {};
    para (var i = 0; i <options.length; i++) {
        var splitEquals = options [i] .split ('=');
        optionsObj [splitEquals [0]] = splitEquals [1];
    }
    print ("Final obj:" + JSON.stringify (optionsObj));
}
```

Se a solicitação de entrada for um comando HTTP POST ou PUT, podemos ter dados de carga útil enviados como parte da solicitação do cliente. Nós acessamos esses dados por registrar um retorno de chamada de evento de dados no objeto de solicitação:

```
request.on ("data", function (receivedData) {
    // Processar dados
});
```

Vamos fazer uma pausa aqui por um momento para considerar o que está acontecendo. Os dados estão chegando de o chamador como um soquete TCP de fluxo. Isso significa que os dados chegarão em ordem, mas não necessariamente de uma vez. Se o cliente deseja enviar 100 bytes de dados de carga útil no solicitação, podemos receber todos os 100 bytes como um único pedaço ou podemos tão facilmente receba 100 retornos de chamada de 1 byte cada. Como tal, é nossa responsabilidade garantir que recebemos todos os dados de que precisamos antes do processamento. Se seguirmos esta noção, iremos também perceber que os métodos `request.available()` e `request.read()` devem ser compreendido corretamente. Chamar `request.available()` nos diz quais dados foram recebidos

---

**Página 271**

até agora... e não indica quantos dados reais podem ser eventualmente recebidos. Da mesma forma, o método `request.read()` retorna dados que foram recebidos, mas não bloco esperando que novos / adicionais dados cheguem.

Um evento é disponibilizado para determinar quando o cliente enviou todos os dados necessários. Isso é feito por meio do mecanismo `request.on("close", function ()...)`. Podemos registrar um manipulador de retorno de chamada próximo para ser informado quando o cliente fecha a conexão. No neste ponto, podemos ler todos os dados através de `available()` e `read()` porque sabemos não haverá mais dados chegando.

Para retornar uma resposta, podemos invocar o método `writeHead(statusCode, headers)` para definir um código de status e cabeçalhos para o retorno. Para escrever conteúdo na resposta, podemos usar:

```
response.write(dados);
```

Para testar, precisamos enviar uma solicitação HTTP. Supondo que não estejamos testando com um navegador, podemos usar:

```
wget http://<endereço IP> --quiet --output-document = -
```

A sinalização `--quiet` desliga a conversa do aplicativo enquanto `--output-document` grava o dados para `stdout`.

Para enviar uma solicitação contendo dados, podemos usar:

```
wget http://<endereço IP> --post-data "<alguns dados>" --quiet --output-document = -
```

### Trabalhando com GPIO

O Espruino fornece suporte GPIO por meio de vários métodos globais. Esses incluem `"pinMode()` " para definir o modo de um pino, `"getPinMode()` " para obter o modo de um pino, `"digitalWrite()` " para definir o nível lógico, `"digitalRead()` " para obter o nível lógico, `"digitalPulse()` " para pulsar um nível lógico.

### Trabalhando com I2C e JavaScript

O mecanismo I2C do software está disponível na classe `I2C`, que é embutida. Usar, podemos alavancar uma instância de um objeto `I2C`. Espruino pré-cria um chamado `I2C1`. A partir daí, podemos usar o método `setup()` nele. O método `setup()` leva um objeto como entrada que tem propriedades de:

- `scl` - O pino a ser usado para um relógio (o padrão é 14)
- `sda` - O pino a ser usado para SDA (o padrão é 2)
- `taxa de bits` - a taxa de bits de comunicação (o padrão é 50000)

Para escrever no dispositivo I2C, podemos invocar o método `writeTo()`. Este tem a assinatura:

Página 271

---

## Página 272

```
writeTo(endereço, dados, ...)
```

Onde `endereço` é o endereço do dispositivo escravo I2C e `dados` são os dados a serem transmitidos.

Para ler os dados, podemos usar o método `readFrom()`. Este tem a assinatura:

```
readFrom(endereço, quantidade)
```

Onde `endereço` é o endereço do dispositivo escravo I2C e `quantidade` é o número de bytes para ler. O resultado é uma matriz de bytes.

As constantes de conveniência estão disponíveis chamadas "HIGH" e "LOW".

Veja também:

- [Trabalhando com I2C](#)

### Depurando JavaScript

Existem várias maneiras de depurar o código JavaScript.

A primeira é por meio da instrução dump (). Isso registrará o estado atual do interpretador.

A instrução trace () pode ser usada para despejar as variáveis, incluindo seus tipos. Pode pegue um nome de variável.

A variável global é um qualificador de escopo. Usando global ["\xFF"] irá acessar o Espruino variáveis "ocultas".

### Editando JavaScript

Pessoalmente, prefiro usar o ambiente de programação Eclipse para todo o meu trabalho. Nós pode instalar as ferramentas de desenvolvimento de JavaScript para nos fornecer um bom editor de JavaScript. Esta pode ser instalado por meio dos mecanismos normais de instalação de plug-ins do Eclipse. Simplesmente procure por JavaScript nos componentes instaláveis. Uma vez instalado e editado o script, você obtém todos os tipos de suporte à linguagem JavaScript, incluindo assistência de entrada e um esboço do programa:

Página 272

---

Página 273

### Bibliotecas Espruino ESP8266

Há uma biblioteca específica ESP8266 que pode ser acessada a partir da instrução require

que se parece com:

```
var esp8266 = requer ("ESP8266");
```

Existem vários métodos expostos no objeto retornado, incluindo:

- crc32 - Cria um CRC de 32 bits.
- deepSleep - Faça o ESP8266 entrar no modo "deep sleep". Efetivamente um cronometrado reinício.
- dumpSocketInfo - Depure as informações do soquete gravando-as no log.
- getFreeFlash - Retorna a quantidade de armazenamento flash livre. Descontinuada.
- getResetInfo - Recupere o motivo da última reinicialização / reinicialização.
- getState - recupera o estado do dispositivo
  - sdkVersion - versão do SDK
  - cpuFrequency - MHZ da velocidade da CPU
  - freeHeap - Quantidade de armazenamento de heap livre
  - maxCon - Número máximo de conexões
  - flashMap - Como o flash é mapeado
  - flashKB - tamanho de flash configurado
  - flashChip - Fabricante de chip flash

Página 273

## Página 274

aqui está um exemplo de saída:

```
{
  "sdkVersion": "1.5.0",
  "cpuFrequency": 160, "freeHeap": 10096, "maxCon": 10,
  "flashMap": "4 MB: 512/512",
  "flashKB": 4096,
  "flashChip": "0xe0 0x4016"
}
```

- logDebug - Habilite ou desabilite o registro de depuração.
- neopixelWrite - Grava em uma string de NeoPixels.
- ping - Ping o endereço IP fornecido.
- printLog - Imprime o log de depuração no console.
- readLog
- reiniciar - reiniciar o dispositivo.
- setCPUFreq - Defina a frequência da CPU. Descontinuada.
- setLog - Define o modo de registro
  - 0 - desligado
  - 1 - na memória
  - 2 - na memória e UART0
  - 3 - na memória e UART1

A linguagem JavaScript fornecida pelo Espruino é abordada em detalhes pelo Espruino documentação. No entanto, aqui estão alguns dos elementos essenciais que considero extremamente útil.

Veja também:

- Referência do software Espruino

Executando código em intervalos

Podemos definir um temporizador que irá disparar uma vez (`setTimeout()`) ou periodicamente (`setInterval()`) e chama uma função.

A sintaxe para isso é:

```
setTimeout (função, atraso, [args,...])
setInterval (função, período, [args,...])
```

Página 274

## Página 275

Onde o atraso e o período são tempos medidos em milissegundos. Argumentos opcionais podem também ser fornecidos, os quais são passados para a função. Ambas as funções retornam um id valor que pode ser usado para cancelar a solicitação antes que ela aconteça. A função para fazer isso é chamado de `clearInterval()`.

```
clearInterval (id)
```

Também podemos alterar o intervalo em um retorno de chamada periódico com o `changeInterval()` função.

```
changeInterval (id, newPeriod)
```

Trabalhando com GPIO

Podemos definir um objeto do tipo Pin para representar um pino GPIO e então definir seu valor ou ler seu valor. Por exemplo, aqui está um piscar simples:

```
var ledOn = false;
var ledPin = novo Pin (4);
setInterval (function () {
    digitalWrite (ledPin, ledOn);
    ledOn =! ledOn;
}, 1000);
```

SPI

SPI é um protocolo com fio usado para conduzir componentes de interface compatíveis com SPI. Espruino tem um módulo chamado SPI que nos fornece acesso a esses recursos. Primeiro criamos um Porta SPI usando:

```
var mySPI = novo SPI ();
```

Em seguida, podemos configurar essa porta usando a função `setup()`. O parâmetro para configurar é um objeto que contém:

- sek - O pino a ser usado para o relógio.
- miso - O pino a ser usado para entrada mestre / saída escrava.
- mosi - O pino a ser usado para saída mestre / entrada escrava.
- baud (opcional) - o padrão é 100.000.
- modo (opcional) - o padrão é 0.
- pedido (opcional) - o padrão é "msb".

Finalmente, podemos chamar a função write () para escrever dados. Alternativamente, podemos chamar send () .

## Página 276

Aqui está um exemplo. O MAX7219 é um pequeno CI poderoso que é capaz de conduzir um 8x8 matriz de LEDs. Por ser um dispositivo SPI, ele usa três sinais SPI:

- CS - Baixo para selecionar MAX7219 para comunicação SPI.
- MOSI - A linha de dados sobre a qual os dados seriais fluirão.
- CLK - A linha do relógio que controla a recepção de novos bits de dados.

Se olharmos para um ESP8266, podemos escolher mapeá-los para o seguinte ESP8266 alfinetes:

Função	Alfinete	NodeMCU	Cor
CS	GPIO 12 D6		
MOSI	GPIO 13 D7		
CLK	GPIO 14 D5		

### Principais diferenças do JavaScript

Embora o Espruino seja uma excelente implementação, ele tem algumas diferenças de um Padrões ECMAScript. Algumas coisas são sutis e improváveis de serem encontradas enquanto outros são maiores. aqui estão alguns exemplos:

- Funções de chamada antes de sua declaração não são suportadas.
- Declarar uma variável como const não significa que seja assim.

### Edifício Espruino

Para construir o Espruino a partir da árvore de origem:

```
$ git clone https://github.com/espruino/Espruino.git
$ cd Espruino
$ export ESP8266_BOARD=1
$ export FLASH_4MB=1
$ export ESP8266_SDK_ROOT=/esp8266/sdk/ESP8266_NONOS_SDK
$ export PATH=$PATH:/pot/xtensa-lx106-elf/bin
$ export ESPHOSTNAME=espruino:88
```

Crie o diretório SDK:

```
$ wget
http://espressif.com/sites/default/files/sdks/esp8266\_nonos\_sdk\_v2.0.0\_16\_08\_10.zip
$ wget
http://espressif.com/sites/default/files/sdks/esp8266\_nonos\_sdk\_v2.0.0\_patch\_16\_08\_09.zip
fecho eclair
$ mkdir src
$ unzip esp8266_nonos_sdk_v2.0.0_16_08_10.zip -d sdk
$ unzip esp8266_nonos_sdk_v2.0.0_patch_16_08_09.zip -d sdk /ESP8266_NONOS_SDK/lib
```

## Programação com Lua

Lua é uma linguagem de script poderosa que está disponível em ambientes ESP8266. A implementação mais popular de Lua para o ESP8266 é conhecida como NodeMCU Lua firmware e está disponível em seu repositório github. Algumas pessoas trocam a frase NodeMCU para o próprio firmware Lua, portanto, certifique-se de entender o contexto envolvido.

Builds do firmware podem ser baixados diretamente, assim como a fonte.

Assim que tiver uma cópia do firmware, você pode fazer o flash usando suas ferramentas favoritas de flash.

Não descreveremos a linguagem Lua em si neste livro. Existem livros excelentes já escrito em Lua e também referências e tutoriais podem ser encontrados na Internet.

Em vez disso, vamos dar uma olhada nas especificidades da execução de Lua em um ESP8266.

Supondo que você tenha atualizado um ESP8266 com Lua, você precisará conectar um serial terminal para interagir com ele. A taxa de transmissão serial deve ser 9600.

Veja também:

- Github: [nodemcu / nodemcu-firmware](#)
- [Wiki do firmware NodeMCU](#)
- [Lua 5.1 Manual de Referência](#)
- [lua.org](#)
- [Fórum NodeMCU ua em ESP8266.com](#)
- [nodemcu-unofficial-faq](#)

## ESPlorer IDE

O ESPlorer IDE é um ambiente de desenvolvimento para a construção de aplicativos Lua para o ESP8266.

Veja também:

- [Página inicial do Esplorer](#)
- e-book: [Introdução ao ESPlorer IDE](#)
- GitHub: [4refront / ESPlorer](#)

## GPIO com Lua

Lua tem o conceito de 13 pinos lógicos identificados por 0-12. Esses pinos são mapeados para o GPIO pinos de um ESP8266 da seguinte forma:

Pino lua número	ESP8266 Pin NodeMCU devKit	Pino lua número	ESP8266 Pin NodeMCU devKit
0	GPIO16	D0	7
1	GPIO5	D1	8
2	GPIO4	D2	9
3	GPIO0	D3	10
4	GPIO2	D4	11

Página 277

5	GPIO14	D5	12	GPIO10	SD3
6	GPIO12	D6			

Os pinos GPIO conhecidos como GPIO6, GPIO7 e GPIO8 no ESP8266 não são expostos.

Antes de ler ou escrever de um pino GPIO, precisamos informar Lua sobre o modo desse alfinete. Nossas escolhas são entrada, saída ou interrupção. Para entrada, também podemos declarar se a entrada é pull-up ou flutuante.

A sintaxe da declaração é:

```
gpio.mode(pin, modo, pullup)
```

Depois de definir o modo, se for uma entrada, podemos chamar `gpio.read()` para ler o valor do pino e `gpio.write()` para gravar um valor no pino.

Se quisermos ser disparados por uma interrupção no pino (entrada), podemos usar `gpio.trig()`.

## WiFi com Lua

### Networking com Lua

## Programação com Basic

Veja também:

- [ESP8266 Básico](#)

## Integração com aplicativos da web

### Serviços REST

A noção de computação distribuída remonta a muitas décadas. A ideia daquele o computador poderia executar um serviço em nome de outro é um conceito clássico. O pensando é que o trabalho pode ser distribuído entre os sistemas, os dados podem ser centralizados ou sistemas dedicados podem desempenhar funções especializadas. Ao longo dos anos, muitas formas de computação distribuída foi tentada. Isso inclui servidores de soquete, procedimento remoto chamadas (RPC), Arquitetura de Rede de Sistemas (SNA), Ambiente de Computação Distribuída (DCE), Web Services e outros.

Hoje (2015), o atual encarregado de protocolos e tecnologia de computação distribuída é REST. REST é um protocolo simples que aproveita o transporte de hipertexto existente Protocolo (HTTP) usado como transporte entre navegadores e servidores da web. Esta protocolo foi construído para permitir que um navegador solicite dados de um sistema de arquivos remoto hospedado por um servidor web. Ele fornece "comandos" HTTP que incluem GET, POST, PUT e outras. A noção por trás do REST é mais um acidente do que um design. REST reusa HTTP como um canal de comunicação de um cliente para um servidor onde um cliente faz uma solicitação REST e o servidor oferece um serviço REST. Da rede

Página 278

## Página 279

perspectiva, ele "se parece" com uma interação navegador / servidor da Web, mas ambas as extremidades escolhem chegar a acordo sobre a formação e interpretação da comunicação.

Quando adicionamos um ESP8266 à mistura, nosso desejo é duplo. Queremos o ESP8266 para poder ser um cliente de provedores de serviços REST externos e queremos o ESP8266 para ser o alvo dos clientes que fazem solicitações REST. Da perspectiva do parceiro, não deve estar ciente de que está interagindo com o ESP8266 em comparação com qualquer outro dispositivo de computação.

### Protocolo REST

O protocolo REST é baseado em HTTP.

Veja também:

- [RFC7230 - HTTP / 1.1 - Sintaxe e roteamento da mensagem](#)
- [HTTP: O protocolo que todo desenvolvedor da Web deve conhecer - Parte 1](#)

### **ESP8266 como um cliente REST**

Para que o ESP8266 seja um cliente REST, ele deve construir e transmitir solicitações HTTP para o provedor de serviço. Isso incluirá a construção de cabeçalhos HTTP, transmitindo os dados em um forma esperada pelo provedor (por exemplo, JSON, XML ou outra representação textual) e lidar com a resposta do provedor, que pode incluir a interpretação do recebido carga útil.

Para transmitir uma solicitação REST é composta de duas partes. Primeiro, ele abre uma conexão TCP para o parceiro e, em seguida, transmite os dados compatíveis com HTTP por essa conexão. O a primeira parte é fácil, a segunda parte é mais um desafio. Poderíamos ler e entender a especificação HTTP e construir a solicitação parte por parte, mas isso teria que ser feito para cada projeto que deseja usar a tecnologia do cliente REST. O que seria melhor é se nós tinha uma biblioteca que "sabe" como fazer solicitações REST bem formadas e nós simplesmente alavancou suas funções existentes.

Fazer uma solicitação REST usando Mongoose

Usando as APIs do Mongoose, podemos facilmente enviar uma solicitação REST e trabalhar com o resposta. A história de alto nível é inicializar o Mongoose com `mg_mngr_init()`, solicitar um conexão com o provedor de serviços REST com `mg_connect()`, associe a conexão como sendo orientado para HTTP e, em seguida, comece a processar eventos. O primeiro evento a retornar ser um evento `MG_EV_CONNECT` indicando que agora estamos conectados à rede. De lá podemos usar `mg_printf()` para enviar a solicitação REST. Quando o parceiro REST responde, obteremos um evento `MG_EV_HTTP_REPLY` e concluímos nossa solicitação / resposta emparelhamento.

Página 279

---

### **Página 280**

### **ESP8266 como um provedor de serviços REST**

Para um ESP8266 ser um provedor de serviços REST, basicamente significa que ele deve reproduzir o função de um servidor da Web e responder às solicitações do servidor da Web. No entanto, ao contrário de um simples Servidor da Web que simplesmente recupera e envia o conteúdo do arquivo como uma função do caminho em a URL, é provável que o provedor de serviços REST execute alguns cálculos quando chega uma solicitação de cliente HTTP. Por exemplo, se anexarmos um sensor de temperatura a os GPIOs do ESP8266, quando uma solicitação REST chega, o ESP8266 pode ler o valor da temperatura atual e enviar o resultado codificado de volta como uma resposta ao solicitação.

Ser um servidor da Web significa basicamente ouvir em uma porta TCP e quando as conexões chegar, interpretando os dados recebidos como protocolo HTTP. Seria muito trabalhoso em um projeto por projeto, mas felizmente há uma série de bibliotecas pré-escritas que realizar essa tarefa para nós e tudo o que precisamos nos preocupar é o exame de qualquer parâmetros passados com a solicitação e realizando a lógica que desejamos realizada quando sempre que um novo pedido é recebido.

Uma biblioteca como ESP8266WebServer seria perfeita para essa tarefa.

Veja também:

- [ESP8266WebServer](#)

**Tasker**

Tasker é um aplicativo Android que automatiza e cria scripts de tarefas a serem executadas em um Dispositivo Android. Usando Tasker, podemos criar uma tarefa que é definida como uma sequência de comandos e ações a serem executados. Em seguida, podemos criar um perfil que mapeia um evento, que quando detectado, executa uma tarefa. Embora seja útil, como isso está relacionado a um ESP8266? Imagine que o evento que ocorre é um ESP8266 enviando um mensagem para o seu telefone. Com essa noção, um ESP8266 pode, efetivamente, acionar qualquer coisa que alguém pode fazer com esse telefone. Por exemplo, pode fazer um telefone ligar, enviar uma mensagem SMS ou tirar uma fotografia.

Veja também:

- [Página inicial da Tasker](#)
- YouTube: [Tasker 101 Tutoriais](#)

**AutoRemote**

Na sequência de nossa discussão sobre Tasker acima, agora temos uma admissão. Isto parece que Tasker **não** tem a capacidade de ouvir TCP / IP de entrada com base eventos e mensagens. No entanto, como o Tasker é extensível e os desenvolvedores podem

Página 280

**Página 281**

escrever plug-ins para ele, Tasker pode ser aumentado. Um tal aumento é o Plug-in AutoRemote. Usando esse plugin, uma mensagem TCP / IP pode ser enviada e recebido pelo AutoRemote que pode então atuar como uma fonte de eventos para Tasker.

Com o AutoRemote configurado como um plugin Tasker, podemos configurá-lo para ouvir HTTP solicitações de. Isso faz com que o AutoRemote escute na porta TCP número 1817 . Os dados são ouvir é uma solicitação HTTP. Por exemplo:

`http://<phone ip>: 1817 / sendmessage? message = 1`

Com o Tasker e o AutoRemote instalados, ele ainda não estará ouvindo o WiFi de entrada mensagens em um ambiente WiFi local, a menos que estejamos conectados à Internet. Nós devemos execute uma tarefa Tasker chamada "AutoRemote WiFi".

Por exemplo, no Tasker:

1. Crie um novo perfil acionado por Evento → Sistema → Inicialização do dispositivo
2. Crie uma nova tarefa associada ao perfil
3. Adicione uma ação de Plugin → AutoRemote → Wifi
4. Na configuração da ação, marque "Serviço Wifi"

O que isso fará é iniciar o serviço Wifi sempre que o dispositivo (Android) inicializar.

Infelizmente, o AutoRemote tem uma série desvantagem. Não permite que Tasker envie um resposta de volta na solicitação REST original que pode conter dados que podem ser usados. Por exemplo, se quisermos usar o AutoRemote para enviar uma solicitação que retornou o atual Localização GPS, isso simplesmente não é possível.

Quando uma solicitação AutoRemote chega, ele define uma série de variáveis dentro do Tasker ambiente que pode ser usado como parâmetro para tarefas Tasker. Esses incluem:

- % armassage
- % arpar ()
- % arcomm
- % artime

- % arfiles
- % arsenderbtmac
- % arsenderid
- % arsenderlocalip
- % arsendername
- % arsenderpublicip

Página 281

---

## Página 282

- % arsendertype
- % arvia
  - wi-fi

Veja também:

- [Página inicial do AutoRemote](#)

### DuckDNS

Prevejo que na maioria das casas há um ponto de acesso WiFi que pode ser diretamente ou através de um modem, conecta-se à Internet. Uma vez que o ponto de acesso WiFi oferece um local rede à qual o ESP8266 pode se conectar, agora vemos que o ESP8266 pode alcançar o mundo exterior através do ponto de acesso. No entanto, e quanto ao inverso? E se nós quer um cliente na Internet para chegar ao nosso ESP8266. Como poderíamos conseguir isso?

Se olharmos o diagrama acima (todos os endereços IP compostos), vemos que o ESP8266 conhece seu próprio endereço IP como 192.168.1.2. No entanto, isso não pode ser "compartilhado" com o Internet, pois é um endereço local e não um endereço IP global. O que precisaria ser compartilhado é o endereço IP do ponto de acesso conforme visto na Internet.

Uma maneira de conseguir isso é usando um provedor de serviços como o DuckDNS. Este serviço gratuito permite que você registre um nome. Seu dispositivo (geralmente um PC) periodicamente envia uma solicitação para o site da DuckDNS dizendo "Olá... estou aqui!". O retorno endereço dessa solicitação é sempre o endereço IP do seu ponto de acesso conectado ao Internet e, portanto, DuckDNS aprende seu endereço externo. Mais tarde, alguém (talvez um terceiro) pode perguntar "Qual é o endereço IP" do nome que você registrou e que endereço é disponibilizado. Essencialmente, DuckDNS atua como um corretor em tempo real de lógica nomes para endereços IP.

**Página 283**

Se você está preocupado que "alguma pessoa assustadora" possa descobrir o endereço IP de seu acesso ponto ... então não use DuckDNS. No entanto, para a maioria de nós, nosso modem / roteador / ponto de acesso impede que o tráfego de entrada nos alcance e, essencialmente, bloqueia tudo o que não queremos. Mas espere ... isso também não bloqueará solicitações para o ESP8266? A resposta é "sim, vai", por isso você deve definir o encaminhamento de porta. Encaminhamento de porta é uma função do seu modem / roteador / ponto de acesso que diz que quando um pedido chega para um determinado local de porta, encaminhe-o automaticamente para um endereço IP em sua rede local ... por exemplo, o endereço de rede de seu ESP8266.

<https://www.duckdns.org/update?domains=XXX&token=XXX&ip=>

## Aplicativos móveis

### Blynk

Veja também:

- [Página inicial do Blynk](#)

## Snippets de amostra

Há momentos em que tudo o que precisamos é de um trecho de código que possamos copiar para alcançar uma tarefa. Aqui, apresentamos um conjunto de trechos que podem ser usados simplesmente copiando e colando-os.

### Formando uma conexão TCP

Aqui, vemos um trecho de código que pode ser usado para fazer uma conexão TCP / IP.

```
# define REMOTE_PORT 80
# define REMOTE_IP "216.58.218.206"
struct espconn conn1;
esp_tcp tcp1;

void connectCB (void * arg) {
    os_printf ("Conectamos \n");
}

void errorCB (void * arg, sint8 err) {
    os_printf ("Ocorreu um erro:% d \n", err);
}

void makeConnection () {
    conn1.type = ESPCONN_TCP;
    conn1.state = ESPCONN_NONE;
    conn1.proto.tcp = & tcp1;
    conn1.proto.tcp-> remote_port = REMOTE_PORT;
    *((uint32 *) conn1.proto.tcp-> remote_ip) = ipaddr_addr (REMOTE_IP);
    espconn_regist_connectcb (& conn1, connectCB);
    espconn_regist_reconcb (& conn1, errorCB);
    espconn_connect (& conn1);
    os_printf ("Solicitamos uma conexão!");
}
```

**Página 284**

## Aplicativos de amostra

Ler e revisar aplicativos de amostra é uma boa prática. Isso permite que você estude o que os outros escreveram e veja se você consegue entender cada uma das declarações e o fluxo do programa como um todo.

### Amostra - Acenda um LED com base na chegada de um datagrama UDP

Neste exemplo, teremos o ESP8266 se tornando uma estação WiFi e se conectar. Será começe a escutar os datagramas de entrada e se o primeiro byte de dados recebidos for o caractere "1", acenderá um LED. Se o caractere for "0", o LED apagará.

Aqui está o código completo do aplicativo com comentários a seguir:

```
#include <ets_sys.h>
#include <osapi.h>
#include <os_type.h>
#include <gpio.h>
#include <user_interface.h>
#include <espconn.h>
#include <mem.h>
#include "driver / uart.h"

#define LED_GPIO 15

LOCAL struct espconn conn1;
LOCAL esp_udp udp1;

LOCAL void recvCB (void * arg, char * pData, unsigned short len);
LOCAL void eventCB (System_Event_t * event);
LOCAL void setupUDP ();
LOCAL void initDone ();

LOCAL void reevCB (void * arg, char * pData, unsigned short len) {
    struct espconn * pEspConn = (struct espconn *) arg;
    os_printf ("Dados recebidos !! - comprimento=%d \n", len);
    if (len == 0 || (pData [0] != '0' && pData [0] != '1')) {
        Retorna;
    }
    int v = (pData [0] == '1');
    GPIO_OUTPUT_SET (LED_GPIO, v);
} // Fim do recvCB

LOCAL void initDone () {
    wifi_set_opmode_current (STATION_MODE);
    struct station_config stationConfig;
    strncpy (stationConfig.ssid, "myssid", 32);
    strncpy (stationConfig.password, "senha", 64);
    wifi_station_set_config_current (& stationConfig);
    wifi_station_connect ();
}
```

Página 284

---

## Página 285

```
} // Fim do initDone

LOCAL void setupUDP () {
    conn1.type = ESPCONN_UDP;
    conn1.state = ESPCONN_NONE;
    udp1.local_port = 25867;
    conn1.proto.udp = & udp1;
    espconn_create (& conn1);
    espconn_regist_recvcb (& conn1, recvCB);
    os_printf ("Ouvindo dados \n");
} // Fim da configuração UDP
```

```

LOCAL void eventCB (System_Event_t * event) {
    switch (evento-> evento) {
        case EVENT_STAMODE_GOT_IP:
            os_printf ("IP:% d.% d.% d\\n", IP2STR (& event-> event_info.got_ip.ip));
            setupUDP ();
            pausa;
    }
} // Fim do eventoCB

void user_rf_pre_init (void) {
}

void user_init (void) {
    uart_init (BIT_RATE_115200, BIT_RATE_115200);

    // Definir GPIO15 como um pino GPIO
    PIN_FUNC_SELECT (PERIPH_IO_MUX_MTDO_U, FUNC_GPIO15);

