Оглавление

Введение 24
Обзор 25
- ESP8266 26
- Зрелость 27
- Спецификация ESP8266 27
- Модули ESP8266 28
  - ESP-12 28
  - ESP-1 32
  - Adafruit YPA 38
  - NodeMCU devKit 38
  - узел. IT (иначе ESP-210) 40
  - Щит SparkFun WiFi – ESP8266 40
  - Облегченный эспрессо 41
  - Wemos D1 41
  - Дуб digistump 41
- Соединение с ESP8266 41

Теория WiFi 42
ПРИ программировании команды 44
- Команды 45
- Установка последнего B процессоре команды 51

Сборка схем 52
- USB к конвертерам UART 52
- Макеты 54
- Власть 55
- Мультиметр / Логика исследует / Логика Анализатор 56
- Различные компоненты 56
- Физическое строительство 56
- Рекомендуемая установка для программирования ESP8266 56
- Конфигурация для высвечивания устройства 59

Программирование 60
- Способ ботинка 60
- ESP8266 – Software Development Kit (SDK) 61
  - Включайте справочники 61
- Компилирование 62
  - Погрузка программы в ESP8266 69
  - Программирование окружающей среды 73
  - Инструменты компиляции 73
    - площадь 73
    - esptool.py 74
    - esptool-ck 76

gcc 78
gen_appbin.py 79
сделайте 80
nodemcu-маяк-мигалка 80
nm 82
objcopy 82
objdump 82
xxd 82
ESP8266, связывающийся 83
Высвечивание по воздуху – ФОТА 84
Отладка 88
Регистрация ESP-IDF 88
Регистрация к UART1 90
Управляйте Blinky 90
Демпинг IP-адресов 91
Обработка исключений 91
Отладка и тестирование TCP и связей UDP 93
Android – Протокол гнезда 94
Android – Отправитель/Приемник UDP 94
Windows – Геркулес 94
Завиток 94
Затмение – TCP/MON 94
httpbin.org 97
Архитектура ESP8266 97
Таможенные программы 97
WiFi при запуске 97
Работа с WiFi – ESP8266 98
Просмотр для точек доступа 98
Определение рабочего режима 99
Обработка событий WiFi 99
Станционная конфигурация 101
Соединение с точкой доступа 101
Контроль и потоки данных, соединяясь как станция 102
Будучи точкой доступа 103
Сервер DHCP 104
Текущий IP-адрес, netmask и ворота 105
WiFi защищенная установка – WPS 105
Работа с TCP/IP 106
espconn архитектура 106
TCP 107
Отправка и получение данных TCP 111
Управление потоками 113
Обработка ошибок TCP 113
UDP 114

Передача с UDP 116
Запрос звона 117
Служба имен 117
Системы доменных имен передачи 118
Установка добрый день 119
Работа с SNTP 122
ESP ТЕПЕРЬ 123
GPIOs 124
Усилие и сбрасывает параметры настройки 130
Обработка Перерыва GPIO 130
Расширение количества доступного GPIOs 132
Библиотека ESP_PCF8574 C 136
Библиотека PCF8574 JavaScript 137
Работа с I2C 137
Работа с SPI – последовательный периферийный интерфейс 139
Аппаратные средства SPI 141
MetalPhreak/ESP8266_SPI_Driver 143
Работа с сериалом 144
Обработка Задачи ESP8266 146
Таймеры и время 147
Работа с памятью 148
Работа с флэш-памятью 152
Модуляция ширины импульса – PWM 153
Аналог к цифровому преобразованию 154
Режимы ожидания 156
Охранительный таймер 157
Получение контроля 158
Безопасность 159
Отображение из Ардуино 159
Файловая система магарычей при торговой сделке 160
Партнер API TCP/IP 161
Гнезда TCP/IP 162
Ошибки из-за неправильного обращения 165
Гнезда – принимают () 168
Гнезда – связывают () 169
Гнезда – близко () 169
Гнезда – closesocket () 169
Гнезда – соединяются () 169
Гнезда – fcntl () 170
Гнезда – freeaddrinfo () 170
Гнезда – getaddrinfo () 170
Гнезда – gethostbyname () 170

Гнезда – getpeername () 170
Гнезда – getsockname () 170

Страница 4

Гнезда – getsockopt () 170
Гнезда – htonl () 171
Гнезда – htons () 171
Гнезда – inet_ntop () 171
Гнезда – inet_pton () 171
Гнезда – ioctlsocket () 171
Гнезда – слушают () 171
Гнезда – читали () 171
Гнезда – recv () 172
Гнезда – recvfrom () 172
Гнезда – избранный () 173
Гнезда – посылают () 173
Гнезда – sendto () 173
Гнезда – setsockopt () 173
Гнезда – закрытие () 174
Гнезда – гнездо () 174
Гнезда – пишут () 174
Структуры данных гнезда 175
Гнезда – структура sockaddr 175
Гнезда – структура sockaddr_in 175
Явские гнезда 175
WebSockets 178
Приложение браузера WebSocket 178
FreeRTOS WebSocket 180
Мангуста WebSocket 180
Веб-серверы 181
Мангуста 181
Программирование использующий Затмение 182
Установка Затмения Последовательный терминал 186
Веб-разработка используя Затмение 192
Программирование использования IDE Ардуино 193
Последствия Ардуино поддержка IDE 194
Установка IDE Ардуино с поддержкой ESP8266 195
Советы для работы в окружающей среде Ардуино 201
Инициализируйте глобальные классы в установке () 201