    // Chame "initDone" quando o ESP8266 for inicializado
    system_init_done_cb (initDone);
    wifi_set_event_handler_cb (eventCB);
} // Fim do user_init

```

O controle começa na função `user_init()` onde configuramos a transmissão UART. Nisso exemplo, escolhemos GPIO15 como nosso pino de saída para mapear a função do pino físico denominado "MTDO\_U" para a função lógica de "GPIO15". Registraremos uma função chamado `initDone()` para ser chamado quando a inicialização do dispositivo estiver completa e nós também registrar uma função chamada `eventCB()` a ser chamada quando os eventos WiFi chegarem, indicando um Mudança de estado.

Com esses itens configurados, devolvemos o controle ao sistema operacional. Esperamos ser chamado de volta por meio de `initDone()` quando o dispositivo é totalmente lido para o trabalho. Em `initDone()` nos definimos como uma estação Wifi e nomeamos o ponto de acesso com sua senha que que desejamos usar. Finalmente, pedimos uma conexão com o ponto de acesso.

Se tudo correr bem, seremos conectados ao ponto de acesso e, em seguida, teremos um IP Morada. Ambos resultarão na geração de eventos que nos farão acordar em `eventCB()`. O único evento que estamos interessados em ver é a alocação do IP

---

## Página 286

Morada. Quando somos notificados disso, chamamos a função chamada `setupUDP()` para inicializar nosso ambiente de escuta UDP.

Em `setupUDP()`, criamos um bloco de controle struct `espconn` definido para UDP e configurado para escutar em nossa porta escolhida de 25867. Também registramos um retorno de chamada de recepção para a função `recvCB()`. Isso será chamado quando novos dados chegarem. Neste ponto, todos os nossos a configuração está concluída e temos um dispositivo conectado à rede WiFi ouvindo Porta UDP 25867 para datagramas.

Quando chega um datagrama, acordamos em `recvCB()` tendo sido passado no datagrama dados. Verificamos se realmente temos dados e se eles são bons ... se não, encerramos o retorno de chamada imediatamente.

Por fim, examinamos o primeiro caractere dos dados e, com base em seu valor, alteramos o valor de saída do GPIO. O GPIO físico é conectado a um LED e um resistor.

Se um caractere '1' for transmitido, a saída do GPIO15 será alta e o LED acenderá. Se o valor do caractere é '0', a saída do GPIO15 fica baixa e o LED se apaga.

**Amostra - Medição de distância ultrassônica**

O HC SR-04 é um sensor ultrassônico para medição de distância.

Envie no mínimo um pulso de 10us para Trig (baixo para alto para baixo). Mais tarde, Echo irá baixo / alto / baixo. O tempo que o Echo está alto é o tempo que o pulso sônico leva para atingir um back-end e salto para trás.

A velocidade do som é 340,29 m / s ( $340,29 * 39,3701$  polegadas / s). Chame esse  $V_{\text{som}}$ .

Página 286

**Página 287**

Se  $T_{\text{echo}}$  for o tempo para resposta de eco, então  $d = (T_{\text{echo}} * V_{\text{som}}) / 2$ .

Além disso, a equação para os comprimentos de  $\text{eco}$  T esperados é dada por:

$$T_{\text{echo}} = \frac{2d}{V}$$

Por exemplo:

Distância	Tempo
1cm	$2 * 0,01 / 340 = 0,058$ mseg = 59 usegs
10cm	$2 * 0,1 / 340 = 0,59$ mseg = 590 usegs
1m	$2 * 1/340 = 5,9$ mseg = 5900 usegs (5,9 mseg)

Como a resposta do eco é um sinal de 5 V, é vital reduzi-lo para 3,3 V para entrada em no ESP8266. Um divisor de tensão funcionará. Os pinos do dispositivo são:

- Vcc - A tensão de entrada é 5V.

- Trig - Pulso (baixo a alto) para acionar uma transmissão ... mínimo de 10usecs.
- Eco - pulsa de baixo para alto para baixo quando um eco é recebido. Atenção, este é um 5V resultado.
- Gnd - Ground.

Para acionar este dispositivo, precisamos utilizar dois pinos no ESP8266 que iremos logicamente chame Trig e Echo . Em meu projeto, configurei Trig como GPIO4 e Echo como GPIO5.

Nosso design para o aplicativo não incluirá nenhuma rede, mas deve ser direto para associá-lo conforme necessário. Vamos configurar um cronômetro que dispara uma vez por segundo que é a frequência com que desejamos fazer uma medição. Quando o cronômetro acordar, nós iremos pulso Trig de baixo para alto e de volta para baixo segurando alto por 10 microssegundos. Nós vamos agora registre a hora e comece a pesquisar o pino Echo, esperando que ele fique alto. Quando o fizer, vamos registrar o tempo novamente e subtrair um do outro nos dirá como

Página 287

## Página 288

o som demorou muito para voltar. A partir daí podemos calcular a distância para um objeto. Se nenhuma resposta for recebida em 20 ms, assumiremos que não houve nenhum objeto detectar. Em seguida, registraremos o resultado no console serial.

Um programa de exemplo que executa este design é mostrado a seguir:

```
# define TRIG_PIN 4
#define ECHO_PIN 5

os_timer_t myTimer;

void user_rf_pre_init (void) {
}

void timerCallback (void * pArg) {
    os_printf ("Assinale! \n");
    GPIO_OUTPUT_SET (TRIG_PIN, 1);
    os_delay_us (10);
    GPIO_OUTPUT_SET (TRIG_PIN, 0);
    uint32 val = GPIO_INPUT_GET (ECHO_PIN);
    enquanto (val == 0) {
        val = GPIO_INPUT_GET (ECHO_PIN);
    }
    uint32 startTime = system_get_time ();
    val = GPIO_INPUT_GET (ECHO_PIN);
    while (val == 1 && (system_get_time () - startTime) <(20 * 1000)) {
        val = GPIO_INPUT_GET (ECHO_PIN);
    }
    if (val == 0) {
        uint32 delta = system_get_time () - startTime;
        // Calcule a distância em cm.
        distância uint32 = 340,29 * 100 * delta / (1000 * 1000 * 2);
        os_printf ("Distância:% d \n", distância);
    } senão {
        os_printf ("Sem eco! \n");
    }
} // Fim do timerCallback

void user_init (void) {
    uart_init (BIT_RATE_115200, BIT_RATE_115200);
    // Configurar pinos ultrassônicos como GPIO
    setAsGpio (TRIG_PIN);
    setAsGpio (ECHO_PIN);
    setupBlink (15);
    // Defina o pino do gatilho como padrão baixo
    GPIO_OUTPUT_SET (TRIG_PIN, 0);
    os_timer_setfn (& myTimer, timerCallback, NULL);
    os_timer_arm (& myTimer, 1000, 5);
}
```

```
} // Fim do user_init
```

Depois que isso for escrito e testado, faremos uma segunda tentativa no quebra-cabeça, mas desta vez usando uma interrupção para acionar a resposta ao eco.

Veja também:

Página 288

## Página 289

- [GPIOs](#)

### **Amostra - Scanner WiFi**

Um scanner WiFi é um aplicativo que verifica periodicamente as redes WiFi disponíveis e os mostra ao usuário. Em nosso projeto, vamos escanear periodicamente e lembrar o conjunto de redes que encontramos. Quando realizarmos novas varreduras, verificaremos se cada uma das redes localizadas é uma rede que vimos anteriormente e, se não, liste-a ao usuário. Nós também manterá um tempo de "última visualização" para cada rede e se uma rede não foi vista por um minuto, então vamos esquecer de tal forma que se ele aparecer novamente, iremos mais uma vez listar para o usuário.

Para ilustrar nosso design, dividiremos a solução em várias partes. A primeira parte será registrar uma função de retorno de chamada que é chamada a cada 30 segundos. Este retorno de chamada irá ser responsável por solicitar uma varredura de WiFi usando `wifi_station_scan()`. Isso leva uma função de retorno de chamada que será chamada quando a varredura for concluída.

Quando a varredura for concluída, teremos uma nova lista de redes detectadas. Andaremos esta lista e para cada rede detectada, determine se já a vimos antes. Se tivermos, vamos atualizar a última vez que foi visto. Caso contrário, iremos adicioná-lo à lista de vistos anteriormente redes e registrá-lo para o usuário.

Um segundo retorno de chamada do cronômetro será executado uma vez por minuto e percorrerá a lista de vistos anteriormente redes. Se algum deles tiver mais de um minuto, nós os removeremos.

Veja também:

- [Procurando pontos de acesso](#)

### **Amostra - Trabalhando com cartões micro SD**

Um cartão micro SD é um pequeno dispositivo de armazenamento portátil que pode hospedar gigabytes de dados. Através do uso de um adaptador, um cartão micro SD pode ser aproveitado em conjunto com um ESP8266 fornecendo acesso de leitura e gravação a dados que persistem em ESP8266 reinicia.

### **Amostra - Reproduzindo áudio de um evento**

Neste exemplo, desejamos que um evento seja detectado pelo ESP8266 que, quando acontece, faz com que um arquivo de áudio seja reproduzido.

### **Amostra - Uma luz de humor mutável**

NeoPixels são LEDs acionados por uma única linha de dados de sinalização de alta velocidade. A maioria dos NeoPixels têm uma fonte de tensão +ve e terra, bem como uma linha de dados para entrada e um

Página 289

---

**Página 290**

linha de dados para saída. A saída de um NeoPixel pode ser alimentada na entrada do próximo um para produzir uma série de LEDs. Os dados de entrada para o LED são um fluxo de 24 bits de dados codificados que devem ser interpretados como 8 bits para o canal vermelho, 8 bits para o canal verde e 8 bits para o canal azul. Cada canal pode, portanto, ter um valor de luminância entre 0 e 255. Misturando os valores para cada um dos canais Juntos, você pode colorir um LED com qualquer cor de sua escolha. Depois de enviar um stream de 24 bits, se enviarmos um segundo fluxo de 24 bits rapidamente após o primeiro fluxo, o segundo fluxo é "empurrado" para o próximo LED da cadeia. Isso pode ser repetido tanto quanto desejado. Se pausarmos o envio de dados, os valores atuais serão "travados" em coloque e cada LED deles lembra seu próprio valor.

Os tempos dos sinais de dados para esses LEDs podem ser bastante complicados, mas felizmente ótimos mentes já construíram bibliotecas fantásticas para conduzi-los corretamente, então não precisamos nos preocuparmos com esses tempos de baixo nível e podemos nos concentrar em planejar projetos e propósitos interessantes para os quais os LEDs podem ser colocados. Há um número de diferentes tipos desses LEDs, sendo os mais comuns conhecidos como WS2811, WS2812 ou PL9823.

Dentro do ambiente Espruino JavaScript, um método chamado `neopixelWrite()` pode ser encontrado. Isso leva dois parâmetros. O primeiro é o pino GPIO ESP8266 que será usado como a fonte dos sinais para os LEDs. É a esse pino que os LEDs devem ser conectados. O pino usado para a saída de dados do ESP8266 para os NeoPixels deve ser definido no GPIO modo de saída. Por exemplo:

```
pinMode (pin, "saída");
```

O segundo parâmetro é uma matriz de inteiros. Os valores da matriz devem ser fornecido em grupos de 3 correspondentes aos 3 canais de vermelho, verde e azul. Para exemplo, se tivéssemos um NeoPixel conectado ao GPIO4 em um ESP8266 e quiséssemos para defini-lo como todo vermelho, podemos codificar:

```
neopixelWrite (novo Pin (4), [255, 0, 0]);
```

Se quiséssemos que o próximo pixel fosse verde enquanto o primeiro fosse vermelho, poderíamos escrever:

```
neopixelWrite (novo Pin (4), [255, 0, 0, 255, 0]);
```

Novamente, não há limite óbvio para o número de LEDs que podemos conectar.

Agora que vemos que podemos definir o brilho e a cor de um LED, vamos ver como podemos projetar algum código para fazer algo. Vamos imaginar que tivemos uma seqüência de 16 LEDs e queria torná-los da mesma cor ... podemos definir uma função como segue:

```
function colorLeds (vermelho, verde, azul) {
    var data = [];
    para (var i = 0; i < 16; i++) {
        data.push (verde);
        data.push (vermelho);
```

Página 290

---

**Página 291**

```
        data.push (azul);
    }
    esp8266.neopixelWrite (NodeMCU.D2, dados);
}
```

Se chamarmos esta função com os valores corretos de vermelho, verde e azul, ela definirá os LEDs string corretamente.

Agora, vamos dar um passo adiante. Imagine que recebemos uma solicitação REST de rede que descreveu a cor que queremos que os LEDs mostrem. Um aplicativo completo pode ser:

```
var esp = require ("ESP8266");
var NodeMCU = {
    // D0: novo pino (16),
    D1: novo pino (5),
    D2: novo pino (4),
    D3: novo pino (0),
    D4: novo pino (2),
    D5: novo pino (14),
    D6: novo pino (12),
    D7: novo pino (13),
    D8: novo pino (15),
    D9: novo pino (3),
    D10: novo pino (1)
};
```

```
pinMode (NodeMCU.D2, "saída");
```

```
function colorLeds (vermelho, verde, azul) {
```

```
    var data = [];
    para (var i = 0; i < 16; i++) {
        data.push (verde);
        data.push (vermelho);
        data.push (azul);
    }
    esp.neopixelWrite (NodeMCU.D2, dados);
}
```

```
function beServer () {
```

```
    var http = requer ("http");
    var httpServer = http.createServer (função (solicitação, resposta) {
        imprimir (solicitar);
        var partsOfUrl = request.url.split ("?");
        if (partsOfUrl.length > 1) {
            var options = partsOfUrl [1].split ('&');
            var optionsObj = {};
            para (var i = 0; i < options.length; i++) {
                var splitEquals = options [i].split ('=');
                optionsObj [splitEquals [0]] = splitEquals [1];
            }
            print ("Final obj:" + JSON.stringify (optionsObj));
            if (optionsObj.color != null) {
                var red = parseInt (optionsObj.color.substr (0,2), 16);
                var green = parseInt (optionsObj.color.substr (2,2), 16);
                var blue = parseInt (optionsObj.color.substr (4,2), 16);
            }
        }
    });
}
```

Página 291

---

## Página 292

```
    imprimir ("vermelho:" + vermelho + ", verde:" + verde + ", azul:" + azul);
    colorLeds (vermelho, verde, azul);
}
imprimir ("URL do resultado =" + url);
response.writeHead (200, {
    "Access-Control-Allow-Origin": "*"
});
response.end ("");
}); // Fim de uma nova solicitação de navegador

httpServer.listen (80);
print ("Agora sendo um servidor HTTP!");
} // Fim do beServer

var ssid = "ssid";
var senha = "senha";
```

```
// Conecte-se ao ponto de acesso
var wi-fi = requer ("wi-fi");
imprimir ("Conectando ao ponto de acesso.");
wifi.connect (ssid, senha, nulo, função (err, ipInfo) {
    if (err) {
        imprimir ("Erro ao conectar ao ponto de acesso.");
        Retorna;
    }
    var ESP8266 = requer ("ESP8266");
    print ("Conectar diz que agora estamos conectados !!!");
    print ("Iniciando servidor web em http: // " + ESP8266.getIpAddressString (ipInfo.ip)
+ ": 80");
    beServer ();
});
```

Quando este aplicativo é executado, ele se conecta ao ponto de acesso WiFi local e, em seguida, inicia escutando as solicitações REST de entrada. Espera-se que um pedido de descanso tenha uma consulta parâmetro no final com o formato cor = valor onde o valor é codificado como 6 hex caracteres correspondentes à cor. Finalmente, podemos escrever uma página da web que apresentará um seletor de cores e, quando escolhemos uma cor, enviamos uma solicitação REST para o ESP8266 para iluminar os LEDs de forma adequada. Aqui está um exemplo de página da web para realizar esta tarefa:

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "ISO-8859-1">
<title> Definir as cores do LED </title>

<link
    href = "http://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.11.2/jquery-ui.min.css"
    rel = "folha de estilo" type = "text / css" />
<script
src = "http://cdnjs.cloudflare.com/ajax/libs/require.js/2.1.15/require.min.js"> </script>
<link rel = 'stylesheet' href = 'spectrum.css' />
<script>
    exigir
```

Página 292

## Página 293

```
.config ({
    baseUrl: "src",
    caminhos: {
        "jquery": "http://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.1/jquery.min",
        "jquery-ui": "http://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.11.2/jquery-
estou dentro",
    },
    calço: {
        "jquery-ui": {
            deps: ["jquery"],
            exportações: 'jQueryUI'
        }
    }
})
// Fim dos calços
});
require ("jquery", "spectrum", "jquery-ui"], function ($) {
    $ (function () {
        var allowHttp = true;
        $ ("# plano"). espectro ({
            plano: verdadeiro,
            preferredFormat: "rgb",
            mover: função (cor) {
                if (allowHttp) {
                    allowHttp = false;
                    $.ajax ({
                        url: "http://192.168.1.10",
                        dados : {
                            color: color.toHexString ()
                        },
                    })
                }
            }
        })
    })
})
```

```

        sucesso: function () {
            allowHttp = true;
        },
        erro: função () {
            allowHttp = true;
        }
    );
}
},
showInput: true,
showButtons: false
);
});
}); // Fim do carregamento
}); // Fim da exigência
</script>
</head>
<body>
    <div id = "flat" style = "width: 500px; height: 500px;"></div>
</body>
</html>

```

O resultado final, conforme visto na página da web, é o seguinte:

Página 293

## Página 294

A seleção de uma nova cor faz com que os dados sejam enviados para o ESP8266 que colore os LEDs apropriadamente com o resultado final geral sendo a capacidade de mudar a luz do humor do String LED.

### **Amostra - rede de inicialização**

Imagine que você receba um aplicativo ESP8266 fantástico e o instale em seu dispositivo. A probabilidade é que ele use uma rede com base em WiFi. Agora vem a questão interessante de como você inicializa o dispositivo?

### **Bibliotecas de amostra**

Há momentos em que funções comumente usadas podem ser capturadas e reutilizadas sobre. Esta seção descreve exatamente esse conjunto de funções que foram coletadas. A fonte para essas funções foi colocada no Github em <local a ser fornecido>.

As funções, quando compiladas, são colocadas em uma biblioteca chamada libcommon.a . Isso pode então ser linkado em seu Makefile para que referências não resolvidas a essas funções possam ser satisfeitas.

Um arquivo de cabeçalho chamado " common.h " é tudo o que é necessário para adicionar em seus próprios aplicativos.

## **Lista de funções**

### **authModeToString**

Dado um AUTH\_MODE, retorna uma representação de string do modo.

```
char * authModeToString (modo AUTH_MODE)
```

Página 294

## **Página 295**

### **checkError**

Verifique se há um erro no código de retorno.

```
void checkError (sint8 err)
```

Verifique a err código para um erro e se é um, registrá-lo.

### **delayMilliseconds**

Atrase por um período de milissegundos.

```
void delayMilliseconds (uint32 milissegundos)
```

Os parâmetros de milissegundos são o número de milissegundos de atraso antes de retornar.

### **dumpBSSINFO**

Despeje uma instância de struct bss\_info no log.

```
void dumpBSSINFO (struct bss_info * bssInfo)
```

### **dumpEspConn**

Despeja no log uma representação decodificada da estrutura espconn .

```
void dumpEspConn (struct espconn * pEspConn)
```

### **dumpRestart**

Despeje as informações de reinicialização no log.

```
void dumpRestart ()
```

Veja também:

- [Manipulação de exceção](#)

### **dumpState**

Despeje o estado da estação WiFi no registro.

```
void dumpState ()
```

Veja também:

- [Erro: fonte de referência não encontrada](#)

**Página 296****errorToString**

Dado um código de erro, retorna uma representação de string dele.

```
char * errorToString (sint8 err)
```

**eventLogger**

Grave um evento de WiFi no log.

```
void eventLogger (System_Event_t * event)
```

Podemos registrar essa função como um retorno de chamada para um evento WiFi. Grave os dados do evento no registro.

Veja também:

- [Tratamento de eventos WiFi](#)

**eventReasonToString**

Converte um motivo de evento em uma representação de string.

```
char * eventReasonToString (int reason)
```

Alguns dos retornos de chamada de evento WiFi podem retornar um valor de motivo que é uma codificação do razão pela qual algo falhou. Esta função retorna uma representação de string do int código do valor.

**flashSizeAndMapToString**

Retorne uma representação de string do tamanho do flash e do mapa.

```
char * flashSizeAndMapToString ()
```

**setAsGpio**

Defina um pino para ser usado como GPIO.

```
void setAsGpio (uint8 pin)
```

Defina o GPIO fornecido como pino para ser a função GPIO.

Veja também:

- [GPIOs](#)
- [GPIOs](#)

**setupBlink**

Configure um LED piscando no pino fornecido.

```
void setupBlink (uint8 blinkPin)
```

**Página 297**

O parâmetro blinkPin é o pino a ser usado para piscar.

### **toHex**

Converte uma matriz de bytes em uma string hexadecimal.

```
uint8 * toHex (uint8 * ptr, int size, uint8 * buffer)
```

Converte os bytes apontados por ptr para bytes de tamanho em uma string hexadecimal. O buffer parâmetro será onde o resultado será armazenado. Deve ter 2 \* tamanho + 1 bytes de comprimento (ou mais). Cada byte tem 2 caracteres hexadecimais mais um terminador NULL de byte único no final. A função retorna o início do buffer.

## **Usando FreeRTOS**

Quando pensamos em um computador moderno, rapidamente percebemos que ele tem um sistema operacional de algum tipo. Exemplos comuns são Microsoft Windows ou Linux. O objetivo de um sistema operacional é fornecer uma interface entre os aplicativos de software e a infraestrutura de hardware subjacente. Se não fosse por um sistema operacional, cada aplicativo provavelmente teria que realizar sua própria implementação semelhante de tais funções o que seria um desperdício. Por que não escrever uma vez e fornecer uma camada de abstração sobre quais funções de nível superior (como aplicativos) podem ser construídas. As capacidades de sistemas operacionais em PCs são muito semelhantes. Eles lidam com o gerenciamento de memória, I / O de hardware (leitura de teclados e mouses e placas gráficas de condução), tarefa gerenciamento (vários programas em execução ao mesmo tempo), disco e interações do sistema de arquivos e muito mais. Os primeiros sistemas operacionais forneciam funções básicas, enquanto os de hoje sistemas operacionais tornaram-se cada vez mais ricos a ponto de não poderem mais ser considerados apenas sistemas operacionais. Desde quando um sistema operacional precisa fornecer Freecell ou Minesweeper?

Se voltarmos o relógio e começarmos de novo e olharmos para os aspectos centrais de uma operação sistema, chegamos ao FreeRTOS de hoje. FreeRTOS é uma operação de código aberto sistema que fornece funções muito básicas para aplicativos de nível superior ... novamente ... a noção central da finalidade de um sistema operacional em primeiro lugar. Contudo,

O FreeRTOS foi projetado para sistemas embarcados como o ESP8266. São ordens de magnitude mais simples do que outros sistemas operacionais como o Linux, mas isso é intencional.

FreeRTOS foi portado para uma ampla variedade de plataformas de hardware, incluindo o CPUs Xtensa usadas no ESP8266. Quando compilado, resulta em uma biblioteca que está sob 5 mil bytes de tamanho.

As principais funções que ele fornece são:

- gerenciamento de memória

Página 297

---

## **Página 298**

- gerenciamento de tarefas
- sincronização de API

Veja também:

- Página inicial RTOS grátis
- [Estudo de um sistema operacional: FreeRTOS](#)
- Github: [espressif/ESP32\\_RTTOS\\_SDK](#)

## A arquitetura de uma tarefa no FreeRTOS

Comecemos com a noção de tarefa. Uma tarefa é um trabalho que desejamos realizar.

Se desejar, você pode pensar nisso como uma função da linguagem C. Por exemplo:

```
int add (int a, int b) {
    retornar a + b;
}
```

poderia ser considerada uma tarefa ... embora isso fosse ridiculamente simples. Genericamente, pense em uma tarefa como a execução de um trecho de código C de sua autoria. Nós normalmente pensamos no código em execução desde o início até o fim ... no entanto, esta não é necessariamente a maneira mais eficiente de proceder. Considere a ideia de um aplicativo que deseja enviar alguns dados pela rede. Ele pode querer enviar um megabyte de dados ... no entanto, também pode descobrir que só pode enviar 100 K de cada vez antes de ter que esperar que os dados transmitidos sejam entregues. Nessa história, enviaria 100K e aguarde a conclusão da transmissão, envie os próximos 100K e espere por isso transmissão para completar e assim por diante. Mas o que dizer daqueles períodos de tempo em que o código está esperando a conclusão de uma transmissão anterior? O que a CPU está fazendo nesses vezes?

As chances são de que ele não está fazendo nada além de monitorar a bandeira que afirma que a transmissão foi concluída. Isso é um desperdício. Em teoria, a CPU poderia estar funcionando outro trabalho (assumindo que há de fato outro trabalho que poderia ser executado). Se lá é de fato outro trabalho disponível, poderíamos "alternar o contexto" entre esses itens de trabalho de modo que, quando alguém bloqueia esperando que algo aconteça, o controle pode ser passado para outro para fazer algo útil.

Se chamarmos cada trabalho de "uma tarefa", esse é o valor de uma tarefa no FreeRTOS. A tarefa representa um trabalho a ser executado, mas em vez de assumir que o trabalho será ir rapidamente do início ao fim, estamos declarando que pode haver momentos dentro do trabalho onde pode ceder o controle a outro trabalho (tarefas).

Com isso em mente, devemos pensar em como uma tarefa é criada. Existe uma API fornecida por FreeRTOS chamado "xTaskCreate ()" que cria uma instância de uma tarefa.

Página 298

---

## Página 299

Aqui é importante perceber que uma tarefa é uma abstração lógica. Não tem nada específico fornecido na CPU que sabe o que é uma tarefa. Em vez disso, é a operação sistema (FreeRTOS no nosso caso) que está fornecendo o modelo da tarefa para nós.

Se pensarmos profundamente sobre uma tarefa, podemos conceber que a tarefa tenha um estado. Em qualquer com o tempo, uma tarefa está em execução ou não. Uma tarefa em execução é aquela que está usandoativamente a CPU (ou seja, não está esperando que mais nada aconteça). Uma tarefa que não é correr é aquele que não tem CPU. Por exemplo, se criarmos duas tarefas, uma das eles estariam correndo e o outro não. Se o que está em execução chegar a um ponto onde ele não pode mais realizar um trabalho significativo, ele abrirá mão do controle da CPU e torne-se não executando. A outra tarefa tem a oportunidade de ser executada.

Indo ainda mais fundo, quando uma tarefa não está em execução, ela pode "não estar em execução" para um determinado razão ... como:

- Bloqueado esperando algo ser concluído
- Suspenso pelo usuário
- Pronto para ser executado de forma que, quando a tarefa em execução não estiver mais em execução, este

tarefa está qualificada para ser executada

No FreeRTOS, definimos uma tarefa como uma função C que recebe um parâmetro void \*. Para exemplo,

```
void myTask(void * myParameters)
```

pode ser uma assinatura para uma função de tarefa.

Espera-se que uma função de tarefa seja executada para sempre. Se precisar acabar, deve se limpar antes de retornar invocando vTaskDelete () .

Quando uma tarefa cede o controle de volta para o sistema operacional, o sistema operacional pode ter uma escolha entre várias tarefas quanto a qual delas deve ser executada. Este processo de seleção é chamado de "agendamento". FreeRTOS usa o conceito de "prioridade" para determinar quais tarefa a ser executada em seguida. Cada tarefa que está pronta para ser executada é considerada um candidato potencial e aquele que tiver a prioridade mais alta se tornará aquele que está em execução.

Ao codificar diretamente para FreeRTOS em um ambiente não ESP8266, normalmente tem que fazer uma chamada para vTaskStartScheduler () para garantir que o agendador de tarefas é operacional. Isso não deve ser tentado no ambiente ESP8266 como os internos do ambiente ESP8266 já registrou outras tarefas e já iniciou o planejador.

Existem várias funções relacionadas ao temporizador no FreeRTOS que funcionam com a noção de "tiques", em que um tique é uma unidade de tempo. No ESP8266 FreeRTOS o intervalo de escala é 1/100 de segundo (10 ms).

Página 299

## Página 300

### Bloqueio e sincronização no RTOS

Com a noção de tarefas de processamento paralelo dentro do RTOS, devemos ter um mecanismo para sincronizar ações entre tarefas. Por exemplo, imagine uma tarefa que produz dados e uma segunda tarefa que consome dados produzidos pela primeira. A tarefa de produção deve ter um mecanismo que descreve que os dados foram produzidos e a tarefa de consumo deve ter um mecanismo para bloquear a espera pela produção de dados.

Uma forma de conseguir é através da noção de "grupo de eventos". Pense em um evento grupo como um conjunto de sinalizadores que podem ter o valor "0" ou "1". Uma tarefa pode definir o valor de um sinalizador e uma segunda tarefa podem ser configurados para esperar (bloquear) até que um sinalizador faça a transição de "0" para "1". O que isso significa é que há um sistema assíncrono e vagamente acoplado comunicação através do uso desses sinalizadores. De uma perspectiva de implementação, o RTOS fornece um tipo de dados denominado "identificador de grupo de eventos" que é implementado pelo tipo de dados opaco denominado " EventGroupHandle\_t ". Uma instância disso é criada por meio de um chamar xEventGroupCreate () . Devemos supor que um identificador de grupo de eventos pode conter um máximo de 8 sinalizadores distintos que são identificados como 0 a 7. Podemos definir o sinalizadores dentro de um grupo de eventos usando xEventGroupSetBits () e sinalizadores claros usando xEventGroupClearBits () . Se precisarmos obter os valores de um grupo de eventos, nós pode chamar xEventGroupGetBits () . Simplesmente alternar os bits não é tão útil, mas entramos no cerne da história com a chamada de função xEventGroupWaitBits () . Quando invocado, causa o chamador será bloqueado até que um ou mais bits nomeados sejam definidos.

### Listas em RTOS

O FreeRTOS fornece funções de processamento de lista.

## ESP8266 - Criação de aplicativos para RTOS

A Espressif distribui um SDK para a construção de aplicativos RTOS para o ESP8266. Este SDK é disponível no Github. Minha escolha pessoal para recuperar o SDK é usar o mais recente versão do Eclipse e usar suas ferramentas de recuperação Git embutidas.

A versão do FreeRTOS fornecida pela Espressif parece ser v7.5.2. O mais recente disponível do próprio FreeRTOS parece ser v8.2.3.

O Eclipse também fornece um ambiente para compilação de programa C. O sugerido C → Sinalizadores de compilação de objeto são:

Página 300

## Página 301

Parâmetro / sinalizador	Significado
-g	
-Wpointer-arith	
-Wundef	
-Werror	
-WI, -El	
-fno-inline-functions	
-nostdlib	
-mlongcalls	
-mtext-section-literals	
-funções-seções	
-fdatasections	

Para vincular, usamos os seguintes sinalizadores de vinculador:

```
-L $(SDK_PATH) / lib
-Wl, --gc-sections
-nostdlib
-T $(LD_FILE)
-Wl, --no-check-sections
-u call_user_start
-Wl, -static
-Wl, --start-group
-lminic
-lgcc
-lhal
-lphy
-lpp
-lnet80211
-lwpa
-lcrypto
-lmain
```

```
-lfreertos
-llwip
```

Ao configurar o CDT no Eclipse, as seguintes configurações são sugeridas:

- Compilação C / C ++ → Ambiente - ESP8266\_SDK\_ROOT = <Caminho para RTOS SDK>

Página 301

## Página 302

- C / C ++ Geral → Caminhos e Símbolos - GNU C -
 {ESP8266\_SDK\_ROOT} / incluir / espressif
- C / C ++ Geral → Caminhos e Símbolos - GNU C -
 {ESP8266\_SDK\_ROOT} / include / lwip
- C / C ++ Geral → Caminhos e Símbolos - GNU C -
 {ESP8266\_SDK\_ROOT} / include / lwup / ipv4

Tendo compilado o código-fonte em arquivos-objeto e, em seguida, vinculado os arquivos-objeto em um Executável formatado em ELF, o que resta é dividir este executável em seções para ser carregado na memória flash ESP8266 em locais diferentes. Isso resultará em dois arquivos para carregar no flash. Um arquivo serão os dados que eventualmente serão carregados na RAM em runtime enquanto o outro estará disponível como memória flash endereçável.

Podemos usar esptool\_ckpt para esta tarefa:

```
esptool_ckpt -eo bin.elf -bo app_0x00000.bin -bs .text -bs .data -bs .rodata -bs
.irom0.text -bc -ec
esptool_ckpt -eo $ bin.elf -es .irom0.text ap_0x10000.bin -ec
```

Depois de atualizar para o ESP8266, podemos descobrir que as seguintes mensagens de inicialização são produzido:

```
pp_task_hdl: 3ffef4e0, prio: 13, pilha: 512
pm_task_hdl: 3ffefdb0, prio: 1, pilha: 176
tcpip_task_hdl: 3ffef260, prio: 10, pilha: 512
idle_task_hdl: 3ffef300, prio: 0, pilha: 176
tim_task_hdl: 3ffff1428, prio: 2, pilha: 256
xPortStartScheduler
frc2_timer_task_hdl: 3fff1938, prio: 12, pilha: 512
```

OS SDK ver: 1.3.0 (68c9e7b) compilado em 2 de novembro de 2015 18:53:21  
phy ver: 484, pp ver: 9.9

Versão do SDK: 1.3.0 (68c9e7b)

Veja também:

- Github: [espressif / ESP8266\\_RTOS\\_SDK](https://github.com/espressif/ESP8266_RTOS_SDK)
- 

### Consoles com RTOS

O fluxo de depuração padrão é gravado no UART0 a uma taxa de transmissão de 74880.

---

**Página 303****Dicas de depuração**

Se as coisas ficarem estranhas, apague todo o flash do seu dispositivo e comece de novo.

No FreeRTOS, para fazer com que a depuração seja gravada no UART1, o seguinte pode ser usado:

```
UART_ConfigTypeDef uart_config;
uart_config.baud_rate = BIT_RATE_115200;
uart_config.data_bits = UART_WordLength_8b;
uart_config.parity = USART_Parity_None;
uart_config.stop_bits = USART_StopBits_1;
uart_config.flow_ctrl = USART_HardwareFlowControl_None;
uart_config.UART_RxFlowThresh = 120;
uart_config.UART_InverseMask = UART_None_Inverse;
UART_ParamConfig (UART1, &uart_config);
UART_SetPrintPort (UART1);
```

---

**Página 304****Desenvolvendo soluções em Linux**

Ao trabalhar em ambientes Linux, existem algumas dicas e técnicas que

[https://translate.googleusercontent.com/translate\\_f](https://translate.googleusercontent.com/translate_f)

pode ser útil / valioso.

- Ao conectar a uma placa SP8266 usando um conector USB → UART, o dispositivo pode aparecer em / dev como ttyUSB0 . Se examinarmos as permissões neste arquivo, podemos descobrir que ele está configurado como:

```
crw-rw ---- discagem root
```

Isso significa que ele pode ser acessado pelo root e pelos usuários do grupo de discagem . Se você deseja atualizar o ESP8266 por meio deste dispositivo, sua ID de usuário deve ser um membro deste grupo. Para adicionar seu usuário ao grupo, o seguinte Linux comando pode ser usado:

```
sudo usermod -a -G dialout <yourUserid>
```

depois de fazer a alteração, você deve fazer logout e login novamente.

- Um cliente de terminal útil é o GtkTerm . Esta ferramenta fornece um visualizador de terminal que pode ser usado para monitorar o conector USB → UART para visualizar mensagens de registro e depuração. Ele cria um arquivo de configuração em \$ HOME /.gtktermrc que pode ser editado para alterar a porta serial padrão (por exemplo, / dev / ttyUSB0 ), bem como alterar a taxa de transmissão para seu valor desejado.
- Outro bom cliente de terminal é a tela . Screen é um emulador de terminal de tela inteira.
- Ainda outro cliente de terminal é o comando cu clássico . Novamente, muito fácil de usar. Um exemplo de uso seria:

```
$ cu --line / dev / ttyUSB0 --velocidade 115200
```

Para sair de uma sessão cu, digite " ~ ".

### **Construindo um ambiente Linux**

Se você não executa o Linux nativamente, pode considerar executar o Oracle VirtualBox para hospedar um ambiente Linux em sua máquina Windows ou Mac. Oracle VirtualBox é um implementação de código aberto de um produto de virtualização de sistema operacional. Um pode baixe o VirtualBox aqui:

<https://www.virtualbox.org/>

Em meus testes, executei o Ubuntu 15.10.

Eu defino um tamanho de disco de pelo menos 20 GBytes e 2 GBytes de RAM. Se você tem vários núcleos, você pode definir aqueles como disponíveis.

Página 304

---

### **Página 305**

Depois de construir uma imagem, certifique-se de ativar a capacidade de copiar e colar entre o sistema operacional host e o sistema operacional convidado.

Certifique-se também de que as ferramentas de convidado do VirtualBox estão instaladas.

Existem alguns pacotes que você realmente não pode fazer sem incluir:

- git

Você pode instalar novos pacotes com:

```
sudo apt-get install <pacote>
```

Depois de instalar o sistema operacional Linux, queremos construir um ambiente de compilação.

O popular pfalcon / esp-open-sdk é o que iremos ilustrar.

Antes de iniciar o restante de nossa construção, devemos garantir que uma série de opcionais componente para Linux são instalados, pois são obrigatórios para construir o conjunto de ferramentas. O seguinte comando pode ser executado para instalar o conjunto de componentes de que precisamos:

```
sudo apt-get install make unrar autoconf automake libtool-bin gcc g++ gperf \
flex bison texinfo gawk ncurses-dev libexpat-dev python python-serial sed \
git unzip bash help2man wget bzip2
```

Primeiro, devemos executar o comando para baixar o projeto baseado no github:

```
git clone --recursive https://github.com/pfalcon/esp-open-sdk.git
```

Depois de fazer o download, podemos construir a solução com:

```
fazer STANDALONE = y
```

Observe que a construção precisa de acesso à rede e acessará sites externos, incluindo:

Página 305

## Página 306

- [www.mpfr.org](http://www.mpfr.org)

A construção / compilação levará cerca de uma hora para ser concluída.

Na conclusão, um conjunto de novos diretórios pode ser encontrado. Entre eles estão:

- xtensa-lx106-elf / bin - As ferramentas compiladas incluindo gcc, objcopy e gdb.
- sdk - Um link simbólico para o Espressif SDK mais recente

Queremos adicionar algumas variáveis de ambiente em nosso perfil de usuário:

- Adicione o xtensa-lx106-elf / bin ao PATH
- Exportar ESP8266\_SDK\_ROOT para a raiz do SDK

Você também vai querer instalar um esptool-ck em sua pasta bin local.

Você também desejará adicionar seu ID de usuário ao grupo chamado dialout.

Aqui está um exemplo de Makefile para Linux:

```
PROJ_NAME = test1
COMPORT = /dev/ttyUSB0
OBJS = user_main.o uart.o
#
CC = xtensa-lx106-elf-gcc
OBJS = user_main.o uart.o
APP = a.out
```

```

ESPTOOL_CK = esptool
CCFLAGS = -Wimplicit-function-declaration -fno-inline-functions -mlongcalls -mtext-section-literals \
-mno-serialize-volatile -I $ (ESP8266_SDK_ROOT) / incluir -I . -D __ETS__ -DICACHE_FLASH -DXTENSA -DUSE_US_TIMER

LDFLAGS = -nostdlib \
-L $ (ESP8266_SDK_ROOT) / lib -L $ (ESP8266_SDK_ROOT) / ld -T $ (ESP8266_SDK_ROOT) /ld/eagle.app.v6.ld \
-Wl , -no-check-sections -u call_user_start -Wl , -static -Wl , -start-group \
-Wl , -lgec -lhal -lphy -lpp -lnet80211 -lwip -lwpa -lmain -ljson -lupgrade -lssl \
-lpwm -lsmartconfig -Wl , -end-group

todos: $ (PROJ_NAME)_0x00000.bin $ (PROJ_NAME)_0x40000.bin

a.out: $ (OBJS)
$ (CC) → a.out $ (LDFLAGS) $ (OBJS)

$ (PROJ_NAME)_0x00000.bin: a.out
$ (ESPTOOL_CK) -eo $ <-bo $ @ -bs .text -bs .data -bs .rodata -bs .iram0.text -bc -ec || verdadeiro

$ (PROJ_NAME)_0x40000.bin: a.out
$ (ESPTOOL_CK) -eo $ <-es .irom0.text $ @ -ec || verdadeiro

.co:
$ (CC) $ (CCFLAGS) -c $ <

limpar:
rm -f a.out *.o *.bin

flash: tudo
$ (ESPTOOL_CK) -cp $ (COMPRT) -cd nodemcu -cb 115200 -ea 0x00000 -cf $ (PROJ_NAME)_0x00000.bin
$ (ESPTOOL_CK) -ep $ (COMPRT) -cd nodemcu -cb 115200 -ea 0x40000 -cf $ (PROJ_NAME)_0x40000.bin

```

E também um aplicativo simples " hello world ":

Página 306

## Página 307

```

#include "osapi.h"
#include "user_interface.h"
#include "uart.h"

#include "espmissingincludes.h"

void uart_init_2 (UartBautRate uart0_br, UartBautRate uart1_br);

void systemInitDoneCB () {
    os_printf ("Olá, mundo \n");
}

void user_init () {
    uart_init_2 (115200, 115200);
    system_init_done_cb (systemInitDoneCB);
}

```

Em seguida, instalamos o Java 8. Baixe do Oracle e extraia para / usr / java. Por exemplo arquivo tar zxvf. Atualize JAVA\_HOME e PATH.

Agora que temos um ambiente de compilação, são grandes as chances de querermos um IDE.... meu favorito é Eclipse.

<https://eclipse.org/downloads/>

Execute o instalador Eclipse

Página 307

---

**Página 308**

Página 308

---

**Página 309**

Página 309

---

**Página 310**

Página 310

---

**Página 311**

Depois de lançar o Eclipse, queremos instalar as atualizações. No entanto, para Marte, há não será nenhum, pois é tão novo.

Instale GtkTerm

Página 311

---

**Página 312**

Execute-o uma vez para criar o arquivo `~/.gtktermrc`. Edite esse arquivo e altere:

- porta = /dev/ttyUSB0
- velocidade = 115200

Adicionar o usuário atual ao grupo de discagem

```
sudo usermod -a -G dialout <yourUserid>
```

Você precisará fazer logout e login novamente para que isso tenha efeito. Você pode validar que seu userid tem o grupo correto executando o comando id a partir de um shell:

```
$ id
uid=1000(kolban) gid=1000(kolban) grupos=1000(kolban),4(adm),20(dialout),24(cdrom),
27(sudo),30(dip),46(plugdev),108(lpadmin),124(sambashare)
```

Download [igrr / esptool-ck](#) como esptool dos lançamentos do Github. Certifique-se de que está na sua lixeira pasta e essa pasta bin está no seu caminho.

Agora estamos finalmente prontos para construir um aplicativo. Podemos baixar uma amostra desta história em Github no seguinte URL:

<https://github.com/nkolban/Sample-ESP8266-App.git>

Este exemplo assume que as ferramentas xtensa estão em seu PATH e que o ambiente a variável ESP8266\_SDK\_ROOT está definida corretamente.

Eu também instalaria:

- [cromada](#)

Página 312

## Página 313

Veja também:

- [Oracle Virtual Box](#)
- [Downloads do Ubuntu](#)

## Referência API

Agora temos uma mini referência à sintaxe de muitas das APIs expostas do ESP8266.

Não use esta referência exclusivamente. Consulte também o Espressif SDK publicado Guia de programação.

Alguns acrônimos e outros nomes são usados na nomenclatura de APIs e podem precisar de alguma explicação para apreciá-los plenamente:

- dhcpc - cliente DHCP
- dhcps - servidor DHCP
- softap - Ponto de acesso implementado no software
- wps - Configuração protegida por WiFi
- sntp - Protocolo de Tempo de Rede Simples
- mdns - Sistema de Nome de Domínio Multicast
- uart - receptor / transmissor assíncrono universal
- pwm - modulação por largura de pulso

## Referência da API FreeRTOS

Veja também:

- [Usando FreeRTOS](#)

### eTaskGetState

Recupere o estado de uma tarefa.

`eTaskState eTaskGetState (TaskHandle_t xTask)`

### pcTaskGetName

Obtenha o nome da tarefa.

`char * pcTaskGetTaskName (TaskHandle_t xTaskToQuery)`

Página 313

## Página 314

### xEventGroupClear

Limpe um ou mais bits em um grupo de eventos.

`EventBits_t xEventGroupClearBits (EventGroupHandle_t eventGroup,  
const EventBits_t bitsToClear)`

- `eventGroup` - O grupo de eventos que contém os bits a serem limpos.
- `bitsToClear` - O conjunto de bits a serem limpos no grupo de eventos.

Inclui:

- `<freertos / event_groups.h>`

### xEventGroupCreate

Crie um novo grupo de eventos FreeRTOS.

`EventGroupHandle_t xEventGroupCreate ()`

Inclui:

- `<freertos / event_groups.h>`

### xEventGroupSetBits

Defina um ou mais bits em um grupo de eventos.

`EventBits_t xEventGroupSetBits (EventGroupHandle_t eventGroup,  
const EventBits_t bitsToSet)`

- `eventGroup` - O grupo de eventos que contém os bits a serem definidos.
- `bitsToSet` - O conjunto de bits a ser definido no grupo de eventos.

Inclui:

- `<freertos / event_groups.h>`

### xEventGroupWaitBits

Bloco à espera de um ou mais "bits" serem definidos.

`EventBits_t xEventGroupWaitBits (`

```
const EventGroupHandle_t eventGroup,
const EventBits_t bitsToWaitFor,
const BaseType_t clearOnExit,
const BaseType_t waitForAllBits,
TickType_t ticksToWait)
```

Chamar essa função pode fazer com que o chamador bloquee até que os bits sejam definidos ou até que o tempo limite seja atingido. Os bits podem ser definidos por meio de xEventGroupSetBits () .

Página 314

## Página 315

- eventGroup - O grupo de eventos que faz referência aos bits a serem observados.
- bitsToWaitFor - O conjunto de bits dentro do grupo de eventos que estamos esperando.
- clearOnExit - Os bits que estamos esperando devem ser apagados automaticamente quando definido e este método retorna?
- waitForAllBits - Devemos desbloquear no primeiro bit definido que estamos observando ou, alternativamente, devemos esperar que todos os bits sejam definidos?
- ticksToWait - Por quantos ticks devemos esperar antes de retornar? Esta fornece um tempo limite (se necessário). A variável RTOS " portMAX\_DELAY " pode ser usada para especificar que desejamos esperar indefinidamente.

Inclui:

- <freertos / event\_groups.h>

### xTaskCreate

Crie uma nova instância de uma tarefa.

```
BaseType_t xTaskCreate (
    pdTASK_CODE pvTaskCode,
    const assinado portCHAR * pcName,
    porta não assinadaSHORT usStackDepth,
    void * pvParameters,
    portBASE_TYPE uxPriority unsigned,
    xTaskHandle * pxCreatedTask)
```

- pvTaskCode - Ponteiro para a função de tarefa. Na programação C, podemos simplesmente fornecer o nome de uma função ou, como foi visto em alguns exemplos, o endereço do nome da função. Aparentemente, isso equivale a itens que são perto o suficiente para ser usado de forma intercambiável.
- pcName - nome de depuração da tarefa.
- usStackDepth - Tamanho da pilha para a tarefa.
- pvParameters - Parâmetros para a instância da tarefa. Isso pode ser NULL.
- uxPriority - Prioridade da instância da tarefa.
- xTaskHandle - Referência à instância de tarefa recém-criada. Isso pode ser passado como NULL se nenhum identificador de tarefa precisar ser retornado.

Quando uma tarefa criada é invocada e, em seguida, decide terminar por meio de um retorno, é essencial que a tarefa chama vTaskDelete (NULL) antes de ser concluída. Ligar para isso é uma indicação para

---

**Página 316**

FreeRTOS que a tarefa foi concluída e não precisa mais ser considerada para o contexto comutação.

Veja também:

- [vTaskDelete](#)
- [xTaskCreate](#)

### vTaskDelay

Atrasa uma tarefa por um número especificado de ticks.

```
void vTaskDelay (const TickType_t xTicksToDelay)
```

A constante chamada portTICK\_PERIOD\_MS fornece o número de ticks em um milissegundo.

Se desejássemos atrasar 1 segundo, poderíamos fornecer  $1000 / \text{portTICK\_PERIOD\_MS}$ .

Veja também:

- [vTaskDelay](#)

### vTaskDelayUntil

Atrasa uma tarefa até um tempo absoluto especificado.

```
void vTaskDelayUntil (const TickType_t * pxPreviousWakeTime, const TickType_t xTimeIncrement)
```

Esta função bloqueia uma tarefa até algum momento absoluto no futuro.

- pxPreviousWakeTime - O tempo base a partir do qual o incremento será relativo.
- xTimeIncrement - O tempo em ticks que, quando adicionado ao pxPreviousWakeTime , será o momento em que a tarefa estará pronta para ser executada novamente.

Veja também:

- [vTaskDelayUntil](#)

### vTaskDelete

Exclua uma instância de uma tarefa.

```
void vTaskDelete (TaskHandle_t pxTask)
```

Esta função excluirá uma instância de uma tarefa. Se o identificador pxTask for NULL , o a tarefa atual será excluída.

Veja também:

- [xTaskCreate](#)
- [vTaskDelete](#)

---

**Página 317**

### xTaskGetCurrentTaskHandle

Obtenha o identificador de tarefa atual.

**xTaskGetTickCount**

Obtenha a contagem de ticks atual.

```
portTickType xTaskGetTickCount ()
```

Retorne o número de tiques que ocorreram desde que o agendador de tarefas foi iniciado.

**vEventGroupDelete**

Exclua um grupo de eventos.

```
void vEventGroupDelete (EventGroupHandle_t eventGroup)
```

Inclui:

- <freertos / event\_groups.h>

**vTaskList**

```
void vTaskList (char * pcWriteBuffer)
```

**NÃO DISPONÍVEL****vTaskPrioritySet**

```
void vTaskPrioritySet (TaskHandle_t pxTask, UBaseType_t uxNewPriority)
```

Veja também:

- [vTaskPrioritySet](#)

**vTaskResume**

```
void vTaskResume (TaskHandle_t pxTaskToResume)
```

**xTaskResumeAll**

Veja também:

- [vTaskResumeAll](#)

**vTaskResumeFromISR**

```
void xTaskResumeFromISR (TaskHandle_t pxTaskToResume)
```

Página 317

**Página 318****vTaskSuspend**

```
void vTaskSuspend (TaskHandle_t pxTaskToSuspend)
```

**vTaskSuspendAll**

Veja também:

- [vTaskSuspendAll](#)

**xQueueCreate**

Crie uma fila para guardar itens.

```
xQueueHandle xQueueCreate (
    unsigned portBASE_TYPE uxQueueLength,
    unsigned portBASE_TYPE uxItemSize)
```

- uxQueueLength - O número máximo de itens que a fila pode conter.
- uxItemSize - O tamanho em bytes reservados para cada elemento na fila.

Veja também:

- [xQueueCreate](#)

### **vQueueDelete**

```
void vQueueDelete (xQueueHandle xQueue)
```

### **xQueuePeek**

```
portBASE_TYPE xQueuePeek (
    xQueueHandle xQueue,
    void * pvBuffer,
    portTickType xTicksToWait)
```

### **xQueueReceive**

```
portBASE_TYPE xQueueReceive (
    xQueueHandle xQueue,
    void * pvBuffer,
    portTickType xTicksToWait)
```

### **xQueueSend**

```
portBASE_TYPE xQueueSend (
    xQueueHandle xQueue,
    const void * pvItemToQueue,
    portTickType xTicksToWait)
```

Página 318

## **Página 319**

### **xQueueSendToBack**

```
portBASE_TYPE xQueueSendToBack (
    xQueueHandle xQueue,
    const void * pvItemToQueue,
    portTickType xTicksToWait);
```

### **xQueueSendToFront**

```
portBASE_TYPE xQueueSendToFront (
    xQueueHandle xQueue,
    const void * pvItemToQueue,
    portTickType xTicksToWait)
```

### **vSemaphoreCreateBinary**

### **xSemaphoreCreateCounting**

### **vSemaphoreGive**

**xSemaphoreGiveFromISR**

**vSemaphoreTake**

**pvPortMalloc**

**pvPortFree**

### Processamento de lista

**vListInitialise**

Inicialize uma lista.

`void vListInitialise (xList * const pxList)`

O pxList é uma lista que deve ser inicializada.

**vListInitialiseItem**

Inicialize um item para inserção em uma lista.

Página 319

## Página 320

`void vListInitialiseItem (xListItem * const pxItem)`

Inicialize um item que pode ser adicionado a uma lista.

**vListInsert**

Insira um item em uma lista.

`void vListInsert (xList * const pxList, xListItem * const pxNewItem)`

**vListInsertEnd**

Insira um item no final de uma lista

`void vListInsertEnd (xList * const pxList, xListItem * const pxNewItem)`

## Referência lwip

### tomadas

- accitar
- ligar
- desligar
- closesocket
- conectar
- getsocketname
- getpeername
- setssockopt

- getsockopt
- ouço
- recv
- recvfrom
- enviar
- enviar para
- soquete
- selecionar

Página 320

## Página 321

- ioctlsocket
- leitura
- Escreva
- perto
- fcntl

### Funções de cronômetro

As funções de cronômetro nos permitem registrar funções que serão executadas em um momento no futuro ou periodicamente depois que o tempo passa. Também agrupamos funções que manipulam ou recuperam valores de tempo neste conjunto.

#### **os\_delay\_us**

Atraso de microssegundos.

```
void os_delay_us (uint16 us)
```

Atraso para um intervalo máximo de 65535 microssegundos.

Inclui:

- osapi.h

Veja também:

- [Timers e tempo](#)
- [system\\_set\\_os\\_print](#)

#### **os\_timer\_arm**

Habilite um cronômetro de granularidade de milissegundos.

```
void os_timer_arm (
    os_timer_t * pTimer,
    uint32_t milissegundos,
    bool repeat)
```

Arme um cronômetro de forma que comece a funcionar e dispare quando o relógio chegar a zero.

O parâmetro pTimer é apontado para uma estrutura de controle do temporizador.

O parâmetro milissegundos é a duração do cronômetro medido em milissegundos.

O parâmetro de repetição é se o cronômetro irá reiniciar ou não ao atingir zero.

Inclui:

- osapi.h

Veja também:

- [Timers e tempo](#)

Página 321

## Página 322

- [os\\_timer\\_disarm](#)
- [os\\_timer\\_setfn](#)

### **os\_timer\_disarm**

Desarmar / Cancelar um cronômetro previamente armado.

```
void os_timer_disarm (os_timer_t * pTimer)
```

Pare um cronômetro iniciado anteriormente que foi iniciado por uma chamada para `os_timer_arm ()`.

O parâmetro `pTimer` é um ponteiro para uma estrutura de controle do temporizador.

Inclui:

- `osapi.h`

Veja também:

- [Timers e tempo](#)
- [os\\_timer\\_arm](#)
- [os\\_timer\\_setfn](#)

### **os\_timer\_setfn**

Defina uma função a ser chamada quando o cronômetro disparar

```
void os_timer_setfn (
    os_timer_t * pTimer,
    os_timer_func_t * pFunction,
    void * pArg)
```

Defina a função de retorno de chamada que será chamada quando o cronômetro chegar a zero.

Os parâmetros `pTimer` são um ponteiro para a estrutura de controle do temporizador.

Os parâmetros `pFunction` são um ponteiro para a função de retorno de chamada.

O parâmetro `pArg` é um valor que será passado para a função de retorno chamada.

A função de retorno de chamada deve ter a assinatura:

```
void (* functionName) (void * pArg)
```

O parâmetro `pArg` é o valor registrado com a função de retorno de chamada.

Inclui:

- `osapi.h`

Veja também:

- [Timers e tempo](#)
- [os\\_timer\\_arm](#)
- [os\\_timer\\_disarm](#)

Página 322

**Página 323****system\_timer\_reinit**

Usado para definir um cronômetro de microssegundos

**os\_timer\_arm\_us**

Habilitar um cronômetro de micro segundos

**hw\_timer\_init**

Inicializar um temporizador de hardware

**hw\_timer\_arm**

Defina o atraso de disparo

**hw\_timer\_set\_func**

Definir o cronômetro de retorno

**Funções do sistema****system\_adc\_read**

Leia o valor do conversor A / D.

`uint16 system_adc_read ()`

Leia o valor do conversor analógico para digital. A granularidade é de 1024 etapas discretas.

Veja também:

- [Conversão analógica para digital](#)

**system\_deep\_sleep\_set\_option**

Defina o que o chip fará na próxima vez que for ativado.

`bool system_deep_sleep_set_option (opção uint8)`

**system\_get\_boot\_mode**

Obtenha o modo de inicialização atual

`uint8 system_get_boot_mode ()`

Página 323

---

**Página 324**

O valor de retorno indica o modo de inicialização atual e será um dos seguintes:

- `SYS_BOOT_ENHANCE_MODE` - 0
- `SYS_BOOT_NORMAL_MODE` - 1

Em meus dispositivos, o valor retornado é "0".

**system\_get\_boot\_version**

A versão do carregador de boot.

```
uint8 system_get_boot_version()
```

O valor atual retornado pelo teste de meus dispositivos é "5".

**system\_get\_chip\_id**

Obtenha a id do chip

```
long system_get_chip_id()
```

Por exemplo: 0xf94322

**system\_get\_cpu\_freq**

Obtenha a frequência atual da CPU

```
int system_get_cpu_freq()
```

Retorna a frequência da CPU em MHz. O valor será 80 ou 160.

**system\_get\_flash\_size\_map**

Obtenha o tamanho atual do flash e mapa

```
enum flash_size_map system_get_flash_size_map()
```

O valor retornado é um enum que possui as seguintes definições:

- FLASH\_SIZE\_4M\_MAP\_256\_256
- FLASH\_SIZE\_2M
- FLASH\_SIZE\_8M\_MAP\_512\_512
- FLASH\_SIZE\_16M\_MAP\_512\_512
- FLASH\_SIZE\_32M\_MAP\_512\_512
- FLASH\_SIZE\_16M\_MAP\_1024\_1024
- FLASH\_SIZE\_32M\_MAP\_1024\_1024

Veja também:

Página 324

**Página 325**

- [Carregando um programa no ESP8266](#)

**system\_get\_rst\_info**

Informações sobre a inicialização atual.

```
struct rst_info * system_get_rst_info()
```

Recupere informações sobre a inicialização do dispositivo atual.

Inclui:

- user\_interface.h

Veja também:

- [Manipulação de exceção](#)
- [struct rst\\_info](#)

**system\_get\_userbin\_addr**

Obtenha o endereço do bin do usuário

```
uint32 system_get_userbin_addr()
```

O valor atual retornado em meus dispositivos é 0x0 .

**system\_get\_vdd33**

Meça a tensão

Desconhecido ... mas relacionado à conversão de analógico para digital.

Veja também:

- [Conversão analógica para digital](#)
- [Erro: fonte de referência não encontrada](#)

**system\_init\_done\_cb**

Registre uma função a ser chamada quando a inicialização do sistema for concluída

```
void system_init_done_cb (init_done_cb_t callbackFunction)
```

Esta função foi projetada para ser chamada apenas em user\_init () . Irá registrar uma função para ser chamado uma vez após a inicialização do ESP8266. O init\_done\_cb\_t define um função:

```
void (* functionName) (void)
```

Página 325

**Página 326**

Inclui:

- user\_interface.h

Veja também:

- [Programas personalizados](#)

**system\_os\_post**

Publique uma mensagem em uma tarefa.

```
bool system_os_post (prioridade uint8,
                     sinal os_signal_t,
                     parâmetro os_param_t)
```

Publique uma mensagem em uma tarefa. A tarefa não será executada imediatamente, mas será executada assim que posso.

O campo de prioridade é a prioridade da solicitação de tarefa. Três valores são definidos -

USER\_TASK\_PRIO\_0 , USER\_TASK\_PRIO\_1 e USER\_TASK\_PRIO\_2 .

O parâmetro de sinal é usado pelo manipulador de tarefas para determinar quem deve processar o sinal. Na verdade, é um uint32\_t .

O parâmetro parâmetro é usado para passar dados opcionais para o manipulador.

O retorno é verdadeiro em caso de sucesso e falso em caso de falha.

Inclui:

- user\_interface.h

Veja também:

- [ESP8266 Gerenciamento de tarefas](#)

### **system\_os\_task**

Configure uma tarefa para execução posterior.

```
bool system_os_task (tarefa os_task_t,
                     prioridade uint8,
                     os_event_t * queue,
                     uint queueLength)
```

O "os\_task\_t" é um ponteiro para uma função do gerenciador de tarefas que tem a assinatura:

```
void (* functionName) (os_event_t * event)
```

Esta função é definida para ser um gerenciador de tarefas que receberá todas as postagens diferentes notificações do mesmo nível de prioridade.

O os\_event\_t é uma estrutura que contém:

- sinal os\_signal\_t

Página 326

## **Página 327**

- os\_param\_t param

Ambos são inteiros de 32 bits sem sinal.

O campo de prioridade é a prioridade da solicitação de tarefa. Três valores são definidos:

- USER\_TASK\_PRIO\_0
- USER\_TASK\_PRIO\_1
- USER\_TASK\_PRIO\_2

O retorno é verdadeiro em caso de sucesso e falso em caso de falha.

Inclui:

- user\_interface.h

Veja também:

- [ESP8266 Gerenciamento de tarefas](#)

### **system\_phys\_set\_rfoption**

Habilite o RF após acordar de um sono (ou não)

### **system\_phys\_set\_max\_tpw**

Defina a potência máxima de transmissão

### **system\_phys\_set\_tpw\_via\_vdd33**

Defina a potência de transmissão como uma função da tensão

### **system\_print\_meminfo**

Imprimir informação de memória

```
void system_print_meminfo ()
```

As informações de memória para diagnóstico são gravadas no fluxo de saída, que é comumente UART1. O formato dos dados é o seguinte:

```

dados: 0x3ffe8000 ~ 0x3ffe853c, len: 1340
rodata: 0x3ffe8540 ~ 0x3ffe8af0, len: 1456
bss: 0x3ffe8af0 ~ 0x3ffffc18, len: 37160
heap: 0x3ffffc18 ~ 0x3fffc000, len: 41960

```

A seção .data é onde as variáveis locais inicializadas globais e estáticas são mantidas.

A seção .rodata é onde os dados globais e estáticos somente leitura são mantidos.

O .bss é onde os dados estáticos globais e locais não inicializados são mantidos.

Página 327

## Página 328

O .heap é onde o heap do programa pode ser encontrado.

Veja também:

- [Wikipedia - bss](#)
- [Wikipedia - segmento de dados](#)

### **system\_restart\_enhance**

Reinicia o sistema em modo de inicialização aprimorada

### **system\_rtc\_clock\_cali\_proc**

Calibração do relógio.

```
uint32 system_rtc_clock_cali_proc (void)
```

Recupere a calibração do relógio em tempo real. Esta é a duração do relógio de parede de um ciclo de relógio medido em micro segundos. O número de 16 bits retornado tem bits 11-0 que representam o valor após o ponto decimal. Podemos multiplicar o valor retornado aqui pelo número de ciclos desde uma reinicialização anterior e determinar um valor de relógio de parede decorrido.

### **system\_set\_os\_print**

Ative ou desative o registro.

```
void system_set_os_print (uint8 onOff)
```

Um valor de 0 desliga-o enquanto um valor de 1 liga-o. Foi inicialmente pensado que este log de nível de sistema operacional controlado, entretanto, parece controlar **todo o** log por meio de `os_printf()`.

Inclui:

- `user_interface.h`

Veja também:

- [os\\_printf](#)
- [os\\_install\\_putchar](#)
- [Registrando no UART1](#)

### **system\_show\_malloc**

Depure possíveis problemas de vazamento de memória.

```
void system_show_malloc ()
```

Esta API também deve ser habilitada definindo explicitamente `MEMLEAK_DEBUG`.

A documentação sobre esta função no guia de programação SDK fornece uma série de avisos e advertências que ainda não foram totalmente compreendidos, portanto, use com cuidado.

---

**Página 329****system\_rtc\_clock\_cali\_proc**

Calibração do relógio.

```
uint32 system_rtc_clock_cali_proc (void)
```

Recupere a calibração do relógio em tempo real. Esta é a duração do relógio de parede de um ciclo de relógio medido em micro segundos. O número de 16 bits retornado tem bits 11-0 que representam o valor após o ponto decimal. Podemos multiplicar o valor retornado aqui pelo número de ciclos desde uma reinicialização anterior e determinar um valor de relógio de parede decorrido.

**system\_uart\_swap**

Troque UARTs seriais.

Quando um ESP8266 é inicializado, ele usa certos pinos para controle UART0. Especificamente, precisa de pinos para as funções TX, RX, CTS e RTS. Ao chamar esta função, os pinos físicos usados para UART0 são trocados.

<b>Função</b>	<b>Padrão</b>	<b>Trocado</b>
U0TXD	U0TXD	MTDO
U0RXD	U0RXD	MTCK
U0CTS	MTCK	U0RXD
U0RTS	MTDO	U0TXD

**system\_soft\_wdt\_feed**

Alimente o watchdog do software.

```
void system_soft_wdt_feed ()
```

Alimente o watchdog do software. A função só tem valor quando o watchdog do software está ativado. Se precisarmos executar looping em nosso código, precisamos chamar esta função periodicamente para que não diminuamos o tempo de execução do WiFi. O motivo aqui é fome e comida e, portanto, a noção de um cronômetro de vigilância que verifica se não gastamos muito tempo longe de WiFi ... então devemos alimentar o cachorro. Metáforas interessantes.

No entanto ... experimentos estão mostrando que não parece realmente FAZER nada. O mistério de seu propósito continua. Ver:<http://bbs.espressif.com/viewtopic.php?f=7&t=1055>

**system\_soft\_wdt\_stop**

Desative o watchdog do software.

```
void system_soft_wdt_stop ()
```

Pare o watchdog do software. Recomenda-se não interromper este cronômetro por muito tempo (8 segundos ou menos), caso contrário, o watchdog do hardware forçará uma reinicialização.

---

**Página 330**

Veja também:

[https://translate.googleusercontent.com/translate\\_f](https://translate.googleusercontent.com/translate_f)

- [Cronômetro de vigilância](#)

### **system\_soft\_wdt\_restart**

Reinic peace o watchdog do software.

```
void system_soft_wdt_restart ()
```

Reinic peace o watchdog do software seguindo uma chamada anterior para interrompê-lo.

Veja também:

- [Cronômetro de vigilância](#)

### **system\_uart\_de\_swap**

Volte para o UART original.

### **system\_update\_cpu\_freq**

Defina a frequência da CPU

```
void system_update_cpu_freq (int freq)
```

Defina a frequência da CPU. 80 ou 160.

### **os\_memset**

Defina os valores da memória

```
void os_memset (void * pBuffer, valor int, size_t size)
```

Defina a memória apontada por pBuffer para o valor de bytes de tamanho .

Inclui:

- osapi.h

Veja também:

- [Trabalhando com a memória](#)
- [os\\_memcpy](#)

### **os\_memcmp**

Compare duas regiões da memória.

```
int os_memcmp (uint8 * ptr1, uint8 * ptr2, tamanho int)
```

Compare duas regiões da memória. O retorno é 0 se eles forem iguais.

Inclui:

- osapi.h

Página 330

## **Página 331**

### **os\_memcpy**

Copie os valores da memória.

```
void os_memcpy (void * destination, void * source, size_t size)
```

Copie a memória do buffer apontado por fonte para o buffer apontado por destino para o número de bytes especificado por tamanho .

Inclui:

- osapi.h

Veja também:

- [Trabalhando com a memória](#)
- [os\\_memset](#)

### **os\_malloc**

Alocar armazenamento do heap.

```
void * malloc (size_t size)
```

Alocar bytes de tamanho do heap e retornar um ponteiro para o armazenamento alocado.

Inclui:

- mem.h

Veja também:

- [Trabalhando com a memória](#)
- [os\\_zalloc](#)
- [os\\_free](#)

### **os\_calloc**

Aloque armazenamento para um conjunto de elementos.

```
void * calloc (size_t num, size_t size)
```

Aqui, alocamos num instâncias de objetos de tamanho dimensionado na memória contígua.

Inclui:

- mem.h

### **os\_realloc**

Realoque um pedaço de memória obtido anteriormente com um novo tamanho.

```
void * os_realloc (void * buf, size_t newSize)
```

Inclui:

Página 331

## **Página 332**

- mem.h

### **os\_zalloc**

Alocar armazenamento do heap e zerar seus valores.

```
void * os_zalloc (size_t size)
```

Alocar bytes de tamanho do heap e retornar um ponteiro para o armazenamento alocado. Antes retornando, a área de armazenamento é zerada.

Inclui:

- mem.h

Veja também:

- [Trabalhando com a memória](#)
- [os\\_malloc](#)
- [os\\_free](#)

### **os\_free**

Libere o armazenamento alocado anteriormente de volta para o heap.

```
void os_free (void * pBuffer)
```

Libere o armazenamento anteriormente alocado por `os_malloc()` ou `os_zalloc()` de volta para o pilha.

Inclui:

- `mem.h`

Veja também:

- [Trabalhando com a memória](#)
- [os\\_malloc](#)
- [os\\_zalloc](#)

### **os\_bzero**

Defina os valores da memória para zero.

```
void os_bzero (void * pBuffer, size_t size)
```

Define o ponteiro de dados por `pBuffer` para zero para bytes de tamanho .

Inclui:

- `osapi.h`

Veja também:

- [Trabalhando com a memória](#)

Página 332

## **Página 333**

### **os\_delay\_us**

Atraso de microsegundos.

```
void os_delay_us (uint16 us)
```

Atraso para um intervalo máximo de 65535 microsegundos.

Inclui:

- `osapi.h`

Veja também:

- [Timers e tempo](#)
- [system\\_set\\_os\\_print](#)

### **os\_printf**

Imprima uma string para UART.

```
void os_printf (formato char *,...)
```

Os sinalizadores de formato que funcionam incluem:

- % d - exibe um inteiro
- % ld - exibe um inteiro longo
- % lu - exibe um inteiro longo sem sinal
- % x - exibir como um número hexadecimal
- % s - exibir como uma string
- "\n" - exibe uma nova linha (inclui um retorno de carro prefixado)

Observe que não há % f para imprimir um float ou double.

O texto de saída é enviado para a função registrada com `os_install_putc()`. Por padrão,

este é UART0, mas pode ser alterado para UART1 definindo o `uart1_write_char()` função.

Inclui:

- `osapi.h`

Veja também:

- [Depurando](#)
- [os\\_install\\_putc1](#)
- [system\\_set\\_os\\_print](#)

### **os\_install\_putc1**

Registrar uma função imprimir um caractere

```
void os_install_putc1 (void (* pFunc) (char c));
```

Página 333

## Página 334

Registre uma função que será chamada por funções de saída como `os_printf()` que irão saída de log. Por exemplo, isso pode ser usado para gravar nas portas seriais. Quando uma chamada é feito para o método `uart_init()` fornecido, a função de gravação é definida para gravar em UART1.

Inclui:

- `osapi.h`

Veja também:

- [os\\_printf](#)
- [system\\_set\\_os\\_print](#)

### **os\_random**

`os_random` longo sem sinal ()

Inclui:

- `osapi.h`

### **os\_get\_random**

`int os_get_random (unsigned char * buf, size_t len)`

Inclui:

- `osapi.h`

### **os\_strlen**

Obtenha o comprimento de uma corda.

```
int os_strlen (char * string)
```

Retorna o comprimento da string terminada em nulo.

Inclui:

- `osapi.h`

### **os\_streac**

Concatene duas strings.

```
char * os_streac (char * str1, char * str2)
```

Concatene o sting terminado nulo apontado por str1 com a string apontada por

str2 e armazena o resultado em str1 .

Inclui:

- osapi.h

Página 334

## Página 335

### **os\_strchr**

Inclui:

- osapi.h

### **os\_strcmp**

Compare duas strings.

`int os_strcmp (char * str1, char * str2)`

Compare a string terminada em nulo apontada por str1 com a string terminada em nulo apontado por str2 . Se str1 < str2 então o retorno é <0. Se str1 > str2 então o retorno é > 0, caso contrário, eles são iguais e o retorno é 0.

Inclui:

- osapi.h

### **os\_strcpy**

Copie uma string para outra.

`char * os_strcpy (char * dest, char * src)`

Copie a string terminada em nulo apontada por src para a memória localizada em dest .

Inclui:

- osapi.h

### **os\_strncmp**

Inclui:

- osapi.h

### **os\_strncpy**

Copie uma string para outra, mas seja sensível à quantidade de memória disponível no buffer de destino.

`char * os_strncpy (char * dest, char * source, size_t sizeOfDest)`

Entenda que a string resultante em dest **não** pode ter terminação nula.

Inclui:

- osapi.h

Página 335

**Página 336****os\_sprintf**

```
sprintf(char * buffer, char * format, ...)
```

O formato não é tão rico quanto sprintf() normal em uma biblioteca C. Por exemplo, sem flutuação ou suporte duplo.

Inclui:

- osapi.h

**os strstr**

Inclui:

- osapi.h

**SPI Flash**

O SPI Flash apis nos permite ler, escrever e apagar setores contidos dentro do flash memória. Note que existe um documento específico da Espressif que cobre o SPI O Flash funciona exclusivamente.

**spi\_flash\_get\_id**

Ge as informações de ID do flash SPI

```
uint32 spi_flash_get_id (void)
```

Este é um valor codificado por bits que representa informações sobre o chip flash que está sendo usado em conjunto com o ESP8266.

O esptool também inclui um comando (flash\_id) que pode ser usado para recuperar e exibir as informações do flash.

Inclui:

- spi\_flash.h

Veja também:

- [esptool.py](#)

**spi\_flash\_erase\_sector**

Apague um setor do flash. Cada setor tem 4k de tamanho.

```
SpiFlashOpResult spi_flag_erase_sector (uint16 s)
```

O parâmetro sec é o número do setor (um setor tem 4096 bytes de tamanho).

Inclui:

Página 336

**Página 337**

- spi\_flash.h

Veja também:

**spi\_flash\_read**

Leia os dados do flash

```
SpiFlashOpResult spi_flash_read (uint32 src_addr, uint32 des_addr, tamanho uint32)
```

O parâmetro src\_addr é o endereço em flash que será lido. O des\_addr é o endereço na memória que será escrito. O parâmetro de tamanho é o tamanho dos dados a serem leitura.

Inclui:

- spi\_flash.h

Veja também:

#### **spi\_flash\_set\_read\_func**

```
void spi_flash_set_read_func (user_spi_flash_read lido)
```

Inclui:

- spi\_flash.h

Veja também:

#### **system\_param\_save\_with\_protect**

Economia de memória

```
bool system_param_save_with_protect (uint16 start_sec, void * param, uint16 len)
```

Inclui:

- spi\_flash.h

Veja também:

#### **spi\_flash\_write**

Grave dados no flash

```
SpiFlashOpResult spi_flash_write (uint32 destAddr, uint32 * srcAddr, tamanho uint32)
```

O destAddr é o endereço em flash que deve ser escrito. O srcAddr é a fonte endereço na memória de onde os novos dados devem ser retirados. O parâmetro de tamanho é o tamanho dos dados a serem gravados.

Inclui:

Página 337

---

## Página 338

- spi\_flash.h

Veja também:

#### **system\_param\_load**

Leia os dados salvos com proteção de flash

```
bool system_param_load (uint16 start_sec, uint16 offset, void * param, uint16 len)
```

Inclui:

- spi\_flash.h

Veja também:

## WiFi - ESP8266

```
wifi_fpm_close
wifi_fpm_do_sleep
wifi_fpm_do_wakeup
wifi_fpm_get_sleep_type
wifi_fpm_open
wifi_fpm_set_sleep_type
wifi_fpm_set_wakeup_cb
wifi_get_channel
```

### wifi\_get\_ip\_info

Recupere as informações de IP atuais sobre a estação.

```
bool wifi_get_ip_info (
    uint8 if_index,
    struct ip_info * info)
```

O parâmetro if\_index define a interface a ser recuperada. Dois valores são definidos:

- STATION\_IF - 0 - A interface da estação
- SOFTAP\_IF - 1 - A interface do Soft Access Point

O parâmetro info é preenchido com detalhes do endereço IP atual, máscara de rede e Porta de entrada.

Página 338

## Página 339

Inclui:

- user\_interface.h

Veja também:

- [Endereço IP atual, máscara de rede e gateway](#)
- [struct ip\\_info](#)

### wifi\_get\_macaddr

Obtenha o endereço MAC.

```
bool wifi_get_macaddr (uint8 if_index, uint8 * macaddr)
```

Um endereço MAC tem 6 bytes.

Inclui:

- user\_interface.h

### wifi\_get\_opmode

Obtenha o modo de operação do WiFi

```
uint8 wifi_get_opmode ()
```

Retorne o modo de operação atual do dispositivo.

Existem quatro valores definidos:

- NULL\_MODE - modo nulo. (0)
- STATION\_MODE - Modo estação. (1)
- SOFTAP\_MODE - modo Soft Access Point (AP). (2)
- STATIONAP\_MODE - modo Estação + Ponto de Acesso Soft (AP). (3)

Inclui:

- user\_interface.h

Veja também:

- [Definindo o modo de operação](#)
- [wifi\\_get\\_opmode](#)

### wifi\_get\_opmode\_default

Obtenha o modo de operação padrão

```
uint8 wifi_get_opmode_default()
```

Retorne ao modo de operação padrão do dispositivo após a inicialização.

Existem três valores definidos:

Página 339

## Página 340

- STATION\_MODE - Modo estação
- SOFTAP\_MODE - modo Soft Access Point (AP)
- STATIONAP\_MODE - modo Estação + Ponto de Acesso Soft (AP)

Inclui:

- user\_interface.h

Veja também:

- [Definindo o modo de operação](#)
- [wifi\\_get\\_opmode](#)
- [wifi\\_set\\_opmode](#)
- [wifi\\_set\\_opmode\\_current](#)

### wifi\_get\_phy\_mode

Obtenha o modo WiFi de nível físico.

```
enum phy_mode wifi_get_phys_mode();
```

Isso é usado para recuperar o tipo de rede IEEE 802.11 como ab / g / n.

Inclui:

- user\_interface.h

Veja também:

- [enum phy\\_mode](#)
- 

### wifi\_get\_sleep\_type

Inclui:

- user\_interface.h

### wifi\_get\_user\_fixed\_rate

```
int wifi_get_user_fixed_rate (uint8 * enable_mask, uint8 * taxa)
```

```
wifi_get_user_limit_rate_mask
uint8 wifi_get_user_limit_rate_mask()
```

```
wifi_set_broadcast_if
bool wifi_set_broadcast_if(interface uint8)
```

Inclui:

Página 340

## Página 341

- user\_interface.h

Veja também:

- [Transmitir com UDP](#)

```
wifi_get_broadcast_if
uint8 wifi_get_broadcast_if()
```

Inclui:

- user\_interface.h

Veja também:

- [Transmitir com UDP](#)

**wifi\_set\_sleep\_type**

Inclui:

- user\_interface.h

**wifi\_promiscuous\_enable**

**wifi\_promiscuous\_set\_mac**

**wifi\_register\_rfid\_locp\_recv\_cb**

**wifi\_register\_send\_pkt\_freedom\_cb**

**wifi\_register\_user\_ie\_manufacturer\_recv\_cb**

**wifi\_rfid\_locp\_recv\_close**

**wifi\_rfid\_locp\_recv\_open**

**wifi\_send\_pkt\_freedom**

**wifi\_set\_channel**

**wifi\_set\_event\_handle\_cb**

Defina uma função de retorno de chamada para detectar eventos WiFi.

void wifi\_set\_event\_handler\_cb(wifi\_event\_handler\_cb\_t callbackFunction)

Registra uma função a ser chamada quando um evento é detectado pelo subsistema WiFi.

A assinatura da função de retorno de chamada registrada é:

void (\* functionName) (evento System\_Event\_t \*)

---

**Página 342**

Inclui:

- user\_interface.h

Veja também:

- [Tratamento de eventos WiFi](#)
- [System\\_Event\\_t](#)

### wifi\_set\_ip\_info

Defina os dados da interface para o dispositivo.

```
bool wifi_set_ip_info (uint8 if_index, struct ip_info * info)
```

O parâmetro if\_index define a interface a ser recuperada. Dois valores são definidos:

- STATION\_IF - 0 - A interface da estação
- SOFTAP\_IF - 1 - A interface do Soft Access Point

O parâmetro info é um ponteiro para uma struct ip\_info que contém os valores que desejamos definir.

Inclui:

- user\_interface.h

Veja também:

- [Endereço IP atual, máscara de rede e gateway](#)
- [struct ip\\_info](#)

### wifi\_set\_macaddr

Defina o endereço MAC.

```
bool wifi_set_macaddr (uint8 if_index, uint8 * macaddr)
```

Um endereço MAC tem 6 bytes.

Inclui:

- user\_interface.h

### wifi\_set\_opmode

Defina o modo de operação do WiFi, incluindo salvar no flash.

```
bool wifi_set_opmode (uint8 opmode)
```

Existem três valores definidos:

- STATION\_MODE - Modo estação
- SOFTAP\_MODE - modo Soft Access Point (AP)

- STATIONAP\_MODE - modo Estação + Ponto de Acesso Soft (AP)

Inclui:

- user\_interface.h

Veja também:

- [Definindo o modo de operação](#)
- [wifi\\_get\\_opmode](#)
- [wifi\\_get\\_opmode\\_default](#)

### wifi\_set\_opmode\_current

Defina o modo de operação do WiFi, mas não salve no flash.

```
bool wifi_set_opmode_current (opmode uint8)
```

Existem três valores definidos:

- STATION\_MODE - Modo estação
- SOFTAP\_MODE - modo Soft Access Point (AP)
- STATIONAP\_MODE - modo Estação + Ponto de Acesso Soft (AP)

Inclui:

- user\_interface.h

Veja também:

- [Definindo o modo de operação](#)
- [wifi\\_get\\_opmode](#)
- [wifi\\_get\\_opmode\\_default](#)

### wifi\_set\_phy\_mode

Defina o modo WiFi de nível físico.

```
bool wifi_set_phy_mode (enum phy_mode mode)
```

Isso é usado para definir o tipo de rede IEEE 802.11 como ab / g / n.

Inclui:

- user\_interface.h

Veja também:

- [enum\\_phy\\_mode](#)

### wifi\_set\_promiscuous\_rx\_cb

### wifi\_set\_sleep\_type

### wifi\_set\_user\_fixed\_rate

```
int wifi_set_user_fixed_rate (uint8 enable_mask, taxa uint8)
```

A máscara de ativação pode ser uma das seguintes:

- FIXED\_RATE\_MASK\_NONE
- FIXED\_RATE\_MASK\_STA
- FIXED\_RATE\_MASK\_AP

- FIXED\_RATE\_MASK\_ALL

A taxa pode ser uma das seguintes:

- PHY\_RATE\_6
- PHY\_RATE\_9
- PHY\_RATE\_12
- PHY\_RATE\_18
- PHY\_RATE\_24
- PHY\_RATE\_36
- PHY\_RATE\_48
- PHY\_RATE\_54

`wifi_set_user_ie`

`wifi_set_user_limit_rate_mask`

`bool wifi_set_user_limit_rate_mask (uint8 enable_mask)`

`wifi_set_user_rate_limit`

`bool wifi_set_user_rate_limit (modo uint8, uint8 ifidx, uint8 max, uint8 min)`

`wifi_set_user_sup_rate`

`int wifi_set_user_sup_rate (uint8 min, uint8 max)`

- RATE\_11B5M
- RATE\_11B11M
- RATE\_11B1M
- RATE\_11B2M
- RATE\_11G6M

Página 344

## Página 345

- RATE\_11G12M
- RATE\_11G24M
- RATE\_11G48M
- RATE\_11G54M
- RATE\_11G9M
- RATE\_11G18M
- RATE\_11G36M

`wifi_status_led_install`

Associe um pino GPIO ao LED de status WiFi.

```
void wifi_status_led_install (
    uint8 gpio_id,
    uint32 mux_name,
    uint8 gpio_func)
```

Quando o tráfego de WiFi flui, podemos desejar que um LED de status pisque ou pisque indicando fluxo tráfego. Esta função nos permite especificar um GPIO que deve ser pulsado para indicar WiFi

tráfego.

O parâmetro `gpio_id` é o número do pino numérico.

O `mux_name` é o nome do nome lógico do multiplexador.

O `gpio_func` é a função a ser habilitada para esse multiplexador.

Inclui:

- `user_interface.h`

Veja também:

- [wifi\\_status\\_led\\_uninstall](#)

#### **wifi\_status\_led\_uninstall**

Desassocie um LED de status de um pino GPIO.

```
void wifi_status_led_uninstall()
```

Desassocia uma configuração de associação anterior com uma chamada para `wifi_status_led_install()`.

Inclui:

- `user_interface.h`

Veja também:

- [wifi\\_status\\_led\\_install](#)

## Página 346

#### **wifi\_unregister\_rfid\_locp\_recv\_cb**

#### **wifi\_unregister\_send\_pkt\_freedom\_cb**

#### **wifi\_unregister\_user\_ie\_manufacturer\_recv\_cb**

### **Estação WiFi**

As seguintes APIs estão relacionadas ao dispositivo ESP \* agindo como uma estação e conectando-se a um ponto de acesso externo.

#### **wifi\_station\_ap\_change**

Mude a conexão para outro ponto de acesso

```
bool wifi_station_ap_change (uint newApld)
```

Inclui:

- `user_interface.h`

#### **wifi\_station\_ap\_number\_set**

Número de estações que serão armazenadas em cache

```
bool wifi_station_ap_number_set (uint8 ap_number)
```

Inclui:

- `user_interface.h`

#### **wifi\_station\_connect**

Conekte a estação a um ponto de acesso.

```
bool wifi_station_connect()
```

Se já estivermos conectados a um ponto de acesso diferente, primeiro precisamos desconectar a partir dele usando `wifi_station_disconnect()`. Há também um atributo de conexão automática que pode ser usado para permitir que o dispositivo tente se conectar ao último ponto de acesso visto quando está ligado. Isso pode ser definido com `wifi_station_set_auto_connect()` função.

Inclui:

- `user_interface.h`

Veja também:

- [Conectando a um ponto de acesso](#)
- 

Página 346

## Página 347

### `wifi_station_dhcpc_start`

Inicie o cliente DHCP.

```
bool wifi_station_dhcpc_start()
```

Se o DHCP estiver habilitado, o IP, a máscara de rede e o gateway serão recuperados do DHCP servidor, enquanto se desativado, estaremos usando valores estáticos.

Inclui:

- `user_interface.h`

Veja também:

- [Endereço IP atual, máscara de rede e gateway](#)
- [wifi\\_station\\_dhcpc\\_stop](#)

### `wifi_station_dhcpc_status`

Obtenha o status do cliente DHCP

```
enum dhcp_status wifi_station_dhcpc_status()
```

Um de:

- `DHCP_STOPPED`
- `DHCP_STARTED`

Inclui:

- `user_interface.h`

### `wifi_station_dhcpc_stop`

Pare o cliente DHCP

```
bool wifi_station_dhcpc_stop()
```

Se o DHCP estiver habilitado, o IP, a máscara de rede e o gateway serão recuperados do DHCP servidor, enquanto se desativado, estaremos usando valores estáticos.

Inclui:

- `user_interface.h`

Veja também:

- [Endereço IP atual, máscara de rede e gateway](#)

**wifi\_station\_disconnect**

Desconecte a estação de um ponto de acesso.

```
bool wifi_station_disconnect()
```

Página 347

**Página 348**

Devemos presumir que já nos conectamos por meio de um `wifi_station_connect()`.

Podemos determinar nosso status de conexão atual por meio de

`wifi_station_get_connect_status()`.

O retorno é verdadeiro em caso de sucesso e falso em caso de erro.

Inclui:

- `user_interface.h`

**wifi\_station\_get\_ap\_info**

Obtenha as informações dos pontos de acesso em cache

```
uint8 wifi_station_get_ap_info (struct station_config configs [])
```

Inclui:

- `user_interface.h`

**wifi\_station\_get\_auto\_connect**

Determine se o ESP se conectará ou não automaticamente ao último ponto de acesso na inicialização.

```
uint8 wifi_station_get_auto_connect()
```

Determine se o dispositivo tentará ou não conectar-se automaticamente ao último acesso ponto no reinício. Um valor se 0 significa que não, enquanto não 0 significa que sim.

Inclui:

- `user_interface.h`

Veja também:

- [wifi\\_station\\_set\\_auto\\_connect](#)

**wifi\_station\_get\_config**

Obtenha a configuração atual da estação

```
bool wifi_station_get_config (struct station_config * config)
```

Recupere as configurações atuais da estação.

Inclui:

- `user_interface.h`

Veja também:

- [Configuração da estação](#)
- [station\\_config](#)
- [wifi\\_station\\_set\\_config](#)

Página 348

---

**Página 349**

- [wifi\\_station\\_set\\_config\\_current](#)

**wifi\_station\_get\_config\_default**

Obtenha a configuração padrão da estação

Inclui:

- user\_interface.h

Veja também:

- [Configuração da estação](#)

**wifi\_station\_get\_connect\_status**

Obtenha o status da conexão da estação.

`uint8 wifi_station_get_connect_status()`

O resultado é um enum com os seguintes valores possíveis:

Nome do enum	Valor
STATION_IDLE	0
STATION_CONNECTING	1
STATION_WRONG_PASSWORD	2
STATION_NO_AP_FOUND	3
STATION_CONNECT_FAIL	4
STATION_GOT_IP	5
Não está no modo de estação	255

Inclui:

- user\_interface.h

Veja também:

- [WiFi.printDiag](#)

**wifi\_station\_get\_current\_ap\_id**

Obtenha o ID do ponto de acesso atual

`uint8 wifi_station_get_current_ap_id()`

Inclui:

- user\_interface.h

Página 349

---

**Página 350****wifi\_station\_get\_hostname**

Obtenha o nome de host DHCP do dispositivo WiFi.

`char * wifi_station_get_hostname()`

Inclui:

- user\_interface.h

### wifi\_station\_get\_reconnect\_policy

#### wifi\_station\_get\_rssi

Obtenha a indicação de força do sinal recebido (rssi).

```
sint8 wifi_station_get_rssi()
```

Obtenha a indicação de força do sinal recebido (rssi).

Inclui:

- user\_interface.h

### wifi\_station\_scan

Procure pontos de acesso disponíveis

```
bool wifi_station_scan(
    struct scan_config * config,
    scan_done_cb_t callbackFunction)
```

Podemos escanear as frequências WiFi procurando pontos de acesso. Devemos estar na estação modo para executar o comando. Quando a função é executada, fornecemos um função de retorno de chamada que será invocada de forma assíncrona em algum momento no futuro com o resultados.

A estrutura scan\_config contém:

- uint8 \* ssid
- uint8 \* bssid
- canal uint8
- uint8 show\_hidden

Se fornecermos essa estrutura, apenas os pontos de acesso correspondentes serão retornados.

O parâmetro scan\_config pode ser NULL , caso em que nenhuma filtragem será realizada e todos os pontos de acesso serão devolvidos.

Página 350

## Página 351

O scan\_done\_cb\_t é uma função com a seguinte estrutura:

```
void (* functionName) (void * arg, status STATUS)
```

O parâmetro arg é um ponteiro para um struct bss\_info .

É importante notar que a **primeira** entrada na cadeia deve ser ignorada, pois é a cabeça da lista.

Para obter a próxima entrada, podemos usar STAILQ\_NEXT (pBssInfoVar, próximo) .

O AUTH\_MODE é um enum

Nome do enum	Valor
AUTH_OPEN	0
AUTH_WEP	1

AUTH_WPA_PSK	2
AUTH_WPA2_PSK	3
AUTH_WPA_WPA2_PSK	4

STATUS é um enum contendo:

**Valor do nome Enum**

OK	0
FALHOU	1
PENDENTE	2
OCUPADO	3
CANCELAR	4

Em caso de sucesso, a função retorna verdadeiro e falso em caso de falha.

O nome desta função é peculiar. Dado que parece localizar pontos de acesso e não estações, acredito que um nome mais apropriado seria

wifi\_access\_point\_scan () .

Inclui:

- user\_interface.h

Veja também:

- [Procurando pontos de acesso](#)
- [struct bss\\_info](#)
- [STATUS](#)

### wifi\_station\_set\_auto\_connect

Defina se o ESP se conectará automaticamente ou não ao último ponto de acesso na inicialização.

Página 351

## Página 352

```
bool wifi_station_set_auto_connect (uint8 setValue)
```

Defina se o dispositivo tentará ou não conectar-se automaticamente ao último ponto de acesso em reiniciar. Um valor 0 significa que não, enquanto um valor diferente de 0 significa que sim. Se for chamado user\_init () , a configuração terá efeito imediato. Se for chamado em outro lugar, a configuração terá efeito na próxima reinicialização.

Inclui:

- user\_interface.h

Veja também:

- [wifi\\_station\\_get\\_auto\\_connect](#)

### wifi\_station\_set\_cert\_key

Defina o certificado e a chave privada para se conectar ao ponto de acesso WPA2-Enterprise.

```
bool wifi_station_set_cert_key (
    uint8 * client_cert, int client_cert_len,
    uint8 * private_key, int private_key_len,
    uint8 * private_key_passwd, int private_key_passwd_len)
```

### wifi\_station\_clear\_cert\_key

Libere recursos e limpe o status após conectar a um acesso WPA2-Enterprise apontar.

```
void wifi_station_clear_cert_key (void)
```

### wifi\_station\_set\_config

Defina a configuração da estação.

```
bool wifi_station_set_config (struct station_config * config)
```

Esta função só pode ser chamada quando o modo do dispositivo inclui suporte de estação.

Especificamente, os detalhes de qual ponto de acesso interagir são fornecidos aqui. O os detalhes são mantidos durante a reinicialização do dispositivo.

Um valor de retorno verdadeiro indica sucesso e um valor falso indica falha.

Inclui:

- user\_interface.h

Veja também:

- [Configuração da estação](#)
- [station\\_config](#)

Página 352

## Página 353

- [wifi\\_station\\_get\\_config](#)
- [wifi\\_station\\_get\\_config\\_default](#)

### wifi\_station\_set\_config\_current

Defina a configuração da estação, mas não salve em flash.

```
bool wifi_station_set_config_current (struct station_config * config)
```

Esta função só pode ser chamada quando o modo do dispositivo inclui suporte de estação.

Especificamente, os detalhes de qual ponto de acesso interagir são fornecidos aqui. O os detalhes não são mantidos durante a reinicialização do dispositivo.

Um valor de retorno verdadeiro indica sucesso e um valor falso indica falha.

Inclui:

- user\_interface.h

Veja também:

- [Configuração da estação](#)
- [station\\_config](#)
- [wifi\\_station\\_get\\_config](#)
- [wifi\\_station\\_get\\_config\\_default](#)

### wifi\_station\_set\_reconnect\_policy

O que deve acontecer quando o ESP for desconectado do AP

```
bool wifi_station_set_reconnect_policy (conjunto bool)
```

Inclui:

- user\_interface.h

### wifi\_station\_set\_hostname

Defina o nome de host DHCP do dispositivo WiFi.

```
bool wifi_station_set_hostname (char * name)
```

Inclui:

- user\_interface.h
- 

### **WiFi SoftAP**

As seguintes APIs estão relacionadas ao dispositivo ESP \* agindo como um ponto de acesso para estações.

#### **wifi\_softap\_dhcps\_start**

Inicie o serviço do servidor DHCP.

Página 353

## **Página 354**

```
bool wifi_softap_dhcps_start()
```

Inicie o serviço do servidor DHCP dentro do dispositivo.

Inclui:

- user\_interface.h

Veja também:

- [O servidor DHCP](#)
- [wifi\\_softap\\_dhcps\\_stop](#)
- [wifi\\_softap\\_dhcps\\_offer\\_option](#)

#### **wifi\_softap\_dhcps\_status**

Retorne o status do serviço do servidor DHCP.

```
enum dhcp_status wifi_softap_dhcps_status()
```

Recupere o status do serviço do servidor DHCP. O valor retornado será um dos seguintes:

- DHCP\_STOPPED
- DHCP\_STARTED

Inclui:

- user\_interface.h

Veja também:

- [O servidor DHCP](#)
- [wifi\\_softap\\_dhcps\\_stop](#)
- [wifi\\_softap\\_dhcps\\_offer\\_option](#)

#### **wifi\_softap\_dhcps\_stop**

Pare o serviço do servidor DHCP.

```
bool wifi_softap_dhcps_stop()
```

Pare o serviço do servidor DHCP dentro do dispositivo.

Inclui:

- user\_interface.h

Veja também:

- [O servidor DHCP](#)
- [wifi\\_softap\\_dhcps\\_offer\\_option](#)

#### **wifi\_softap\_free\_station\_info**

Libere os dados associados a um struct station\_info .

```
void wifi_softap_free_station_info()
```

Página 354

## Página 355

Após uma chamada para `wifi_softap_get_station_info()`, podemos ter os dados retornados para nós. Os dados foram alocados pelo SO e devemos retorná-los com esta chamada de função. Observe que esta função **não** leva os dados que foram retornados.

Inclui:

- `user_interface.h`

Veja também:

- [Ser um ponto de acesso](#)
- 

### `wifi_softap_get_config`

Recupere os detalhes da configuração atual do softAP.

```
bool wifi_softap_get_config (struct softap_config * pConfig)
```

Quando chamado, a estrutura `softap_config` apontada como `pConfig` será preenchida com o detalhes da configuração atual do softAP. Os detalhes retornados são aqueles realmente em usar e podem ser diferentes daqueles salvos por padrão.

Um valor de 1 será retornado em caso de sucesso e 0 caso contrário.

Inclui:

- `user_interface.h`

Veja também:

- [struct softap\\_config](#)
- [wifi\\_softap\\_get\\_config\\_default](#)
- [wifi\\_softap\\_set\\_config\\_current](#)

### `wifi_softap_get_config_default`

Recupere os detalhes de configuração padrão do softAP.

```
bool wifi_softap_get_config_default (struct softap_config * config)
```

Quando chamado, a estrutura `softap_config` apontada como `pConfig` será preenchida com o detalhes da configuração padrão do softAP. Os detalhes retornados são aqueles usados na inicialização e podem ser diferentes dos atualmente em uso.

Um valor de 1 será retornado em caso de sucesso e 0 caso contrário.

Inclui:

- `user_interface.h`

Veja também:

- [struct softap\\_config](#)
- [wifi\\_softap\\_set\\_config\\_current](#)

Página 355

## Página 356

**wifi\_softap\_get\_dhcps\_lease****wifi\_softap\_get\_dhcps\_lease\_time**

Obtenha o valor do tempo de concessão do servidor DHCP.

```
uint32 wifi_softap_get_dhcps_lease_time()
```

Retorne o número de minutos que um endereço IP de concessão DHCP do servidor será mantido.

**wifi\_softap\_get\_station\_info**

Retorne os detalhes de todas as estações conectadas.

```
struct station_info * wifi_softap_get_station_info()
```

Os dados de retorno são uma lista vinculada de estruturas de dados struct station\_info .

Inclui:

- user\_interface.h

Veja também:

- [Ser um ponto de acesso](#)
- [wifi\\_softap\\_get\\_station\\_num](#)

**wifi\_softap\_get\_station\_num**

Retorna a contagem de estações atualmente conectadas.

```
uint8 wifi_softap_get_station_num()
```

Retorna o número de estações atualmente conectadas. O número máximo de conexões em um ESP8266 é 4, mas podemos reduzir isso na configuração do softAP se necessário.

Inclui:

- user\_interface.h

Veja também:

- [Ser um ponto de acesso](#)

**wifi\_softap\_reset\_dhcps\_lease\_time**

Redefina o tempo de concessão do servidor DHCP para o valor padrão.

```
bool wifi_softap_reset_dhcps_lease_time()
```

Redefina o tempo de concessão do servidor DHCP para o valor padrão, que atualmente é de 120 minutos.

**wifi\_softap\_set\_config**

Defina a configuração atual e padrão do softAP.

```
bool wifi_softap_set_config (struct softap_config * config)
```

Quando chamado, o struct softap\_config apontado como pConfig será usado como o detalhes da configuração padrão e atual do softAP.

Um valor de 1 será retornado em caso de sucesso e 0 caso contrário.

Inclui:

- `user_interface.h`

Veja também:

- [struct softap\\_config](#)
- [wifi\\_softap\\_get\\_config\\_default](#)
- [wifi\\_softap\\_set\\_config\\_current](#)

### `wifi_softap_set_config_current`

Defina a configuração padrão do softAP.

```
bool wifi_softap_set_config_current (struct softap_config * config)
```

Quando chamado, o struct softap\_config apontado como pConfig será usado como o detalhes da configuração atual do softAP, mas não serão salvos como padrão.

Inclui:

- `user_interface.h`

Veja também:

- [struct softap\\_config](#)
- [wifi\\_softap\\_get\\_config\\_default](#)
- [wifi\\_softap\\_set\\_dhcps\\_lease](#)

### `wifi_softap_set_dhcps_lease`

Defina o intervalo de endereços IP que será concedido por este servidor DHCP.

```
bool wifi_softap_set_dhcps_lease (struct dhcps_lease * pLease)
```

O parâmetro pLease é um ponteiro para uma struct dhcps\_lease que contém um IP intervalo de endereços de endereços IP que serão alugados por este servidor DHCP. A diferença entre o limite superior e inferior dos endereços IP deve ser 100 ou menos. Esta função não terá efeito até que o servidor DHCP seja parado e reiniciado (assumindo que já está a correr).

Inclui:

- `user_interface.h`

Veja também:

Página 357

## Página 358

- [O servidor DHCP](#)
- [wifi\\_softap\\_dhcps\\_stop](#)
- [wifi\\_softap\\_dhcps\\_offer\\_option](#)
- [struct dhcps\\_lease](#)

### `wifi_softap_set_dhcps_lease_time`

Defina o tempo de concessão do servidor DHCP.

```
bool wifi_softap_set_dhcps_lease_time (uint32 minutos)
```

Defina por quanto tempo uma concessão de endereço IP DHCP é válida. o padrão é 120 minutos. O parâmetro é o número de minutos que o aluguel deve ser mantido. Tem um permitido intervalo de 1-2880.

### `wifi_softap_dhcps_offer_option`

Defina as opções do servidor DHCP.

```
bool wifi_softap_set_dhcps_offer_option (nível uint8, void * ativarg)
```

Atualmente, o parâmetro de nível só pode ser OFFER\_ROUTER com argelg sendo uma máscara de bits

com valores:

- 0b0 - Desativa as informações do roteador.
- 0b1 - Habilita as informações do roteador.

Inclui:

- `user_interface.h`

Veja também:

- [wifi\\_softap\\_dhcps\\_stop](#)

## **WPS WiFi**

### **wifi\_wps\_enable**

`bool wifi_wps_enable (WPS_TYPE_t wps_type)`

O parâmetro de tipo pode ser um dos seguintes:

- `WPS_TYPE_DISABLE` - Incompatível
- `WPS_TYPE_PBC` - Configuração de botão de pressão - compatível
- `WPS_TYPE_PIN` - sem suporte
- `WPS_TYPE_DISPLAY` - Sem suporte
- `WPS_TYPE_MAX` - Sem suporte

Página 358

## **Página 359**

Veja também:

- [Configuração protegida de WiFi - WPS](#)

### **wifi\_wps\_disable**

`bool wifi_wps_disable ()`

Veja também:

- [Configuração protegida de WiFi - WPS](#)

### **wifi\_wps\_start**

`bool wifi_wps_start ()`

Veja também:

- [Configuração protegida de WiFi - WPS](#)

### **wifi\_set\_wps\_cb**

`bool wifi_set_wps_cb (retorno de chamada wps_st_cb_t)`

A assinatura da função de retorno de chamada é:

```
void (* functionName) (int status)
```

O parâmetro de status será um dos seguintes:

- `WPS_CB_ST_SUCCESS`
- `WPS_CB_ST_FAILED`
- `WPS_CB_ST_TIMEOUT`

Veja também:

- [Configuração protegida de WiFi - WPS](#)

## APIs de atualização

### system\_upgrade\_flag\_check

Recupere o sinalizador de status de atualização.

```
uint8 system_upgrade_flag_check()
```

O valor retornado será um dos seguintes:

- UPGRADE\_FLAG\_IDLE
- UPGRADE\_FLAG\_START
- UPGRADE\_FLAG\_FINISH

Página 359

## Página 360

### system\_upgrade\_flag\_set

Defina o sinalizador de status de atualização.

```
void system_upgrade_flag_set(sinalizador uint8)
```

A bandeira pode ser uma das seguintes:

- UPGRADE\_FLAG\_IDLE
- UPGRADE\_FLAG\_START
- UPGRADE\_FLAG\_FINISH

### system\_upgrade\_reboot

Reinicialize o ESP8266 e execute o novo firmware.

```
void system_upgrade_reboot()
```

### system\_upgrade\_start

Comece a baixar o novo firmware do servidor.

```
bool system_upgrade_start(struct upgrade_server_info *server)
```

O parâmetro do servidor é uma estrutura ...

### system\_upgrade\_userbin\_check

Determine qual das duas imagens de firmware possíveis pode ser atualizada.

```
uint8 system_upgrade_userbin_check()
```

O resultado será UPGRADE\_FW\_BIN1 ou UPGRADE\_FW\_BIN2 .

## APIs sniffer

### wifi\_promiscuous\_enable

```
void wifi_promiscuous_enable(uint8 promíscuo)
```

### wifi\_promiscuous\_set\_mac

```
void wifi_promiscuous_set_mac(const uint8_t *endereço)
```

```
wifi_promiscuous_rx_cb
void wifi_promiscuous_rx_cb (wifi_promiscuous_cb_t cb)
```

Página 360

## Página 361

wifi\_get\_channel

wifi\_set\_channel

### APIs de configuração inteligente

**smartconfig\_start**

bool smartconfig\_start (sc\_callback\_t cb, log uint8)

**smartconfig\_stop**

bool smartconfig\_stop (void)

### API SNTP

Lida com a solicitação do Simple Network Time Protocol.

**sntp\_setserver**

Defina o endereço de um servidor SNTP.

void sntp\_serverserver (índice de char não assinado, ip\_addr\_t \* addr)

Defina o endereço de um dos três servidores SNTP possíveis a serem usados.

O parâmetro de índice deve ser 0, 1 ou 2 e especifica qual servidor SNTP slots deve ser definido.

O parâmetro addr é o endereço IP do servidor SNTP a ser registrado.

Inclui:

- sntp.h

Veja também:

- [Trabalhando com SNTP](#)

**sntp\_getserver**

Recupere o endereço IP do servidor SNTP.

ip\_addr\_t sntp\_getserver (índice de char não assinado)

Recupere o endereço IP de um servidor SNTP registrado anteriormente.

O parâmetro index é o índice do servidor SNTP a ser recuperado. Pode ser 0, 1 ou 2.

Inclui:

Página 361

---

**Página 362**

- sntp.h

Veja também:

- [Trabalhando com SNTP](#)

**sntp\_setservername**

Defina o nome do host de um servidor SNTP de destino.

```
void sntp_setservername (índice de char não assinado, char * servidor)
```

Especifique um servidor SNTP por seu nome de host.

O parâmetro de índice é o índice de um servidor SNTP a ser definido. Pode ser 0, 1 ou 2

O parâmetro do servidor é uma string terminada em NULL que nomeia o host que é um SNTP servidor.

Veja também:

- [Trabalhando com SNTP](#)

**sntp\_getservername**

Obtenha o nome do host de um servidor SNTP de destino.

```
char * sntp_getservername (índice de char não assinado)
```

Recupere o nome do host de um servidor SNTP específico que foi registrado anteriormente.

O parâmetro de índice é o índice de um servidor SNTP que foi definido anteriormente. Pode ser 0, 1 ou 2.

O retorno desta função é uma string terminada em NULL.

Inclui:

- sntp.h

Veja também:

- [Trabalhando com SNTP](#)

**sntp\_init**

```
void sntp_init ()
```

Iniciaiza as funções SNTP.

Inclui:

- sntp.h

Veja também:

Página 362

---

**Página 363**

- [Trabalhando com SNTP](#)

**sntp\_stop**

`void sntp_stop ()`

Inclui:

- `sntp.h`

Veja também:

- [Trabalhando com SNTP](#)

### **`sntp_get_current_timestamp`**

Obtenha o carimbo de data / hora atual como um valor de 32 bits sem sinal que representa o número de segundos desde 1 de janeiro de 1970 UTC.

`uint32 sntp_get_current_timestamp ()`

Inclui:

- `sntp.h`

Veja também:

- [Trabalhando com SNTP](#)

### **`sntp_get_real_time`**

`char * sntp_get_real_time (long t)`

????

Inclui:

- `sntp.h`

Veja também:

- [Trabalhando com SNTP](#)

### **`sntp_set_timezone`**

Defina o fuso horário local atual.

`bool sntp_set_timezone (fuso horário sint8)`

Invocar esta função declara nosso fuso horário local como um deslocamento assinado em horas do UTC.

Só deve ser chamado quando as funções SNTP não estiverem em execução, por exemplo, após uma chamada para `sntp_stop ()`.

O parâmetro de fuso horário é um fuso horário no intervalo de -11 a 13.

Página 363

## **Página 364**

O valor de retorno é verdadeiro em caso de sucesso e falso em caso contrário.

Inclui:

- `sntp.h`

Veja também:

- [Trabalhando com SNTP](#)

### **`sntp_get_timezone`**

Obtenha o fuso horário atual.

`sint8 sntp_get_timezone ()`

Recupere o valor atual para o fuso horário conforme definido anteriormente com uma chamada para

sntp\_set\_timezone () .

Inclui:

- sntp.h

Veja também:

- [Trabalhando com SNTP](#)

## APIs TCP / UDP genéricas

### **espconn\_delete**

Exclua uma estrutura de bloco de controle.

`sint8 espconn_delete (struct espconn * espconn)`

O dispositivo mantém dados e armazenamento para cada conversa (TCP e UDP). Quando essas conversas terminaram e não vamos mais nos comunicar com o parceiros, podemos indicar que, chamando esta função que irá liberar o interno armazenar. Prevê-se que a falha em fazer isso resultará em vazamentos de memória.

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_ARG - Argumento ilegal

Esta API desfaz o efeito de `espconn_create` ou `espconn_accept` .

Veja também:

- [UDP](#)
- [espconn\\_create](#)
- [espconn\\_accept](#)

### **espconn\_dns\_setserver**

Defina o servidor DNS padrão.

Página 364

## Página 365

`void espconn_dns_setserver (char numdns, ip_addr_t * dnsservers)`

O numdns é o número de servidores DNS fornecidos, que deve ser 1 ou 2. Não mais do que 2 servidores DNS podem ser fornecidos. Esta função não deve ser chamada se DHCP estiver sendo usava.

O parâmetro dnsservers é uma matriz de 1 ou 2 endereços IP.

Veja também:

- [Serviço de Nome](#)

### **espconn\_gethostbyname**

```
err_t espconn_gethostbyname (struct espconn * espconn,
    const char * hostname,
    ip_addr_t * addr,
    dns_found_callback found)
```

Os parâmetros são:

- espconn - é necessário cuidado e compreensão ao examinar este parâmetro. Uma vez que é uma struct `espconn` , imediatamente pensariam que tem algo a ver com comunicações e é de alguma forma usado para controlar o função `espconn_gethostbyname` () . A resposta é muito mais simples. Isto é ignorado. Yup ... a operação de `gethostbyname` () não não dependem desta parâmetro em tudo. No entanto, ele aparece em mais um lugar. Quando o

a função de retorno de chamada é invocada como resultado de ter terminado o gethostbyname ... o parâmetro arg para o retorno de chamada é definido como o valor deste parâmetro espconn . Então, na realidade, talvez fosse melhor definir o tipo de dados deste primeiro parâmetro para ser um " void \* ", pois basicamente é assim que é usado.

- hostname - O nome do host a ser pesquisado.
- addr - O endereço de uma área de armazenamento onde o endereço IP será colocado **apenas** se ele foi consultado recentemente antes e é mantido no cache. O endereço encontrado aqui é válido se ESPCONN\_OK for retornado.
- encontrado - uma função de retorno de chamada que será invocada quando o endereço tiver sido resolvido. O retorno de chamada será invocado apenas se ESPCONN\_INPROGRESS for retornado.

O dns\_found\_callback é uma função com a seguinte assinatura:

```
void (* functionName) (const char * name, ip_addr_t * ipAddr, void * arg)
```

onde o parâmetro arg é um ponteiro para uma struct espconn , o nome é o nome do host sendo procurado e o ipAddr é o endereço do endereço IP usado para armazenar o resultado.

Página 365

## Página 366

Quando um nome de host não pode ser encontrado, o ipAddr é retornado como NULL ... no entanto, seu provedor de DNS pode optar por fornecer um endereço IP de um mecanismo de pesquisa e, portanto, você receba um endereço de volta ... mas não aquele do host que você esperava !!

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_OK - Sucesso .
- ESPCONN\_INPROGRESS - Indica que não temos cache e precisamos olho para cima.
- ESPCONN\_ARG - Argumento ilegal.

Veja também:

- [Serviço de Nome](#)
- lwIP - [DNS](#)

```
espconn_port
uint32 espconn_port ()
```

### espconn\_regist\_sentcb

Registre uma função de retorno de chamada que será chamada quando os dados forem enviados.

```
sint8 espconn_regist_sentcb (
    struct espconn * espconn,
    espconn_sent_callback sent_cb)
```

O formato da função de retorno de chamada é:

```
void (* functionName) (void * arg)
```

O parâmetro arg é um ponteiro para uma estrutura espconn que descreve a conexão.

Veja também:

- [Envio e recebimento de dados TCP](#)
- [struct espconn](#)

**espconn\_regist\_recvcb**

Registre uma função a ser chamada quando os dados se tornarem disponíveis na conexão TCP ou Datagrama UDP.

```
sint8 espconn_regist_recvcb (
    struct espconn * espconn,
    espconn_recv_callback recv_cb)
```

O formato da função de retorno de chamada é:

```
void (* functionName) (void * arg, char * pData, unsigned short len)
```

Página 366

**Página 367**

Onde args é um ponteiro para um struct espconn , pData é um ponteiro para os dados recebidos e len é o comprimento dos dados recebidos.

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_ARG - Argumento ilegal

Veja também:

- [Envio e recebimento de dados TCP](#)
- [UDP](#)
- [espconn\\_create](#)
- [struct espconn](#)

**especonn\_send**

Envie dados através da conexão para o parceiro.

```
sint8 especonn_send (
    struct especonn * pEspconn,
    uint8 * pBuffer,
    comprimento uint16)
```

O parâmetro pEspconn identifica a conexão por meio da qual transmitir os dados.

O parâmetro pBuffer aponta para um buffer de dados a ser transmitido.

O parâmetro de comprimento fornece o comprimento dos dados em bytes que devem ser transmitidos.

Observe que os dados não precisam ser transmitidos imediatamente. Podemos ser notificados quando os dados foram transmitidos por um retorno de chamada para a função registrada com espconn\_regist\_sentcb () .

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_MEM (-1) - Sem memória
- ESPCONN\_ARG (-12) - Argumento ilegal

Veja também:

- [Envio e recebimento de dados TCP](#)
- [UDP](#)
- [especonn\\_regist\\_sentcb](#)

**especonn\_sendto**

```
sin16 especonn_sendto (struct especonn * espconn, uint8 * psent, comprimento uint16)
```

**ipaddr\_addr**

Crie um endereço TCP / IP a partir de uma representação de string decimal com pontos.

```
uint32 ipaddr_addr (char * addressString)
```

Página 367

---

## Página 368

Retorna um valor de endereço IP (4 bytes) de uma representação de string decimal com pontos fornecido no parâmetro `addressString`. Observe que o tipo `uint32` **não pode ser** atribuído a os endereços em uma estrutura `esp_tcp` ou `esp_udp`. Em vez disso, temos que usar um local variável e, em seguida, copie o conteúdo. Por exemplo:

```
uint32 addr = ipaddr_addr (servidor);
memcpy (m_tcp.remote_ip, &addr, 4);
```

### **IP4\_ADDR**

Defina o valor de uma variável para um endereço IP de sua representação decimal.

```
IP4_ADDR (struct ip_addr * addr, a, b, c, d)
```

O parâmetro `addr` é um ponteiro para armazenamento para conter um endereço IP. Isso pode ser um instância de `struct ip_addr`, `a uint32`, `uint8 [4]`. Deve ser convertido em um ponteiro para uma estrutura `ip_addr` se ainda não for desse tipo.

Os parâmetros `a`, `b`, `c` e `d` são as partes de um endereço IP se ele tiver sido escrito em pontos notação decimal.

Inclui:

- `ip_addr.h`

Veja também:

- [struct ip\\_addr](#)

### **IP2STR**

Gere quatro valores `int` usados em uma instrução `os_printf`

```
IP2STR (ip_addr_t * address)
```

Esta é uma macro que leva um ponteiro para um endereço IP e retorna quatro vírgulas valores decimais separados que representam os 4 bytes de um endereço IP. Isso é comumente usado em códigos como:

```
os_printf ("%d.%d.%d.%d\n", IP2STR (&addr));
```

## **APIs TCP**

### **espconn\_abort**

Força o encerramento de uma conexão TCP / IP.

```
sint8 espconn_abort (struct espconn * espconn)
```

Esta API **não** deve ser chamada em nenhuma função de retorno de chamada `espconn`.

Um código de retorno de 0 indica sucesso.

Página 368

---

## Página 369

**espconn\_accept**

Ouça uma conexão TCP de entrada.

```
sint8 espconn_accept (struct espconn * espconn)
```

Depois de chamar esta função, o ESP8266 começa a escutar as conexões de entrada. Algumas funções de retorno de chamada registradas com `espconn_register_connectcb()` serão invocadas quando chegam novas conexões.

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_MEM - Sem memória
- ESPCONN\_ISCONN - Já conectado
- ESPCONN\_ARG - Argumento ilegal

Nota: depois de pensar um pouco, acho que realmente não gosto do nome disso. Que função esta faz é fazer com que o ESP8266 comece a escutar em uma porta local para novas solicitações de entrada. Essencialmente, tornando o ESP8266 um servidor. Quando estudamos a API de sockets, descobrimos que a chamada de função equivalente para realizar essa tarefa é chamada de escuta . Então meu O novo nome sugerido / recomendado para esta função seria `espconn_listen` .

Então, de onde veio o nome de aceitação ? A resposta é que na API de sockets lá é uma função de parceiro chamada aceitar . Quando executado contra um soquete que anteriormente tinha escuta chamado contra ela, o que ele faz é bloco até um parceiro tenta realmente conectar. No ESP8266, não há equivalente. Em vez disso, após `espconn_accept` ser chamado, o ESP8266 começa a ouvir imediatamente e quando um parceiro se conecta, acordamos o retorno de chamada de conexão. Então ... `espconn_accept` é uma chamada de `listen()` de sockets ou um sockets `aceitar()` ligar? Minha mente diz que está MUITO mais perto de uma chamada `listen()` .

Veja também:

- [TCP](#)
- [espconn\\_register\\_connectcb](#)
- [espconn\\_delete](#)

**espconn\_get\_connection\_info**

```
sint8 espconn_get_connection_info (
    struct espconn * espconn,
    remot_info ** pcon_info,
    uint8 typeFlags)
```

O espconn é um ponteiro para o bloco de controle TCP.

O parâmetro pcon\_info é a informação do parceiro.

O typeFlags define sobre o tipo de parceiro sobre o qual estamos obtendo informações:

Página 369

---

**Página 370**

- 0 - parceiro regular
- 1 - parceiro SSL

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_ARG - Argumento ilegal

**espconn\_connect**

Conecte-se a um aplicativo remoto usando TCP.

```
sint8 espconn_connect (struct espconn * espconn)
```

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_RTE (-4) - Problema de roteamento
- ESPCONN\_MEM (-1) - Sem memória
- ESPCONN\_ISCONN (-15) - Já conectado
- ESPCONN\_ARG (-12) - Argumento ilegal

Perceba que, depois de fazer esta chamada, ainda podemos falhar na conexão. Este é um assíncrono chamada que será realizada posteriormente. Se houver uma falha nesse ponto, encontraremos que o retorno de chamada registrado com espconn\_regist\_reconcb () será invocado.

Quando a conexão for estabelecida, qualquer retorno de chamada registrado feito com espconn\_regist\_connect () será invocado.

Veja também:

- [TCP](#)
- [espconn\\_disconnect](#)
- [espconn\\_regist\\_connectcb](#)
- [espconn\\_regist\\_disconcb](#)
- [espconn\\_regist\\_reconcb](#)

**espconn\_disconnect**

Desconecte uma conexão TCP.

```
sint8 espconn_disconnect (struct espconn * espconn)
```

Desconecte uma conexão TCP que foi formada anteriormente com espconn\_connect () ou espconn\_accept () . Quando a desconexão for bem-sucedida, veremos um retorno de chamada para o função registrada com espconn\_regist\_disconcb () .

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_ARG - Argumento ilegal

Veja também:

Página 370

**Página 371**

- [TCP](#)
- [espconn\\_accept](#)
- [espconn\\_connect](#)
- [espconn\\_regist\\_disconcb](#)

**espconn\_regist\_connectcb**

Registre uma função que será chamada quando uma conexão TCP for formada.

```
sint8 espconn_regist_connectcb (
    struct espconn * espconn,
    espconn_connect_callback connect_cb)
```

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_ARG - Argumento ilegal

A função de retorno de chamada deve ter a seguinte assinatura:

```
void (* functionName) (void * arg)
```

Onde o parâmetro arg é um ponteiro para uma instância de struct espconn .

Pergunta: Esta é uma NOVA struct espconn ou a original?

Veja também:

- [Arquitetura espconn](#)
- [espconn\\_accept](#)
- [espconn\\_connect](#)

### **espconn\_regist\_disconcb**

Registre uma função que será chamada de volta após uma desconexão do TCP.

```
sint8 espconn_regist_disconcb (
    struct espconn * espconn,
    espconn_connect_callback discon_cb)
```

A assinatura da função de retorno de chamada de desconexão é a mesma do retorno de chamada de conexão:

```
void (* functionName) (void * arg)
```

Onde arg é um ponteiro de struct espconn .

Veja também:

- [TCP](#)
- [Arquitetura espconn](#)
- [espconn\\_accept](#)
- [espconn\\_connect](#)
- [espconn\\_disconnect](#)

### **espconn\_regist\_reconcb**

Registre uma função que será chamada quando um erro for detectado.

Página 371

## **Página 372**

```
sint8 espconn_regist_reconcb (
    struct espconn * espconn,
    espconn_reconnect_callback recon_cb)
```

Este retorno de chamada é invocado quando um erro é detectado. Por exemplo, ao tentar conectar-se a um parceiro que não está ouvindo. É provável que o nome desta função fosse simplesmente mal escolhido. Ver:

<http://bbs.espressif.com/viewtopic.php?f=66&t=1063>

A assinatura da função de retorno de chamada é:

```
void (* functionName) (void * arg, sint8 err)
```

O parâmetro arg é um ponteiro para um struct espconn .

O parâmetro err é um dos seguintes:

- ESPCONN\_TIMEOUT (-3)
- ESPCONN\_ABRT (-8)
- ESPCONN\_RST (-9)
- ESPCONN\_CLSD (-10)
- ESPCONN\_CONN (-11) - Falha ao conectar a um parceiro
- ESPCONN\_HANDSHAKE (-28)
- ESPCONN\_PROTO\_MSG ??

Pergunta: O que significa para o status da conexão receber uma indicação de erro?

Devemos então tentar desconectar ou já estamos desconectados? Ver:  
<http://www.esp8266.com/viewtopic.php?f=9&t=5864>

Veja também:

- [A arquitetura espconn](#)
- [TCP](#)
- [espconn\\_accept](#)
- [espconn\\_connect](#)
- [struct espconn](#)

### **espconn\_register\_write\_finish**

Registre uma função de retorno de chamada a ser invocada quando os dados forem transmitidos com sucesso para o parceiro.

```
sint8 espconn_register_write_finish (struct espconn * espconn,
                                     espconn_connect_callback write_finish_cb);
```

Página 372

## **Página 373**

A assinatura do retorno de chamada é:

```
void (* functionName) (void * arg)
```

O parâmetro arg é um ponteiro para um struct espconn .

Veja também:

- [A arquitetura espconn](#)
- [espconn\\_send](#)

### **espconn\_set\_opt**

Defina quais opções ativar para uma conexão.

```
sint8 espconn_set_opt (
    struct espconn * espconn,
    uint8 opt)
```

Esta função deve ser chamada em um espconn\_connect\_callback . O espconn parâmetro é o bloco de controle para a conexão que deve ser modificada.

O parâmetro opt é uma codificação de bits de sinalizadores que devem ser ativados. O parâmetro opt é um enum do tipo espconn\_option :

Nome Enum	Valor
ESPCONN_REUSEADDR	0x01
ESPCONN_NODELAY	0x02
ESPCONN_COPY	0x04
ESPCONN_KEEPALIVE	0x08

Os bits que não estão ativados permanecem inalterados em relação aos seus valores atuais existentes.

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_ARG - Argumento ilegal

Veja também:

- [espconn\\_clear\\_opt](#)
- [espconn\\_set\\_keepalive](#)
- [espconn\\_get\\_keepalive](#)

**espconn\_clear\_opt**

Defina quais opções desligar para uma conexão.

```
sint8 espconn_clear_opt (
    struct espconn * espconn,
    uint8 opt)
```

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

Página 373

**Página 374**

- ESPCONN\_ARG - Argumento ilegal

O valor opt é um enum do tipo espconn\_option :

Nome Enum	Valor
ESPCONN_REUSEADDR	0x01
ESPCONN_NODELAY	0x02
ESPCONN_COPY	0x04
ESPCONN_KEEPALIVE	0x08

Veja também:

- [Tratamento de erros TCP](#)
- [espconn\\_set\\_opt](#)
- [espconn\\_set\\_keepalive](#)
- [espconn\\_get\\_keepalive](#)

**espconn\_regist\_time**

Defina um valor de tempo limite de conexão inativa.

```
sint8 espconn_regist_time (
    struct espconn * espconn,
    intervalo uint32,
    uint8 typeFlag)
```

Se uma conexão ficar inativa por um período de tempo, o ESP8266 é configurado para automaticamente feche a conexão. Parece que o padrão é 10 segundos.

O parâmetro espconn descreve a conexão que deve ter seu tempo limite alterado.

O parâmetro de intervalo define o intervalo de tempo limite em segundos. O valor máximo é 7200 segundos (2 horas).

O parâmetro typeFlag pode ser 0 para indicar que todas as conexões devem ser alteradas ou 1 para definir apenas esta conexão.

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_ARG - Argumento ilegal

Veja também:

- [TCP](#)

**espconn\_set\_keepalive**

```
sint8 espconn_set_keepalive (struct espconn * espconn, nível uint8, void * optArg)
```

---

**Página 375**

**espconn\_get\_keepalive**  
**sint8 espconn\_get\_keepalive (struct espconn \* espconn, nível uint8, void \* optArg)**  
 ???

**espconn\_secure\_accept**  
 Ouça uma conexão SSL TCP de entrada  
**sint8 espconn\_secure\_accept (struct espconn \* espconn)**

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_MEM - Sem memória
- ESPCONN\_ISCONN - Já conectado
- ESPCONN\_ARG - Argumento ilegal

**espconn\_secure\_ca\_disable**  
**bool espconn\_secure\_ca\_disable (nível uint8)**

**espconn\_secure\_ca\_enable**  
**bool espconn\_secure\_ca\_enable (nível uint8, uint16 flash\_sector)**

**espconn\_secure\_set\_size**

**espconn\_secure\_get\_size**

**espconn\_secure\_delete**

Exclua uma conexão SSL ao executar como um servidor baseado em SSL.

**sint8 espconn\_secure\_delete (struct espconn \* espconn)**

Um código de retorno de 0 indica sucesso.

**espconn\_secure\_connect**

Forme uma conexão SSL com um parceiro.

**sint8 espconn\_secure\_connect (struct espconn \* espconn)**

Forme uma conexão SSL com um parceiro.

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_MEM - Sem memória

Página 375

---

**Página 376**

- ESPCONN\_ISCONN - Já conectado

- **ESPCONN\_ARG** - Argumento ilegal

### **especonn\_secure\_send**

Envie dados por meio de uma conexão segura.

`sint8 especonn_secure_send (struct especonn * especonn, uint8 * pBuf, comprimento uint16)`

Envie dados por meio de uma conexão segura.

### **especonn\_secure\_disconnect**

Desconexão TCP segura.

`sint8 especonn_secure_disconnect (struct especonn * especonn)`

Desconexão TCP segura.

Não chame essa função de dentro de uma função de retorno de chamada do ESP.

### **espconn\_tcp\_get\_max\_con**

Retorne o número máximo de conexões TCP simultâneas.

`uint8 espconn_tcp_get_max_con ()`

### **espconn\_tcp\_set\_max\_con**

Defina o número máximo de conexões TCP simultâneas

`sint8 espconn_tcp_set_max_con (uint8 num)`

### **espconn\_tcp\_get\_max\_con\_allow**

Obtenha o número máximo de clientes TCP permitidos para conexão de entrada.

### **espconn\_tcp\_set\_max\_con\_allow**

Defina o número máximo de clientes TCP permitidos para conexão de entrada.

### **espconn\_recv\_hold**

Suspenda o recebimento de dados TCP.

`sint8 espconn_recv_hold (struct especonn * especonn)`

Página 376

## **Página 377**

Suspenda o recebimento de novos dados sobre TCP. Para retomar o recebimento de dados, pode-se usar o chamada de função `espconn_recv_unhold`.

- [espconn\\_recv\\_unhold](#)

### **espconn\_recv\_unhold**

Desbloqueie o recebimento de dados TCP.

`sint8 espconn_recv_unhold (struct especonn * especonn)`

Retome o recebimento de novos dados por TCP. Este método deve ser usado em conjunto com `espconn_recv_hold` que suspende o recebimento de dados.

Veja também:

- [espconn\\_recv\\_hold](#)

## APIs UDP

### espconn\_create

Crie um bloco de controle UDP em preparação para o envio de datagramas.

```
sint8 espconn_create (struct espconn * espconn)
```

Código de retorno 0 em caso de sucesso, caso contrário, o código indica o erro:

- ESPCONN\_ARG - Argumento ilegal
- ESPCONN\_ISCONN - Já conectado
- ESPCONN\_MEM - Sem memória

Veja também:

- [UDP](#)
- [espconn\\_regist\\_sentcb](#)
- [espconn\\_regist\\_recvcb](#)
- [espconn\\_send](#)
- [espconn\\_delete](#)
- [espconn\\_connect](#)

### especonn\_igmp\_join

Junte-se a um grupo multicast.

### especonn\_igmp\_leave

Saia de um grupo multicast.

Página 377

## Página 378

## APIs de ping

### ping\_start

```
bool ping_start (struct ping_option * ping_opt)
```

Inclui:

- ping.h

Veja também:

- [Pedido de ping](#)
- [struct ping\\_option](#)

### ping\_regist\_recv

```
bool ping_regist_recv (struct ping_option * ping_opt, ping_recv_function ping_recv)
```

Registre uma função que será chamada quando um ping for recebido. A assinatura da função é:

```
void (* functionName) (void * pingOpt, void * pingResp)
```

Os parâmetros passados são pingOpt que é um ponteiro para a opção struct ping e pingResp, que é um ponteiro para uma estrutura ping\_resp .

Inclui:

- ping.h

Veja também:

- [Pedido de ping](#)
- [struct ping\\_option](#)
- [struct ping\\_resp](#)

#### **ping\_regist\_sent**

bool ping\_regist\_sent (struct ping\_option \* ping\_opt, ping\_sent\_function ping\_sent)

Registre uma função que será chamada quando um ping for enviado. A assinatura da função é:

void (\* functionName) (void \* pingOpt, void \* pingResp)

Os parâmetros passados são pingOpt que é um ponteiro para a opção struct ping e pingResp, que é um ponteiro para uma estrutura ping\_resp .

Inclui:

- ping.h

Veja também:

- [Pedido de ping](#)
- [struct ping\\_option](#)

Página 378

## **Página 379**

### **APIs mDNS**

Veja também:

- [Sistemas de nomes de domínio multicast](#)

#### **espconn\_mdns\_init**

Inicialize o mDNS no ESP8266.

void espconn\_mdns\_init (struct mdns\_info \* info)

A estrutura do tipo struct mdns\_info contém informações vitais de inicialização e deve ser concluído antes de chamar esta função.

Veja também:

- [struct mdns\\_info](#)

#### **espconn\_mdns\_close**

Feche o suporte mDNS.

void espconn\_mdns\_close ()

Feche o suporte mDNS. Isso pode ser usado após uma chamada para espconn\_mdns\_init () .

Veja também:

- [espconn\\_mdns\\_init](#)

#### **espconn\_mdns\_server\_register**

Registre o servidor mDNS.

void espconn\_mdns\_server\_register ()

#### **espconn\_mdns\_server\_unregister**

Cancele o registro do servidor mDNS.

```
void espconn_mdns_server_unregister()
```

#### **espconn\_mdns\_get\_servername**

Obtenha o nome do servidor mDNS.

```
char * espconn_mdns_get_servername()
```

#### **espconn\_mdns\_set\_servername**

Defina o nome do servidor mDNS.

Página 379

## Página 380

```
char * espconn_mdns_set_servername()
```

#### **espconn\_mdns\_set\_hostname**

Defina o nome do host mDNS.

```
void espconn_mdns_set_hostname(char * name)
```

#### **espconn\_mdns\_get\_hostname**

Obtenha o nome do host mDNS

```
char * espconn_mdns_get_hostname()
```

#### **espconn\_mdns\_disable**

Desative o mDNS.

```
void espconn_mdns_disable()
```

Veja também:

- [espconn\\_mdns\\_enable](#)

#### **espconn\_mdns\_enable**

Habilitar mDNS

```
void espconn_mdns_enable()
```

Veja também:

- [espconn\\_mdns\\_disable](#)

## **GPIO - ESP32**

As funções gpio no ESP32 são fornecidas por meio do ESP-IDF. Deve-se incluir o cabeçalho "driver / gpio.h".

#### **gpio\_config**

```
esp_err_t gpio_config(gpio_config_t * pGPIOConfig)
```

A estrutura de dados gpio\_config\_t contém:

**Página 381**

```
uint64_t pin_bit_mask
modo gpio_mode_t
gpio_pullup_t pull_up_en
gpio_pulldown_t pull_down_en
gpio_int_type_t intr_type
```

O `pin_bit_mask` define quais pinos estamos configurando. Constantes são definidas para ajudar-nos aqui. Por exemplo, se estivermos configurando GPIO34 e GPIO16, podemos definir o `pin_bit_mask` para `GPIO_Pin_16 | GPIO_Pin_34` que é o booleano "ou" dos dois valores constantes.

O modo é usado para definir o modo de todos os pinos que estamos configurando. Os permitidos os valores são:

- `GPIO_MODE_INPUT`
- `GPIO_MODE_OUTPUT`
- `GPIO_MODE_OUTPUT_OD`
- `GPIO_MODE_INPUT_OUTPUT_OD`
- `GPIO_MODE_INPUT_OUTPUT`

O `pull_up_en` habilita um resistor pull-up interno. Os valores permitidos são:

- `GPIO_PULLUP_ENABLE`
- `GPIO_PULLUP_DISABLE`

O `pull_down_en` habilita um resistor pull-down interno. Os valores permitidos são:

- `GPIO_PULLDOWN_ENABLE`
- `GPIO_PULLDOWN_DISABLE`

O `intr_type` configura como as interrupções são tratadas para o pino. Os valores permitidos estão:

- `GPIO_INTR_DISABLE`
- `GPIO_INTR_POSEDGE`
- `GPIO_INTR_NEGEDGE`
- `GPIO_INTR_ANYEDGE`
- `GPIO_INTR_LOW_LEVEL`
- `GPIO_INTR_HIGH_LEVEL`

**Página 382**

**gpio\_get\_level**

Recupere o nível do sinal no pino.

```
int gpio_get_level(gpio_num_t gpioNum)
```

Obtenha o nível do sinal no pino especificado. 0 ou 1.

**gpio\_input\_get**

Recupere uma máscara de bits dos valores dos primeiros 32 GPIOs (0-31).

```
uint32_t gpio_input_get()
```

**gpio\_input\_get\_high**

Recupere uma máscara de bits dos valores dos últimos 10 GPIOs (32-39).

```
uint32_t gpio_input_get_high()
```

**gpio\_intr\_enable**

Habilite interrupções no pino especificado.

```
esp_err_t gpio_intr_enable(gpio_num_t gpioNum)
```

**gpio\_intr\_disable**

Desative as interrupções no pino especificado.

```
esp_err_t gpio_intr_disable(gpio_num_t gpioNum)
```

**gpio\_isr\_register**

Registre um manipulador de interrupção.

```
esp_err_t gpio_isr_register(uint32_t gpioIntr, void (*fn)(void *), void * arg)
```

**gpio\_output\_set**

Defina GPIOs que precisam ser de entrada, saída, alto ou baixo em massa

```
void gpio_output_set(
    uint32_t setMask,
    uint32_t clearMask,
```

```
uint32_t enableMask,
uint32_t disableMask)
```

Trabalhe com os primeiros 32 GPIOs (0-31) especificando quais são de entrada, quais são saída, quais devem ser deixados sozinhos e quais devem ser definidos como alto / baixo. Esta API nos permite definir um lote de GPIOs em uma operação.

### gpio\_output\_set\_high

Defina GPIOs que precisam ser de entrada, saída, alto ou baixo em massa

```
void gpio_output_set_high (
    uint32_t setMask,
    uint32_t clearMask,
    uint32_t enableMask,
    uint32_t disableMask)
```

Trabalhe com os últimos 10 GPIOs (32-39) especificando quais são de entrada, quais são saída, quais devem ser deixados sozinhos e quais devem ser definidos como alto / baixo. Esta API nos permite definir um lote de GPIOs em uma operação.

### gpio\_set\_direction

Defina a direção de um alfinete.

```
esp_err_t gpio_set_direction (gpio_num_t gpioNum, modo gpio_mode_t)
```

O modo é usado para definir o modo do pino que estamos configurando. Os valores permitidos estão:

- GPIO\_MODE\_INPUT
- GPIO\_MODE\_OUTPUT
- GPIO\_MODE\_OUTPUT\_OD
- GPIO\_MODE\_INPUT\_OUTPUT\_OD
- GPIO\_MODE\_INPUT\_OUTPUT

Página 383

## Página 384

### gpio\_set\_intr\_type

Defina o tipo de interrupção de um pino.

```
esp_err_t gpio_set_intr_type (gpio_num_t gpioNum, gpio_int_type_t intrType)
```

O intr\_type configura como as interrupções são tratadas para o pino. Os valores permitidos estão:

- GPIO\_INTR\_DISABLE
- GPIO\_INTR\_POSEDGE
- GPIO\_INTR\_NEGEDGE
- GPIO\_INTR\_ANYEDGE
- GPIO\_INTR\_LOW\_LEVEL
- GPIO\_INTR\_HIGH\_LEVEL

### gpio\_set\_level

Defina o nível de um alfinete.

```
esp_err_t gpio_set_level(gpio_num_t gpioNum, nível uint32_t)
```

Veja o nível de um pino de saída. O nível deve ser 0 ou 1.

### gpio\_set\_pull\_mode

Defina o modo pullup / pullup do pino.

```
esp_err_t gpio_set_pull_mode(gpio_num_t gpioNum, gpio_pull_mode_t pull)
```

Os valores permitidos para pull são:

- GPIO\_PULLUP\_ONLY
- GPIO\_PULLDOWN\_ONLY
- GPIO\_PULLUP\_PULLDOWN
- GPIO\_FLOATING

Página 384

## Página 385

### GPIO - ESP8266

Os nomes dos pinos são:

- PERIPHS\_IO\_MUX\_GPIO0\_U
- PERIPHS\_IO\_MUX\_GPIO2\_U
- PERIPHS\_IO\_MUX\_MTDI\_U
- PERIPHS\_IO\_MUX\_MTCK\_U // GPIO 13
- PERIPHS\_IO\_MUX\_MTMS\_U // GPIO 14

Nome do Pin	Função 1	Função 2	Função 3	Função 4	Alfinete físico
MTDI_U	MTDI	I2SI_DATA	HSPIQ MISO	GPIO12	10
MTCK_U	MTCK	I2SI_BCK	HSPIID MOSI	GPIO13	12
MTMS_U	MTMS	I2SI_WS	HSPICLK	GPIO14	9
MTDO_U	MTDO	I2SO_BCK	HSPICS	GPIO15	13
U0RXD_U	U0RXD	I2SO_DATA		GPIO3	25
U0TXD_U	U0TXD	SPICS1		GPIO1	26
SD_CLK_U	SD_CLK	SPICLK		GPIO6	21
SD_DATA0_U	SD_DATA0	SPIQ		GPIO7	22
SD_DATA1_U	SD_DATA1	SPIID		GPIO8	23
SD_DATA2_U	SD_DATA2	SPIHD		GPIO9	18
SD_DATA3_U	SD_DATA3	SPIWP		GPIO10	19
SD_CMD_U	SD_CMD	SPICS0		GPIO11	20
GPIO0_U	GPIO0	SPICS2			15

GPIO2_U	GPIO2	I2SO_WS	U1TXD	14
GPIO4_U	GPIO4	CLK_XTAL		16
GPIO5_U	GPIO5	CLK_RTC		24

As funções do PIN são:

- FUNC\_GPIO0
- FUNC\_GPIO12
- FUNC\_GPIO13
- FUNC\_GPIO14
- FUNC\_GPIO15
- FUNC\_U0RTS
- FUNC\_GPIO3

Página 385

## Página 386

- FUNC\_U0TXD
- FUNC\_GPIO1
- FUNC\_SDCLK
- FUNC\_SPICLK
- FUNC\_SDDATA0
- FUNC\_SPIQ
- FUNC\_U1TXD
- FUNC\_SDDATA1
- FUNC\_SPID
- FUNC\_U1RXD
- FUNC\_SDATA1\_U1RXD
- FUNC\_SDDATA2
- FUNC\_SPIHD
- FUNC\_GPIO9
- FUNC\_SDDATA3
- FUNC\_SPIWP
- FUNC\_GPIO10
- FUNC\_SDCMD
- FUNC\_SPICS0
- FUNC\_GPIO0
- FUNC\_GPIO2
- FUNC\_U1TXD\_BK
- FUNC\_U0TXD\_BK
- FUNC\_GPIO4

- FUNC\_GPIO5
- LED\_GPIO\_FUNC

**PIN\_PULLUP\_DIS**

Desativar pin pull-up

**PIN\_PULLUP\_DIS (PIN\_NAME)**

Página 386

**Página 387**

Veja também:

- [GPIOs](#)

**PIN\_PULLUP\_EN**

Habilitar pin pull-up

**PIN\_PULLUP\_EN (PIN\_NAME)**

Veja também:

- [GPIOs](#)

**PIN\_FUNC\_SELECT**

Defina a função de um pino específico.

**PIN\_FUNC\_SELECT (PIN\_NAME, FUNC)**

Veja também:

- [GPIOs](#)

**GPIO\_ID\_PIN**

Obtenha a identificação de um pino lógico.

**GPIO\_ID\_PIN (pinNum)**

Converte um número de pin lógico na identidade de um pin. Esta é uma função interessante, pois GPIO\_ID\_PIN (x) é codificado para ser igual a " x ". A questão agora é se um ainda precisa codificar GPIO\_ID\_PIN () ao acessar as funções GPIO.

**GPIO\_OUTPUT\_SET**

Defina o valor de saída de um pino específico.

**GPIO\_OUTPUT\_SET (GPIO\_NUMBER, valor)**

Esta é uma macro auxiliar que invoca gpio\_output\_set () . Tome cuidado ao passar em um valor que faz parte de uma expressão como pData == '1' . O valor é avaliado por número de vezes, portanto, não deve ter efeitos colaterais. Também há um bug atual relacionado a precedência do operador ... é altamente recomendável colocar o valor em extra parênteses durante a codificação. Por exemplo:

**GPIO\_OUTPUT\_SET (GPIO\_NUMBER, (pData == '1'))**

Inclui:

- gpio.h

---

**Página 388**

Veja também:

- [GPIOs](#)

### **GPIO\_DIS\_OUTPUT**

Defina o pino a ser inserido (saída desativada).

GPIO\_DIS\_OUTPUT (GPIO\_NUMBER)

Esta é uma macro auxiliar que invoca `gpio_output_set()`.

Inclui:

- `gpio.h`

Veja também:

- [GPIOs](#)

### **GPIO\_INPUT\_GET**

Leia o valor do pino.

GPIO\_INPUT\_GET (GPIO\_NUMBER)

Esta é uma macro auxiliar que invoca `gpio_input_get()`.

Inclui:

- `gpio.h`

Veja também:

- [gpio\\_input\\_get](#)

### **gpio\_output\_set**

Altere os valores dos pinos GPIO em uma operação.

```
void gpio_output_set (
    uint32 set_mask,
    uint32 clear_mask,
    uint32 enable_output,
    uint32 enable_input)
```

Os parâmetros são:

- `set_mask` - Bits com "1" são configurados para alto, bits com "0" são deixados inalterados.
- `clear_mask` - Bits com "1" são definidos como baixos, bits com "0" são deixados inalterados
- `enable_output` - Bits com "1" são definidos para saída
- `enable_input` - Bits com "1" são definidos para entrada

Inclui:

Página 388

---

**Página 389**

- `gpio.h`

Veja também:

- [GPIOs](#)

### **gpio\_input\_get**

Obtenha os valores dos GPIOs.

```
uint32 gpio_input_get()
```

Recupere os valores dos GPIOs e retorne uma máscara de bits de seus valores.

Inclui:

- gpio.h

Veja também:

- [GPIOs](#)

### **gpio\_intr\_handler\_register**

Registre uma função de retorno de chamada que será invocada quando ocorrer uma interrupção GPIO.

```
void gpio_intr_handler_register (
    gpio_intr_handler_fn_t callbackFunction,
    void * arg)
```

A assinatura da função de manipulador deve ser:

```
void (* functionName) (uint32 interruptMask, void * arg)
```

Inclui:

- gpio.h

Veja também:

- [Tratamento de interrupção GPIO](#)

### **gpio\_pin\_intr\_state\_set**

```
void gpio_pin_intr_state_set (
    uint32 pinId,
    GPIO_INT_TYPE intr_state)
```

O pinId é o valor de ID do pino GPIO retornado de GPIO\_ID\_PIN (num) .

O parâmetro intr\_state define o que dispara a interrupção.

Inclui:

- gpio.h

Veja também:

Página 389

## Página 390

- [Tratamento de interrupção GPIO](#)
- [GPIOs](#)
- [GPIO\\_INT\\_TYPE](#)

### **gpio\_intr\_pending**

Obtenha o conjunto de interrupções pendentes

```
uint32 gpio_intr_pending()
```

Inclui:

- gpio.h

Veja também:

- [Tratamento de interrupção GPIO](#)

### **gpio\_intr\_ack**

Sinalize um conjunto de interrupções como tratadas. Deve ser chamado de uma interrupção função de manipulador.

```
void gpio_intr_ack (uint32 ack_mask)
```

Inclui:

- gpio.h

### **gpio\_pin\_wakeup\_enable**

Defina que o dispositivo pode despertar do modo de suspensão leve quando ocorrer uma interrupção de E / S.

```
void gpio_pin_wakeup_enable (
    pin uint32,
    GPIO_INT_TYPE intr_state)
```

O parâmetro pin define o número do pin usado para despertar o dispositivo.

O intr\_state define qual tipo de transição despertará o dispositivo. As opções são:

- GPIO\_PIN\_INTR\_LOLEVEL
- GPIO\_PIN\_INTR\_HILEVEL

Inclui:

- gpio.h

Veja também:

- [GPIOs](#)
- [GPIO\\_INT\\_TYPE](#)

Página 390

## **Página 391**

### **gpio\_pin\_wakeup\_disable**

```
void gpio_pin_wakeup_disable ()
```

Inclui:

- gpio.h

## **APIs UART**

Essas funções devem ser compiladas a partir dos arquivos uart em driver\_lib .

### **UART\_CheckOutputFinished**

```
bool UART_CheckOutputFinished (uint8 uart_no, uint32 time_out_us)
```

### **UART\_ClearIntrStatus**

```
void UART_ClearIntrStatus (uint8 uart_no, uint32 clr_mask);
```

### **UART\_ResetFifo**

```
void UART_ResetFifo (uint8 uart_no);
```

### **UART\_SetBaudrate**

Defina a taxa de transmissão.

```
void UART_SetBaudrate (uint8 uart_no, uint32 baud_rate)
```

Defina a taxa de transmissão usada pelo UART. O `uart_no` identifica o UART a ser definido (0 ou 1) e `baud_rate` é a taxa de baud desejada. UARTs têm definições de UART0 e UART1.

### **UART\_SetFlowCtrl**

```
void UART_SetFlowCtrl (uint8 uart_no, UART_HwFlowCtrl flow_ctrl, uint8 rx_thresh)
```

### **UART\_SetIntrEna**

```
void UART_SetIntrEna (uint8 uart_no, uint32 ena_mask)
```

### **UART\_SetLineInverse**

```
void UART_SetLineInverse (uint8 uart_no, UART_LineLevelInverse inverse_mask)
```

Página 391

## **Página 392**

### **UART\_SetParity**

Defina a paridade.

```
void UART_SetParity (uint8 uart_no, UartParityMode parity_mode)
```

Defina a paridade usada pelo UART. O `uart_no` identifica o UART para definir (0 ou 1) e o `parity_mode` define o que usar.

### **UART\_SetPrintPort**

Defina o terminal de saída.

```
void UART_SetPrintPort (uint8 uart_no)
```

Defina o terminal de saída. Defina o UART a ser usado ao escrever depuração via `os_printf()`.

UARTs têm definições de UART0 e UART1.

### **UART\_SetStopBits**

Defina quanto tempo os bits de parada devem ter.

```
void UART_SetStopBits (uint8 uart_no, UartStopBitsNum bit_num)
```

Defina quanto tempo os bits de parada devem ter. O `num` identifica o número de bits de parada a serem usados.

### **UART\_SetWordLength**

Defina o número de bits em uma unidade de transmissão.

```
void UART_SetWordLength (uint8 uart_no, UartBitsNum4Char len)
```

Defina o número de bits em uma unidade de transmissão. O `uart_no` identifica o UART a ser definido (0 ou 1) e o parâmetro `len` define quantos bits.

**UART\_WaitTxFifoEmpty**

Aguarde até que o buffer TX se esvazie.

```
void UART_WaitTxFifoEmpty (uint8 uart_no, uint32 time_out_us)
```

Espere o buffer TX esvaziar. O `uart_no` identifica o UART a ser definido (0 ou 1) e o `time_out_us` especifica quanto tempo esperar antes de desistir. O valor é fornecido em microssegundos.

**uart\_init**

```
void uart_init (UartBautRate uart0BaudRate, UartBautRate uart1BaudRate)
```

Página 392

**Página 393**

Parece haver um erro de digitação no tipo de dados ... mas provavelmente ficaremos presos a isso agora. O `UartBautRate` é um enum que contém:

- `BIT_RATE_9600`
- `BIT_RATE_19200`
- `BIT_RATE_38400`
- `BIT_RATE_57600`
- `BIT_RATE_74880`
- `BIT_RATE_115200`
- `BIT_RATE_230400`
- `BIT_RATE_460800`
- `BIT_RATE_921600`

Veja também:

- [Trabalhando com serial](#)

**uart0\_tx\_buffer**

Transmita um buffer de dados via UART0.

```
void uart0_tx_buffer (uint8 * buffer, comprimento uint16)
```

Transmita os dados apontados pelo buffer para o comprimento fornecido .

Veja também:

- [Trabalhando com serial](#)

**uart0\_sendStr**

Transmita uma string de dados via UART0.

```
void uart0_sendStr (const char * str)
```

Transmita uma string de dados via UART0. A string a ser enviada é fornecida no str parâmetro.

**uart0\_rx\_intr\_handler**

Gerenciar o recebimento de dados via UART0.

```
void uart0_rx_intr_handler (parâmetro void *)
```

Página 393

**Página 394**

O parâmetro é um ponteiro para uma estrutura RcvMsgBuff . Meu melhor palpite sobre como usar isso função é criá-lo em user\_main.c e sua mera existência fará com que seja invocado no momento apropriado. Olhando para a amostra fornecida, vemos que precisa de uma detalhada implementação de baixo nível.

Veja também:

- [Trabalhando com serial](#)

**APIs I2C Master**

Essas funções devem ser compiladas a partir dos arquivos i2c\_master em driver\_lib.

Veja também:

- [Trabalhando com I2C](#)

**i2c\_master\_checkAck**

Recupere o ack do barramento de dados e retorne verdadeiro ou falso.

```
bool i2c_master_checkAck ()
```

Recupere o ack do barramento de dados e retorne verdadeiro ou falso.

**i2c\_master\_getAck**

Recupere o ack do barramento de dados e retorne seu valor.

```
uint8 i2c_master_getAck ()
```

Recupere o ack do barramento de dados e retorne seu valor. Não está claro porque esta função pode ser exposto, bem como o i2c\_master\_checkAck () .

**i2c\_master\_gpio\_init**

Configure os GPIOs e chame i2c\_master\_init () .

```
void i2c_master_gpio_init ()
```

Configure os GPIOs e chame i2c\_master\_init () .

**i2c\_master\_init**

Inicialize funções I2C.

```
void i2c_master_init ()
```

Inicialize funções I2C.

**Página 395**

**i2c\_master\_readByte**  
uint8 i2c\_master\_readByte ()

**i2c\_master\_send\_ack**

void i2c\_master\_send\_ack ()

**i2c\_master\_send\_nack**

void i2c\_master\_send\_nack ()

**i2c\_master\_setAck**

Defina ack para i2c bus como valor de nível.

void i2c\_master\_setAck (nível uint8)

Defina ack para i2c bus como valor de nível.

**i2c\_master\_start**

Defina I2C para enviar estado.

void i2c\_master\_start ()

Defina I2C para enviar estado.

**i2c\_master\_stop**

Defina I2C para parar de enviar estado.

void i2c\_master\_stop ()

Defina I2C para parar de enviar estado.

**i2c\_master\_writeByte**

void i2c\_master\_writeByte (uint8 wrdata)

**APIs SPI**

Essas funções devem ser compiladas a partir dos arquivos SPI em driver\_lib.

**cache\_flush**

---

**Página 396**

**spi\_led\_9bit\_write**

**spi\_mast\_byte\_write**

**spi\_byte\_write\_espslave**

**spi\_slave\_init**

**spi\_slave\_isr\_handler**

**hspi\_master\_readwrite\_repeat**

**spi\_test\_init**

## APIs PWM

**pwm\_init**

Inicialize o PWM.

```
void pwm_init (
    periodo uint32,
    dever uint32 *,
    uint32 num_pwm_channels,
    uint32 (* pin_info_list) [3])
```

O parâmetro de período é o período PWM. O valor é medido em microssegundos com um valor mínimo de 1000, dando um período de 1KHz (há 1000 períodos de 1000 microssegundos em um segundo).

O parâmetro de serviço é a taxa de serviço de cada canal PWM.

O num\_pwm\_channels é o número de canais PWM sendo definidos. Pode haver até para canais PWM\_CHANNEL\_NUM\_MAX . Atualmente, isso é definido como 8.

A pin\_info\_list é um ponteiro para uma matriz de num\_pwm\_channels \* 3 instâncias de uint32s que fornece os mapeamentos de pinos PWM. Os parâmetros por canal PWM são:

- registro GPIO
- Reutilização IO do pino correspondente
- número GPIO

Por exemplo:

Página 396

## Página 397

```
uint32 pinInfoList [] [3] = {
    {PERIPH_IO_MUX_MTDI_U, FUNC_GPIO12, 12},
    {PERIPH_IO_MUX_MTDO_U, FUNC_GPIO15, 15},
    {PERIPH_IO_MUX_MTCK_U, FUNC_GPIO13, 13}
};
```

Veja também:

- [Modulação por largura de pulso - PWM](#)
- [pwm\\_set\\_duty](#)
- [pwm\\_set\\_period](#)
- [pwm\\_start](#)

**pwm\_start**

void pwm\_start ()

Após configurar os parâmetros para PWM, esta função deve ser chamada.

Veja também:

- [Modulação por largura de pulso - PWM](#)

### **pwm\_set\_duty**

void pwm\_set\_duty (uint32 duty, uint8 channel)

A resolução de uma etapa de trabalho é de 45 nanossegundos. Aqui podemos definir o número de tarefas etapas de um ciclo. Por exemplo, imagine que temos um período de 1KHz. Isso significa que 1 o ciclo é de 1000 microssegundos. Se quisermos que o ciclo de trabalho seja de 50%, a produção deve ser alto por 500 microssegundos. 500 microssegundos são 11111 unidades de 45 nanossegundos e isso se tornaria o valor do dever. De maneira geral, a relação de dever é  $(dever * 45) / (período * 1000)$ .

O parâmetro **duty** fornece o número de intervalos de 45 nanossegundos que a saída irá estar alto em um período.

**dever** =  $1000000 / 0,045 / \text{frequência}$

O parâmetro **channel** especifica qual dos canais PWM está sendo alterado.

Depois de alterar o valor do dever, uma chamada para **pwm\_start()** é necessária para recalcular os valores.

Veja também:

- [Modulação por largura de pulso - PWM](#)
- [pwm\\_get\\_duty](#)
- [pwm\\_init](#)

### **pwm\_get\_duty**

uint32 pwm\_get\_duty (canal uint8)

Obtenha o valor do dever do canal especificado.

Página 397

## Página 398

Veja também:

- [Modulação por largura de pulso - PWM](#)
- [pwm\\_get\\_duty](#)
- [pwm\\_init](#)

### **pwm\_set\_period**

Defina o período para as operações PWM.

void pwm\_set\_period (período uint32)

O parâmetro de período é o período PWM. O valor é medido em microssegundos com um valor mínimo de 1000, dando um período de 1KHz (há 1000 períodos de 1000 microssegundos em um segundo).

Veja também:

- [Modulação por largura de pulso - PWM](#)
- [pwm\\_get\\_duty](#)
- [pwm\\_init](#)

### **pwm\_get\_period**

uint32 pwm\_get\_period ()

Obtenha a configuração atual do período PWM.

Veja também:

- [Modulação por largura de pulso - PWM](#)
- [pwm\\_set\\_duty](#)
- [pwm\\_init](#)

**get\_pwm\_version**`uint32 get_pwm_version ()`

Veja também:

- [Modulação por largura de pulso - PWM](#)

**set\_pwm\_debug\_en (uint8 print\_en)**

Usado para ativar ou desativar a impressão de depuração.

**Mexendo um pouco**

- BIT (b) - O valor  $2^b$

Página 398

**Página 399****ESP agora**

`esp_now_add_peer`  
`esp_now_deinit`  
`esp_now_del_peer`  
`esp_now_get_peer_key`  
`esp_now_get_peer_role`  
`esp_now_get_self_role`  
`esp_now_init`  
`esp_now_register_recv_cb`  
`esp_now_register_send_cb`  
`esp_now_send`

A quantidade máxima de dados que podem ser enviados como uma unidade é de 256 bytes.

`esp_now_set_kok`  
`esp_now_set_peer_role`  
`esp_now_set_peer_key`  
`esp_now_set_self_role`  
`esp_now_unregister_recv_cb`  
`esp_now_unregister_send_cb`

**SPIFFS**

Quando uma chamada de API é feita para SPIFFS, ele pode definir um código de erro que pode ser recuperado por uma chamada para `SPIFFS_errno()`. O valor retornado pode ser um dos seguintes:

Símbolo	Valor	Significado
SPIFFS_OK	0	
SPIFFS_ERR_NOT_MOUNTED	-10000	
SPIFFS_ERR_FULL	-10001	
SPIFFS_ERR_NOT_FOUND	-10002	
SPIFFS_ERR_END_OF_OBJECT	-10003	
SPIFFS_ERR_DELETED	-10004	

Página 399

---

## Página 400

SPIFFS_ERR_NOT_FINALIZED	-10005	
SPIFFS_ERR_NOT_INDEX	-10006	
SPIFFS_ERR_OUT_OF_FILE_DESC	-10007	
SPIFFS_ERR_FILE_CLOSED	-10008	
SPIFFS_ERR_FILE_DELETED	-10009	
SPIFFS_ERR_BAD_DESCRIPTOR	-10010	
SPIFFS_ERR_IS_INDEX	-10011	
SPIFFS_ERR_IS_FREE	-10012	
SPIFFS_ERR_INDEX_SPAN_MISMATCH	-10013	
SPIFFS_ERR_DATA_SPAN_MISMATCH	-10014	
SPIFFS_ERR_INDEX_REF_FREE	-10015	
SPIFFS_ERR_INDEX_REF_LU	-10016	
SPIFFS_ERR_INDEX_REF_INVALID	-10017	
SPIFFS_ERR_INDEX_FREE	-10018	
SPIFFS_ERR_INDEX_REF_LU	-10019	
SPIFFS_ERR_INDEX_INVALID	-10020	
SPIFFS_ERR_NOT_WRITABLE	-10021	
SPIFFS_ERR_NOT_READABLE	-10022	
SPIFFS_ERR_CONFLICTING_NAME	-10023	
SPIFFS_ERR_NOT_CONFIGURED	-10024	
SPIFFS_ERR_NOT_A_FS	-10025	
SPIFFS_ERR_MOUNTED	-10026	
SPIFFS_ERR_ERASE_FAIL	-10027	
SPIFFS_ERR_MAGIC_NOT_POSSIBLE	-10028	
SPIFFS_ERR_NO_DELETED_BLOCKS	-10029	
SPIFFS_ERR_INTERNAL	-10050	
SPIFFS_ERR_TEST	-10100	
???	-10072	Possivelmente tenta criar um arquivo que já existe. Também pode significar "sem erro".

Veja também:

- [Sistema de arquivos Spiffs](#)

**esp\_spiffs\_deinit****esp\_spiffs\_init**

Inicialize SPIFFS.

**Página 401**

```
sint32 esp_spiffs_init (struct esp_spiffs_config * config)
```

O parâmetro config é uma estrutura que define as informações de inicialização para SPIFFS.

Contém:

- phys\_size
- phys\_addr
- phys\_erase\_block
- log\_block\_size
- log\_page\_size
- fd\_buf\_size
- cache\_buf\_size

Um exemplo de configuração pode ser:

```
struct esp_spiffs_config config;
config.phys_size = FS1_FLASH_SIZE;
config.phys_addr = FS1_FLASH_ADDR;
config.phys_erase_block = SECTOR_SIZE;
config.log_block_size = LOG_BLOCK;
config.log_page_size = LOG_PAGE;
config.fd_buf_size = FD_BUF_SIZE * 2;
config.cache_buf_size = CACHE_BUF_SIZE;
```

Um código de retorno de 0 significa sucesso.

**SPIFFS\_check**

Executa uma verificação de consistência em determinado sistema de arquivos.

```
s32_t SPIFFS_check (spiffs * fs)
```

**SPIFFS\_clearerr**

Limpa o último erro.

```
void SPIFFS_clearerr (spiffs * fs)
```

**SPIFFS\_close**

Fecha um filehandle. Se houver operações de gravação pendentes, elas serão finalizadas antes fechando.

```
void SPIFFS_close (spiffs * fs, spiffs_file filehandle)
```

Feche o filehandle que foi aberto anteriormente com uma chamada para SPIFFS\_open () .

Veja também:

- [SPIFFS\\_open](#)

**Página 402**

**SPIFFS\_closedir**

Fecha um fluxo de diretório.

```
s32_t SPIFFS_closedir (spiffs_DIR * spiffsDir)
```

O fluxo do diretório deveria ter sido aberto anteriormente com uma chamada para

SPIFFS\_opendir () .

Veja também:

- [SPIFFS\\_opendir](#)
- [SPIFFS\\_readdir](#)

**SPIFFS\_creat**

Crie um arquivo específico.

```
s32_t SPIFFS_creat (spiffs * fs, char * caminho, modo spiffs_mode)
```

Normalmente usa-se SPIFFS\_open () para criar um arquivo.

**SPIFFS\_erase\_deleted\_block**

Apague os blocos excluídos do sistema de arquivos.

```
s32_t SPIFFS_erase_deleted_block (spiffs * fs)
```

**SPIFFS\_errno**

Obtenha o último código de erro.

```
s32_t SPIFFS_errno (spiffs * fs)
```

Recupere o último código de erro.

**SPIFFS\_fflush**

Libere todas as operações de gravação do cache para o sistema de arquivos.

```
s32_t SPIFFS_fflush (spiffs * fs, spiffs_file filehandle)
```

**SPIFFS\_format**

Formata todo o sistema de arquivos.

```
s32_t SPIFFS_format (spiffs * fs);
```

Todos os dados serão perdidos. O sistema de arquivos não deve ser montado ao chamar isso. NB:

a formatação é estranha. Devido à compatibilidade com versões anteriores, SPIFFS\_mount DEVE ser chamado

Página 402

---

**Página 403**

antes da formatação para configurar o sistema de arquivos. Se SPIFFS\_mount for bem-sucedido, SPIFFS\_unmount deve ser chamado antes de chamar SPIFFS\_format. Se SPIFFS\_mount falhar, SPIFFS\_format pode ser chamado diretamente sem chamar SPIFFS\_unmount primeiro.

**SPIFFS\_fremove**

Remova um arquivo por seu identificador de arquivo.

```
s32_t SPIFFS_fremove (spiffs * fs, spiffs_file filehandle)
```

Remova um arquivo por seu identificador de arquivo.

### **SPIFFS\_fstat**

Obtenha o status de um arquivo por meio de um identificador de arquivo.

```
s32_t SPIFFS_fstat (spiffs * fs,
                      spiffs_file filehandle,
                      spiffs_stat * spiffsStat)
```

O spiffs\_stat contém:

- obj\_id
- tamanho - O tamanho do conteúdo do arquivo.
- modelo
- nome - O nome do arquivo.

### **SPIFFS\_gc**

Execute uma coleta de lixo explícita.

```
s32_t SPIFFS_gc (spiffs * fs, tamanho u32_t)
```

Invoca a coleta de lixo para garantir que haja espaço suficiente para bytes de tamanho.

### **SPIFFS\_gc\_quick**

Execute uma coleta de lixo explícita.

```
s32_t SPIFFS_gc_quick (spiffs * fs, u16_t max_free_pages)
```

### **SPIFFS\_info**

Retorne a quantidade total de armazenamento e a quantidade realmente usada.

```
s32_t SPIFFS_info (spiffs * fs, u32_t * total, u32_t * usado)
```

O parâmetro total é o número total de bytes no sistema de arquivos. O usado parâmetro é a quantidade de espaço usado.

Página 403

## **Página 404**

### **SPIFFS\_lseek**

Mova o deslocamento de leitura / gravação para o arquivo.

```
s32_t SPIFFS_lseek (spiffs * fs, spiffs_file filehandle, s32_t offset, int whence)
```

- fs - O sistema de arquivos que possui o arquivo.
- filehandle - Um identificador aberto para o arquivo.
- deslocamento - uma quantidade a ser movida dentro do arquivo.
- de onde - A direção do movimento:
  - SPIFFS\_SEEK\_SET - Mover para um local específico.
  - SPIFFS\_SEEK\_CUR - Mova em relação ao local atual.
  - SPIFFS\_SEEK\_END - Mova em relação ao final do arquivo.

### **SPIFFS\_mount**

Inicializa os parâmetros dinâmicos do sistema de arquivos e monta o sistema de arquivos. Se

SPIFFS\_USE\_MAGIC está habilitado, a montagem pode falhar com SPIFFS\_ERR\_NOT\_A\_FS se o flash não contém um sistema de arquivos reconhecível. Neste caso, SPIFFS\_format deve ser chamado antes da remontagem.

```
s32_t SPIFFS_mount (
    spiffs * fs,
    spiffs_config * config,
    u8_t * work,
    u8_t * fd_space, u32_t fd_space_size,
    void * cache, u32_t cache_size,
    spiffs_check_callback check_cb_f);
```

- fs - A estrutura do sistema de arquivos.
- config - a configuração física e lógica do sistema de arquivos.
- trabalhos
- fd\_space
- fd\_space\_size - Exemplo 32 \* 4.
- cache
- cache\_size - Exemplo (128 + 32) \* 8.
- check\_cb\_f

A estrutura spiffs\_config contém:

Página 404

## Página 405

- hal\_read\_f - função de leitura física. Esta é uma função com a assinatura:

função s32\_t (u32\_t addr, u32\_t size, u8\_t \* dst)

- hal\_write\_f - função de gravação física. Esta é uma função com a assinatura:

função s32\_t (u32\_t addr, u32\_t size, u8\_t \* src)

- hal\_erase\_f - função de apagamento físico. Esta é uma função com a assinatura:

função s32\_t (addr u32\_t, tamanho u32\_t)

- phys\_size - tamanho físico do flash spi.
- phys\_addr - deslocamento físico em spi flash usado para spiffs, deve estar no bloco fronteira.
- phys\_erase\_block - tamanho físico ao apagar um bloco.
- log\_block\_size - tamanho lógico de um bloco, deve estar no tamanho do bloco físico limite e nunca deve ser inferior a um bloco físico. Exemplo 4 \* 1024.
- log\_page\_size - tamanho lógico de uma página, deve ser pelo menos log\_block\_size / 8. Exemplo 128.

### SPIFFS\_mounted

Verifica se o sistema de arquivos está montado.

u8\_t SPIFFS\_mounted (spiffs \* fs)

Retorna 0 se **não** montado.

### SPIFFS\_open

Abra um arquivo.

```
spiffs_file SPIFFS_open (
    spiffs * fs,
    char * path,
    spiffs_flags flags,
    modo spiffs_mode)
```

Abra um arquivo. Isso também pode incluir a criação do arquivo quando ele é aberto.

- fs - O sistema de arquivos a ser aberto.
- caminho - O caminho para o arquivo a ser aberto.
- flags - Sinalizadores de controle para abrir o arquivo. Uma combinação de:
  - SPIFFS\_APPEND
  - SPIFFS\_CREAT
  - SPIFFS\_DIRECT
  - SPIFFS\_RDONLY
  - SPIFFS\_RDWR
  - SPIFFS\_TRUNC
  - SPIFFS\_WRONLY
- modo - O modo para o aberto. Ignorado nesta versão.

Página 405

## Página 406

- SPIFFS\_DIRECT
- SPIFFS\_RDONLY
- SPIFFS\_RDWR
- SPIFFS\_TRUNC
- SPIFFS\_WRONLY

- modo - O modo para o aberto. Ignorado nesta versão.

Veja também:

- [SPIFFS\\_close](#)

### SPIFFS\_open\_by\_dirent

Abra um arquivo por sua entrada de diretório.

```
spiffs_file SPIFFS_open_by_dirent (
    spiffs * fs,
    struct spiffs_dirent * spiffsDirEnt,
    spiffs_flags flags,
    modo spiffs_mode)
```

Abra um arquivo.

- fs - O sistema de arquivos a ser aberto.
- spiffsDirEnt - O caminho para o arquivo a ser aberto.
- flags - Sinalizadores de controle para abrir o arquivo. Uma combinação de:
  - SPIFFS\_APPEND
  - SPIFFS\_DIRECT
  - SPIFFS\_RDONLY
  - SPIFFS\_RDWR
  - SPIFFS\_TRUNC
  - SPIFFS\_WRONLY

### SPIFFS\_opendir

Abra um fluxo de diretório para o nome do diretório especificado.

```
spiffs_DIR * SPIFFS_opendir (
    spiffs * fs,
    char * directoryName,
    spiffs_DIR * spiffsDir)
```

- fs - O sistema de arquivos SPIFFS com o qual trabalhar.

Página 406

## Página 407

- directoryName - O nome do diretório a ser lido.
- spiffsDir - A estrutura de diretório a ser preenchida.

Veja também:

### **SPIFFS\_read**

Leia os dados de um arquivo.

```
s32_t SPIFFS_read (spiffs * fs, spiffs_file filehandle, void * buf, s32_t len)
```

Leia os dados de um arquivo e coloque-os em um buffer.

### **SPIFFS\_readdir**

Leia o diretório.

```
struct spiffs_dirent * SPIFFS_readdir (
    spiffs_DIR * spiffsDir,
    struct spiffs_dirent * spiffsDirEnt)
```

Leia o diretório especificado por spiffsDir que havia sido aberto anteriormente com  
SPIFFS\_opendir () .

Um struct spiffs\_dirent contém:

- obj\_id
- nome
- modelo
- Tamanho
- pix

Veja também:

- [Sistema de arquivos Spiffs](#)
- [SPIFFS\\_opendir](#)
- [SPIFFS\\_closedir](#)
- [SPIFFS\\_open\\_by\\_dirent](#)

### **SPIFFS\_remove**

Remova um arquivo pelo nome.

```
s32_t SPIFFS_remove (spiffs * fs, char * caminho)
```

Página 407

---

**Página 408****SPIFFS\_rename**

Renomeie um arquivo.

```
s32_t SPIFFS_rename (spiffs * fs, char * old, char * newPath)
```

**SPIFFS\_stat**

Obtenha o status de um arquivo por caminho.

```
s32_t SPIFFS_stat (
    spiffs * fs,
    char * path,
    spiffs_stat * spiffsStat)
```

- fs - O sistema de arquivos que contém o arquivo.
- caminho - o caminho para o arquivo.
- spiffsStat - Os dados estatísticos do arquivo.

O spiffs\_stat contém:

- obj\_id
- Tamanho
- modelo
- nome

**SPIFFS\_unmount**

Desmonte um sistema de arquivos.

```
void SPIFFS_unmount (spiffs * fs)
```

**SPIFFS\_write**

Grave dados em um arquivo aberto.

```
s32_t SPIFFS_write (
    spiffs * fs,
    spiffs_file filehandle,
    void * buf, s32_t len)
```

**Lib-C**

O ambiente FreeRTOS fornece um conjunto de rotinas de biblioteca de tempo de execução C que são definidas em "esp\_libc.h".

Página 408

---

**Página 409****atoi**

```
int atoi (const char * s)
```

**atol**

```
long atol (const char * s)
```

**bzero**

```
void bzero (void * s, size_t n)
```

**calloc**

```
void * calloc (size_t c, size_t n)
```

**gratuitamente**

```
void free (void * p)
```

**Malloc**

```
void * malloc (size_t n)
```

**memcmp**

```
int memcmp (const void * m1, const void * m2, size_t n)
```

**memcpy**

```
void * memcpy (void * dst, const void * src, size_t n)
```

**memmove**

```
void * memmove (void * dst, const void * src, size_t n)
```

**memset**

```
void * memset (void * dst, int c, size_t n)
```

**os\_get\_random**

```
int os_get_random (unsigned char * buf, size_t len)
```

**os\_random**

```
os_random longo sem sinal (vazio)
```

**printf**

```
int printf (formato const char *, ... )
```

Precisa incluir "stdio.h".

**coloca**

```
int puts (const char * str)
```

**rand**

Gere um número aleatório.

```
int rand ()
```

Retorne um número aleatório. Observe que o resultado é um número inteiro assinado.

**realloc**

```
void * realloc (void * p, size_t n)
```

**snprintf**

```
int snprintf (char * buf, contagem int sem sinal, formato const char *, ...)
```

**sprintf**

```
int sprintf (char * out, const char * format, ...)
```

**strcat**

```
char * strcat (char * dst, const char * src)
```

**strchr**

```
char * strchr (const char * s, int c)
```

Página 410

**Página 411****strcmp**

```
int strcmp (const char * s1, const char * s2)
```

**forte**

```
char * strcpy (char * dst, const char * src)
```

**strcspn**

```
size_t strcspn (const char * s, const char * rejeitar)
```

**strdup**

```
char * strdup (const char * s)
```

**Strlen**

Retorna o comprimento de uma string terminada em nulo.

```
size_t strlen (const char * s)
```

Retorna o comprimento de uma string terminada em nulo.

**strncat**  
char \* strncat (char \* dst, const char \* src, contagem de tamanho\_t)

**strcmp**  
int strcmp (const char \* s1, const char \* s2, size\_t n)

**forte**  
char \* strncpy (char \* dst, const char \* src, size\_t n)

**strrchr**  
char \* strrchr (const char \* s, int c)

**strspn**  
size\_t strspn (const char \* s, const char \* aceitar)

Página 411

---

**Página 412**

**strstr**  
char \* strstr (const char \* s1, const char \* s2)

**Strtok**  
char \* strtok (char \* s, const char \* delim)

**strtok\_r**  
char \* strtok\_r (char \* s, const char \* delim, char \*\* ptrptr)

**strtol**  
long strtol (const char \* str, char \*\* endptr, int base)

**zalloc**  
void \* zalloc (size\_t n)

**Estruturas de dados**

**esp\_spiffs\_config**

- phys\_size - Tamanho físico do Flash SPI.
- phys\_addr - Deslocamento físico no flash SPI usado para spiffs. Deve estar em um quarteirão fronteira.
- phys\_erase\_block - Tamanho físico ao apagar um bloco.
- log\_block\_size - Tamanho lógico de um bloco. Deve corresponder ao tamanho físico de um quadra.
- log\_page\_size - Tamanho lógico de uma página.
- fd\_buf\_size - Tamanho da área da memória do descritor de arquivo.

- cache\_buf\_size - O tamanho do buffer do cache.

### **station\_config**

Uma descrição de uma configuração de estação. Contém os seguintes campos:

- uint8 ssid [32] - O SSID do ponto de acesso.
- senha uint8 [64] - A senha para acessar o ponto de acesso.
- uint8 bssid\_set - Sinalizador para indicar se deve ou não usar a propriedade bssid . UMA o valor 1 significa usar e o valor 0 significa não usar.

Página 412

## **Página 413**

- uint8 bssid [6] - Se vários pontos de acesso tiverem o mesmo SSID, o BSSID pode contêm um endereço MAC para indicar a qual dos pontos de acesso se conectar.

Veja também:

- [Configuração da estação](#)
- [wifi\\_station\\_get\\_config\\_default](#)
- [wifi\\_station\\_set\\_config\\_current](#)

### **struct softap\_config**

Estrutura de controle de configuração para softAP.

- uint8 ssid [32]
- senha uint8 [64]
- uint8 ssid\_len - O comprimento do SSID. Se for 0, o ssid terá terminação nula.
- canal uint8 - O canal a ser usado para comunicação. Os valores vão de 1 a 13.
- uint8 authmode - O modo de autenticação necessário. As opções são:
  - AUTH\_OPEN
  - AUTH\_WPA2\_PSK
  - AUTH\_WPA\_PSK
  - AUTH\_WPA\_WPA2\_PSK

AUTH\_WEP não é compatível.

- uint8 ssid\_hidden - Se este SSID está ou não oculto. Um valor de 1 torna escondido.
- uint8 max\_connection - O número máximo de conexões de estação. O máximo e o padrão é 4.
- uint16 beacon\_interval - O intervalo do beacon em milissegundos. Os valores são 100 - 60000.

Veja também:

- [wifi\\_softap\\_get\\_config](#)
- [wifi\\_softap\\_get\\_config\\_default](#)
- [wifi\\_softap\\_set\\_config\\_current](#)

### **struct station\_info**

Esta estrutura fornece informações sobre as estações conectadas a um ESP8266 enquanto ele está um ponto de acesso. É uma lista vinculada com propriedades:

- uint8 bssid [6] - O ???

---

**Página 414**

- struct ipaddr ip - O endereço IP da estação conectada

Para obter a próxima entrada, podemos usar STAILQ\_NEXT (pStationInfo, próximo) .

Veja também:

- [Ser um ponto de acesso](#)

**struct dhcps\_lease**

Esta estrutura é usada pela função wifi\_softap\_dhcps\_lease () para definir o início e o intervalo final de endereços IP disponíveis.

Os campos contidos em são:

- struct ip\_addr start\_ip
- struct ip\_addr end\_ip

Inclui:

- user\_interface.h

Veja também:

- [O servidor DHCP](#)

**struct bss\_info**

Esta estrutura contém:

- STAILQ\_ENTRY (bss\_info) próximo
- uint8 bssid [6]
- uint8 ssid [32]
- canal uint8
- sint8 rssi - A indicação de força do sinal recebido
- AUTH\_MODE authmode
- uint8 is\_hidden
- sint16 freq\_offset

Para obter a próxima entrada, podemos usar STAILQ\_NEXT (pBssInfoVar, próximo) .

O AUTH\_MODE é um enum

- AUTH\_OPEN - Sem autenticação. Nenhum desafio em qualquer conexão de estação.
- AUTH\_WEP = 1
- AUTH\_WPA\_PSK = 2

- AUTH\_WPA2\_PSK = 3
- AUTH\_WPA\_WPA2\_PSK = 4

Veja também:

- [Procurando pontos de acesso](#)

#### **struct ip\_info**

Esta estrutura define informações sobre uma interface possuída pelo ESP8266. Isto contém os seguintes campos:

- struct ip\_addr ip - O endereço IP da interface.
- struct ip\_addr netmask - A máscara de rede usada pela interface.
- struct ip\_addr gw - O endereço IP do gateway usado pela interface.

Veja também:

- [struct ip\\_addr](#)
- [IP4\\_ADDR](#)

#### **struct rst\_info**

Informações sobre a inicialização / reinicialização atual

Esta estrutura contém:

- uint32 reason
- uint32 desculpe
- uint32 epc1
- uint32 epc2
- uint32 epc3
- uint32 excvaddr
- uint32 depc

O campo de motivo é um enum com os seguintes valores:

- 0 - Reinicialização padrão - inicialização normal na inicialização
- 1 - Temporizador do watchdog - Reinicialização do watchdog do hardware
- 2 - Exceção - uma exceção foi detectada
- 3 - Temporizador do watchdog do software - Reinicialização do watchdog do software
- 4 - Reinicialização suave

- 5 - Sono profundo, acordar

Veja também:

- [Manipulação de exceção](#)
- [Erro: fonte de referência não encontrada](#)

#### **struct espconn**

Esta estrutura de dados é a representação de uma conexão entre o ESP8266 e um parceiro. Ele contém os "blocos de controle" e informações de identificação ... no entanto, é

É importante observar que nem sempre é um dado opaco.

- enum espconn\_type type - O tipo pode ser um dos
  - ESPCONN\_INVALID
  - ESPCONN\_TCP - Identifica esta conexão como sendo do tipo TCP.
  - ESPCONN\_UDP - Identifica esta conexão como sendo do tipo UDP.
- enum espconn\_state - O estado pode ser um dos
  - ESPCONN\_NONE - O estado para uma conexão inicial.
  - ESPCONN\_WAIT
  - ESPCONN\_LISTEN
  - ESPCONN\_CONNECT
  - ESPCONN\_WRITE
  - ESPCONN\_READ
  - ESPCONN\_CLOSE
- sindicato {
 

```
esp_tcp * tcp
esp_udp * udp
```

} proto - Este campo é uma união de tcp e udp, o que significa que apenas um deles deve ser usado para uma instância desta estrutura de dados. Se a estrutura de dados for usado para TCP, a propriedade tcp deve ser usada, enquanto para UDP, o udp propriedade deve ser usada.
- void \* reverse - Nos comentários, isso é sinalizado como um campo *reservado* para o usuário código. É possível que o nome escolhido (*reverso*) seja na verdade um erro de digitação no cabeçalho Arquivo!!
- Outros campos ... existem outros campos na estrutura, mas eles não se destinam a ser lido ou escrito por aplicativos do usuário. Ignore-os. Usar seus valores é indefinido e pode ter efeitos inesperados.

Página 416

## Página 417

Veja também:

- [TCP](#)
- [esp\\_tcp](#)
- [esp\\_udp](#)

### `esp_tcp`

- uint8 local\_ip [4] - O endereço IP local
- int local\_port - A porta local
- uint8 remote\_ip [4] - O endereço IP remoto
- int remote\_port - A porta remota
- Outros campos ... existem outros campos na estrutura, mas eles não se destinam a ser lido ou escrito por aplicativos do usuário. Ignore-os. Usar seus valores é indefinido e pode ter efeitos inesperados.

Veja também:

- [struct espconn](#)

**esp\_udp**

Esta estrutura de dados é usada na propriedade proto do bloco de controle struct espconn .

- int remote\_port - O endereço IP local
- int local\_port - A porta local
- uint8 local\_ip [4] - O endereço IP remoto
- uint8 remote\_ip [4] - A porta remota

Veja também:

- [struct espconn](#)
- [UDP](#)

**struct ip\_addr**

Uma representação de um endereço IP.

Ele contém o seguinte campo:

- uint32 addr - O endereço IP de 4 bytes real.

Inclui:

- [ip\\_addr.h](#)

Veja também:

- [ipaddr\\_addr](#)

Página 417

**Página 418**

- [IP4\\_ADDR](#)
- [ipaddr\\_t](#)

**ipaddr\_t**

Um typedef para struct ipaddr .

Veja também:

- [struct ip\\_addr](#)

**struct ping\_option**

Os campos contidos na estrutura são:

- contagem uint32 - O número de vezes para transmitir um ping
- uint32 ip - O endereço IP que é o alvo do ping
- uint32 coarse\_time
- recv\_function recv\_function
- sent\_função enviada\_função
- vazio \* reverso;

Inclui:

- [ping.h](#)

Veja também:

- [Pedido de ping](#)
- [ping\\_start](#)
- [ping\\_regist\\_recv](#)
- [ping\\_regist\\_sent](#)

**struct ping\_resp**

Os campos contidos na estrutura são:

- uint32 total\_count
- uint32 resp\_time
- uint32 seqno
- uint32 timeout\_count
- uint32 bytes
- uint32 total\_bytes
- uint32 total\_time

Página 418

---

**Página 419**

- sint8 ping\_err - Uma indicação da ocorrência ou não de um erro. Um valor de 0 significa nenhum erro.

Inclui:

- ping.h

Veja também:

- [Pedido de ping](#)
- [ping\\_start](#)
- [ping\\_regist\\_recv](#)
- [ping\\_regist\\_sent](#)

**struct mdns\_info**

- char \* host\_name
- char \* server\_name
- uint16 server\_port
- unsigned long ipAddr - Este deve ser o endereço IP que está sendo oferecido.
- char \* txt\_data [10] - Uma matriz de opções no formato " nome = valor ".

Veja também:

- [Sistemas de nomes de domínio multicast](#)

**enum phy\_mode**

O modo físico 802.11 a ser usado ou em uso.

- PHY\_MODE\_11B
- PHY\_MODE\_11G
- PHY\_MODE\_11N

**GPIO\_INT\_TYPE**

Esses são os gatilhos possíveis para uma interrupção. Este é um enum definido da seguinte forma:

- GPIO\_PIN\_INTR\_DISABLE - As interrupções estão desabilitadas.
- GPIO\_PIN\_INTR\_POSEDGE - Interrompe em uma transição de borda positiva.

- GPIO\_PIN\_INTR\_NEGEDGE - Interrompe em uma transição de borda negativa.
- GPIO\_PIN\_INTR\_ANYEDGE - Interrompe em qualquer transição de borda.
- GPIO\_PIN\_INTR\_LOLEVEL - Interrompe quando baixo.
- GPIO\_PIN\_INTR\_HILEVEL - Interrompe quando alto.

Página 419

**Página 420**

Veja também:

- [gpio\\_pin\\_wakeup\\_enable](#)

**System\_Event\_t**

O tipo de evento contém:

- evento uint32 - O tipo de evento que ocorreu. Pode ser
  - EVENT\_STAMODE\_CONNECTED (0) - Conectamos com sucesso a um ponto de acesso.
  - uint8 [32] event\_info.connected.ssid - O SSID do ponto de acesso.
  - uint8 ssid\_len
  - uint8 [6] bssid
  - event\_info.connected.channel - O canal usado para se conectar ao ponto de acesso.
- EVENT\_STAMODE\_DISCONNECTED (1)
  - uint8 [6] event\_info.disconnected.bssid
  - uint8 [32] event\_info.disconnected.ssid
  - uint8 ssid\_len
  - uint8 event\_info.disconnected.reason - O motivo é um dos

Segue:

- REASON\_UNSPECIFIED = 1
- REASON\_AUTH\_EXPIRE = 2
- REASON\_AUTH\_LEAVE = 3
- REASON\_ASSOC\_EXPIRE = 4
- REASON\_ASSOC\_TOOMANY = 5
- REASON\_NOT\_AUTHED = 6
- REASON\_NOT\_ASSOCED = 7
- REASON\_ASSOC\_LEAVE = 8
- REASON\_ASSOC\_NOT\_AUTHED = 9
- REASON\_DISASSOC\_PWRCAP\_BAD = 10
- REASON\_DISASSOC\_SUPCHAN\_BAD = 11
- REASON\_IE\_INVALID = 13

---

**Página 421**

- REASON\_MIC\_FAILURE = 14
- REASON\_4WAY\_HANDSHAKE\_TIMEOUT = 15
- REASON\_GROUP\_KEY\_UPDATE\_TIMEOUT = 16
- REASON\_IE\_IN\_4WAY\_DIFFERS = 17
- REASON\_GROUP\_CIPHER\_INVALID = 18
- REASON\_PAIRWISE\_CIPHER\_INVALID = 19
- REASON\_AKMP\_INVALID = 20
- REASON\_UNSUPP\_RSN\_IE\_VERSION = 21
- REASON\_INVALID\_RSN\_IE\_CAP = 22
- REASON\_802\_1X\_AUTH\_FAILED = 23
- REASON\_CIPHER\_SUITE\_REJECTED = 24
- REASON\_BEACON\_TIMEOUT = 200
- REASON\_NO\_AP\_FOUND = 201
- EVENT\_STAMODE\_AUTHMODE\_CHANGE (2)
  - event\_info.auth\_change.old\_mode
  - event\_info.auth\_change.new\_mode
- EVENT\_STAMODE\_GOT\_IP (3)
  - event\_info.got\_ip.ip
  - event\_info.got\_ip.mask
  - event\_info.got\_ip.gw
- EVENT\_SOFTAPMODE\_STACONNECTED (4)
  - event\_info.sta\_connected.mac
  - event\_info.sta\_connected.aid
- EVENT\_SOFTAPMODE\_STADISCONNECTED (5)
  - event\_info.sta\_disconnected.mac
  - event\_info.sta\_disconnected.aid
- EVENT\_STAMODE\_DHCP\_TIMEOUT
- EVENT\_SOFTAPMODE\_PROBEREQRECVED
- Event\_Info\_u event\_info

Esta é uma União C contendo dados que estão disponíveis como uma função do tipo de evento.

- Event\_StaMode\_Connected\_t conectado

Página 421

---

**Página 422**

- Event\_StaMode\_Disconnected\_t desconectado
- Event\_StaMode\_AuthMode\_Change\_t auth\_change
- Event\_StaMode\_Got\_IP\_t got\_ip

- Event\_SoftAPMode\_StaConnected\_t sta\_connected
- Event\_SoftAPMode\_StaDisconnected\_t sta\_disconnected

Veja também:

- [Erro: fonte de referência não encontrada](#)

#### códigos de erro espconn

Constante	Valor
ESPCONN_OK	0
ESPCONN_MEM	-1
ESPCONN_TIMEOUT	-3
ESPCONN_RTE	-4
ESPCONN_INPROGRESS	-5
ESPCONN_ABRT	-8
ESPCONN_RST	-9
ESPCONN_CLSD	-10
ESPCONN_CONN	-11
ESPCONN_ARG	-12
ESPCONN_ISCONN	-15
ESPCONN_HANDSHAKE	-28
ESPCONN_PROTO_MSG	-61

#### STATUS

Este é um enum definido da seguinte forma:

Nome Enum	Valor
OK	0
FALHOU	1
PENDENTE	2
OCUPADO	3
CANCELAR	4

Veja também:

Página 422

---

#### Página 423

- [Erro: fonte de referência não encontrada](#)

Página 423

---

## Página 424

### Materiais de referência

Há uma grande variedade de informações disponíveis sobre o ESP8266 de uma variedade de fontes.

#### Programação C ++

##### Definição de classe simples

Amostra de cabeçalho de classe

```
#ifndef MyClass_h
#define MyClass_h

class MyClass {
    público:
        Minha classe();
        static void myStaticFunc ();
        void myFunc ();
};

#endif
```

Fonte da classe de amostra

```
#include <MyClass.h>
MyClass :: MyClass () {
    // Código do construtor aqui ...
}
```

```

String MyClass :: myStaticFunc () {
    // Codifique aqui ...
}

void MyClass :: myFunc () {
    // Codifique aqui ...
}

```

### Funções lambda

O C ++ moderno introduziu funções lambda. Estas são funções da linguagem C ++ que não precisa ser pré-declarado, mas pode ser declarado "embutido". As funções têm nenhum nome associado a eles, mas de outra forma se comportam como outras funções.

Veja também:

- [Funções lambda](#)

### Ignorando avisos

De vez em quando, seu código pode emitir avisos de compilação que você deseja suprimir. Uma maneira de conseguir isso é através do uso da diretiva C compile `#pragma`.

Por exemplo:

```
#pragma Diagnóstico GCC ignorado "-Wformat"
```

Página 424

## Página 425

Veja também:

- [Pragmas de diagnóstico GCC](#)

### Eclipse

Embora não seja tecnicamente uma história ESP8266, sinto uma compreensão do principal componentes do Eclipse não farão mal.

Veja também:

- [Documentação do Eclipse Marte](#)

### Repartição ESPFS

O ESPFS é uma biblioteca que armazena "arquivos" dentro do flash do ESP8266 e permite um aplicativo para lê-los. Faz parte do projeto ESPHTTPD.

#### EspFsInit

```
EspFsInitResult espFsInit (char * flashAddress)
```

Inicialize o ambiente apontando para onde os dados do arquivo podem ser encontrados. O retorno será um de:

- ESPFS\_INIT\_RESULT\_OK
- ESPFS\_INIT\_RESULT\_NO\_IMAGE
- ESPFS\_INIT\_RESULT\_BAD\_ALIGN

#### espFsOpen

```
EspFsFile * espFsOpen (char * fileName)
```

Abra o arquivo especificado pelo nome do arquivo e retorne uma estrutura que é o "identificador" para o arquivo ou NULL se o arquivo não puder ser encontrado.

**espFsClose**

```
void espFsClose (EspFsFile * fileHandle)
```

Feche o arquivo que foi aberto anteriormente por uma chamada para espFsOpen (). Sem mais leituras deveria ser feito.

**espFsFlags**

```
int espFsFlags (EspFsFile * fileHandle)
```

Página 425

**Página 426****espFsRead**

```
int espFsRead (EspFsFile * fileHandle, char * buffer, comprimento int)
```

Leia até bytes de comprimento do arquivo e armazene-os no local de memória apontado para por buffer. O número real de bytes lidos é retornado pela chamada de função.

**mkespfimage**

Esta não é uma função, mas um comando que constrói os dados binários dos arquivos a serem colocado na memória flash.

```
mkespfimage [-c compressor] [-l compression_level]
```

- -c

- 0 - Nenhum

- 1 - Heatshrink

- -eu

- 

**Repartição ESPHTTPD**

A biblioteca ESPHTTPD fornece uma implementação de um servidor HTTP em execução em um ESP8266. Para usar isso, podemos querer entendê-lo melhor.

**httpdInit**

```
void httpdInit (HttpdBuiltInUrl * fixedUrls, porta int)
```

Inicialize o servidor HTTP em execução no ESP. A porta parâmetro é o número da porta que o ESP ouvirá as solicitações de entrada do navegador. O número da porta padrão usado pelos navegadores é 80.

O HttpdBuiltInUrl é um typedef que fornece mapeamento para URLs disponíveis no Servidor HTTP. Os campos contidos em são:

- char \* url - o url correspondente.
- cgiSendCallback cgiCb - A função de retorno de chamada a ser chamada quando houver correspondência.
- const void \* cgiArg - Parâmetros para passar para a função de retorno de chamada.

É vital que o último elemento da matriz tenha NULLs para todos os atributos. Isso serve como um registro de rescisão. Aqui está uma definição de exemplo para um conjunto mínimo de URLs integrados:

```
HttpdBuiltInUrl builtInUrls [] = {
    {NULL, NULL, NULL}
};
```

O cgiSendCallback é uma função com a seguinte assinatura:

```
int (* functionName) (HttpdConnData * connData)
```

Página 426

## Página 427

Inclui:

- httpd.h

### **httpdGetMimetype**

```
char * httpdGetMimeType (char * url)
```

Examine o url passado e, observando seu tipo de arquivo, determine o tipo MIME de os dados. Se nenhum tipo de arquivo for encontrado, o tipo MIME padrão é " text / html ".

Inclui:

- httpd.h

### **httpdUrlDecode**

```
int httpdUrlDecode (char * val, int valLen, char * ret, int retLen)
```

Decodifique um URL de acordo com as regras de decodificação de URL. O url codificado é fornecido em val com um comprimento de bytes valLen . A string de url decodificada resultante será armazenada em ret com um comprimento máximo de retLen . O comprimento real é retornado pela própria chamada de função.

Inclui:

- httpd.h

### **httpdStartResponse**

```
void httpdStartResponse (HttpdConnData * conn, código int)
```

Comece a enviar os dados de resposta pela conexão TCP para o navegador. O código valor é o código de resposta do navegador principal.

Inclui:

- httpd.h

### **httpdSend**

```
int httpdSend (HttpdConnData * conn, const char * data, int len)
```

Envie dados para o navegador por meio da conexão TCP. Os dados são fornecidos como dados e os parâmetros len são o número de bytes a serem gravados. Se len == -1 , então os dados são assumido como sendo uma string terminada em NULL.

Inclui:

- httpd.h

Página 427

## Página 428

**httpdRedirect**

```
void httpdRedirect (HttpdConnData * conn, char * newUrl)
```

Envie uma instrução de redirecionamento HTTP para o navegador. O newUrl é o URL que desejamos que navegador para usar.

Inclui:

- httpd.h

**httpdHeader**

```
void httpdHeader (HttpdConnData * conn, campo const char *, const char * val)
```

Envie um cabeçalho HTTP. O nome do cabeçalho é fornecido no parâmetro de campo e seu valor fornecido no parâmetro val .

Inclui:

- httpd.h

**httpdGetHeader**

```
int httpdGetHeader (HttpdConnData * conn, char * header, char * ret, int retLen)
```

Pesquise o cabeçalho de dados fornecido pelo navegador procurando um cabeçalho que corresponda ao cabeçalho parâmetro. Se encontrado, retorna o valor do cabeçalho no buffer apontado por ret que deve ter pelo menos retLen bytes de comprimento.

Inclui:

- httpd.h

**httpdFindArg**

```
int httpdFindArg (linha char *, char * arg, char * buff, int buffLen)
```

Dada uma linha de texto, procure um parâmetro da forma " nome = valor " dentro da linha. Se o nome corresponde ao nosso nome passado e, em seguida, retorna o valor.

Inclui:

- httpd.h

**httpdEndHeaders**

```
void httpdEndHeaders (HttpdConnData * conn)
```

Conclua a saída de cabeçalhos para o fluxo de saída.

Inclui:

Página 428

**Página 429**

- httpd.h

**Makefiles**

Livros foram escritos na linguagem e no uso de Makefiles e nosso objetivo não é tentar reescrever esses livros. Em vez disso, aqui está um guia trapaceiro para começar a entender como lê-los.

Uma regra geral em um arquivo make tem a forma:

```
alvo: pré-requisitos ...
      recibo ...
```

As variáveis são definidas na forma:

```
nome = valor
```

Podemos usar o valor de uma variável com \$(nome) ou \${nome}.

Outra forma de definição é:

```
nome:= valor
```

Aqui, o valor está bloqueado em seu valor no momento da definição e não será recursivamente expandido.

Algumas variáveis têm significados bem definidos:

Variável	Significado
CC	Comando do compilador C
AR	Comando Archiver
LD	Comando Linker
OBJCOPY	Comando de cópia de objeto
OBJDUMP	Comando de despejo de objeto

Podemos usar o valor de uma variável previamente definida em outras definições de variável. Para exemplo:

```
XTENSA_TOOLS_ROOT? = C:/Espressif/xtensa-lx106-elf/bin
CC           := $(XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-gcc
```

define o compilador C como um caminho absoluto com base no valor de uma variável anterior.

As expansões especiais são:

- \$@ - O nome do alvo
- \$< - O primeiro pré-requisito

Os comentários são linhas que começam com um caractere "#".

Os curingas são:

Página 429

## Página 430

- \* - Todos os personagens
- ? - Um personagem
- [...] - Um conjunto de caracteres

O make pode ser invocado recursivamente usando

```
make -C <directoryName>
```

Imagine que quiséssemos construir uma lista de arquivos de origem nomeando diretórios e a lista de os arquivos de origem tornam-se todos os arquivos ".c" nesses diretórios? Como podemos alcançar naquela?

```
SRC_DIR = dir1 dir2
SRC: = $(foreach sdir, $(SRC_DIR), $(wildcard $(sdir)/*.C))
OBJ: = $(patsubst%.c, $(BUILD_BASE)%.O, $(SRC))
```

O quebra-cabeça

Imagine uma estrutura de diretório com

uma

```

a1.c
a2.c
b
b1.c
b2.c

```

o objetivo é compilá-los para

```

construir
uma
a1.o
a2.o
b
b1.o
b2.o

```

Nós sabemos como compilar xc → xo

```

MÓDULOS = ab
BUILD_BASE = build
BUILD_DIRS = $ (adicionar prefixo $ (BUILD_BASE) /, $ (MÓDULOS))
SRC = $ (foreach dir, $ (MODULES), $ (wildcard $ (dir) / *. C))
# Substitua todos os xc por xo
OBJS = $ (patsubst% .c%, O, $ (SRC))

```

tudo:

```

echo $ (OBJS)
echo $ (wildcard $ (OBJS) / *. c)
echo $ (foreach dir, $ (OBJS), $ (wildcard $ (dir) / *. c))
echo "SRC:" $ (SRC)

```

```

teste: checkdirs $ (OBJS)
    echo "Compilado" $ (SRC)

```

Página 430

## Página 431

```

.co:
echo "Compilando $ (basename $ <)"
$ (CC) -c $ <-o build / $ (addsuffix .o, $ (basename $ <))

checkdirs: $ (BUILD_DIRS)

$ (BUILD_DIRS):
    mkdir -p $ @

limpar:
    rm -f $ (BUILD_DIRS)

```

Makefiles também têm comandos interessantes:

- \$ (shell <shell command>) - Executar comandos de shell
- \$ (info "texto") , \$ (erro "texto"), \$ (aviso "texto") - Gerar saída do make

Veja também:

- [GNU make](#)
- [Folha de dicas do Makefile](#)

## Fóruns

Existem alguns lugares excelentes para fazer perguntas, responder às perguntas de outras pessoas e leia sobre as perguntas e respostas do passado.

- [Espressif ESP8266 BBS](#) - Um fórum moderado dirigido por Espressif. A fonte primária para downloads do SDK e a fonte de muitos dos materiais principais.
- [Fórum da comunidade ESP8266](#) - Um conjunto de fóruns dedicados ao ESP8266 dirigido para e pela comunidade de usuários do ESP8266.

- [Fórum da](#) comunidade ESP32 - O fórum da comunidade ESP32 onde **todas as** discussões sobre o ESP32 acontecem.

## Documentos de referência

Espressif distribui planilhas em PDF e Excel contendo informações básicas sobre o ESP8266. Eles podem ser baixados gratuitamente da web.

- ## Referência técnica ESP8266 - v1.2

Documentos antigos

- [rdware v1.1](#)  
[0C-ESP8266 WROOM WiFi Module Datasheet v0.3](#)  
[0D-ESP8266 Lançamento da lista de pinos 2014-11-15](#)

Página 431

Página 432

- [8E-ESP8266 Interface UART v0.2](#)
  - [Registros UART de interface 8E-ESP8266 v0.1](#)
  - 
  - [8O-ESP8266 HSPI Host Multi-device API v1.0](#)
  - [30A-ESP8266 Mesh Guia do Usuário V1.0](#)
  - [99A-ESP8266 Operação Flash RW v0.2](#)
  - [99B-ESP8266 Timer \(ainda não publicado\)](#)
  - [99C-ESP8266 Atualização OTA v1.6](#)

Aqui estão documentos de referência semelhantes para o ESP32

- ESP32 RTOS SDK API Reference v1.1.0

Github

Há uma série de projetos de código aberto desenvolvidos sobre e em torno do ESP8266 que pode ser encontrado no Github. Aqui está uma lista de links para alguns desses projetos que são vale muito a pena dar uma olhada:

- [EspressifApp](#)
  - [jantje / arduino-eclipse-plugin](#)
  - [eriksl / esp8266-universal-io-bridge](#)
  - [CHERTS / esp8266-devkit](#)
  - Projeto ESPHTTPD
    - [Spritletm / esphttpd](#)

### Github cheats rápidos

Ao trabalhar com projetos de código aberto, há momentos em que gostaríamos de realizar algumas tarefas que envolvem vários comandos. Aqui, tentamos capturar um pouco mais interessantes que são usados em projetos ESP8266 de vez em quando.

Página 432

---

## Página 433

```
git remote -v  
git remote add upstream <URL>  
git fetch upstream  
git merge upstream / master
```

Veja também:

- [Guia simples para garfes no GitHub e Git](#)

### SDK

O Software Development Kit (SDK) é publicado pela Espressif e é necessário para construir Aplicativos baseados em C. Ele contém documentação vital na forma de PDF que não parecem estar disponíveis em outro lugar.

◦

```
esp-open-sdk
```

```
pacotes cygwin
```

```
búfalo
```

```
flex
```

```
texinfo
```

```
wget
```

```
correção
```

```
libtool
```

```
automake
```

```
gettext-devel
```

Execute git clone --recursive <Repo> de dentro do shell.

### Comparações de computador de placa única

Existem vários computadores de placa única no mercado. Embora o ESP8266 seja geralmente não é considerado um desses, muitas pessoas estão usando como tal. Vamos colocar um mesa e compare o ESP8266 com estes computadores:

---

**Página 434**

<b>Dispositivo</b>	<b>CPU</b>	<b>RAM</b>	<b>Flash Wifi</b>	<b>GPIO</b>	<b>SO</b>	<b>Custo</b>
<a href="#">ESP8266</a> 80 MHz		80K	512K Y	9	FreeRTOS	\$ 4
ESP32	160 MHz	512K	Var Y	?	FreeRTOS	??
<a href="#">Arduino</a> 20 MHz		2K	32K N	?	N / D	\$ 2
<a href="#">Pi Zero</a>	1 GHz	512 MB	SD N	?	Linux	\$ 5
<a href="#">Ómega</a>	400 MHz	64 MB	16 MB Y	18	Linux	\$ 19
Omega 2						
<a href="#">LASCA</a>	1 GHz	512 MB	4GB Y	8+	Linux	\$ 9

**Heróis**

Dentro da comunidade de usuários ESP8266, existem indivíduos que considero ter ultrapassou os limites do conhecimento ou desenvolveu ferramentas que drasticamente melhorar o trabalho com os dispositivos. Eu quero tomar alguns momentos e chamar esses boa gente, sem os quais todas as nossas viagens ESP8266 seriam mais dificeis:

**Max Filippov - jcmvbkbc - Compilador GCC para Xtensa**

- Site: Github - <https://github.com/jcmvbkbc>

Um compilador para C baseado em GCC que compila para o binário Xtensa para flash. Isto é duvidoso que qualquer trabalho útil pudesse ser realizado sem esta contribuição.

**Ivan Grokhotkov - igrr - IDE Arduino para desenvolvimento ESP8266**

- Site: Github - [esp8266 / Arduino](https://github.com/esp8266/Arduino)

---

**Página 435**

Uma implementação de tecnologia que permite desenvolver aplicativos ESP8266 usando o IDE do Arduino, bem como bibliotecas que mapeiam as funções do Arduino para ESP8266

equivalentes ou quase equivalentes.

### jantje - Arduino Eclipse

- Local na rede Internet: [Arduino Eclipse](#)
- Site: Github - [jantje / arduino-eclipse-plugin](#)

Embora não seja tecnicamente apenas para o ESP8266, o projeto Arduino Eclipse é importante. Ele fornece a capacidade de construir aplicativos Arduino usando Eclipse, que possui uma ambiente de desenvolvimento para programadores mais experientes. Combine isso com o Projeto Arduino ESP8266 e temos um ambiente fantástico à nossa disposição.

### Richard Sloan - proprietário da comunidade ESP8266

- Local na rede Internet: <http://www.esp8266.com/index.php>

Sem dúvida, qualquer pessoa que toque no ESP8266 deve visitar esta comunidade na web local. O fórum encontrado é extremamente rico em conhecimento e muito traficado. Pessoal novo para o ESP8266 e especialistas semelhantes são bem-vindos. Simplesmente venha e junte-se a diversão. Richard também é o dono da [themindfactory.com](#).

### Mikhail Grigorev - CHERTS - Eclipse para desenvolvimento ESP8266

- Local na rede Internet: [Kit de desenvolvimento não oficial do projeto para Espressif ESP8266](#)
- Site: Github - [CHERTS / esp8266-devkit](#)

Um conjunto extraordinariamente bem polido de artefatos e instruções para a construção de ESP8266 C aplicativos dentro do ambiente de desenvolvimento Eclipse.

Página 435

---

Página 436

### Mmiscool - Intérprete Básico

- Local na rede Internet: <http://www.esp8266basic.com>
- Local na rede Internet: [fóruns](#)

Um intérprete / ambiente básico para escrever aplicativos na programação básica língua. O autor dedica muito tempo e energia para o desenvolvimento contínuo e fornece uma resposta muito rápida às perguntas do usuário.

## Áreas de Pesquisa

- Temporizadores de hardware ... quando são chamados?
- Se eu definir funções em uma biblioteca chamada libcommon.a, o que é adicionado ao compilado aplicação quando me vinculo a esta biblioteca? É tudo na biblioteca ou apenas o arquivos de objeto que são referenciados?
- Qual é o mapa de memória / layout do ESP8266?
- Quanta RAM está instalada e disponível para uso?
- Documente as informações contidas aqui ...  
<http://bbs.espressif.com/viewtopic.php?p=3066#p3066>
- O que é SSDP e como ele se relaciona com as bibliotecas SSDP?
- Estudo de colmeia de dispositivo - <http://devicehive.com/>
- Documento usando o depurador Visual Micro com Visual Studio.  
<http://www.visualmicro.com/>
- Gerenciamento de energia
- Suporte MQTT
- Pesquise a semântica de um wifi\_station\_connect () quando já estamos conectado.
- Dirija um Arduino como escravo de um ESP8266.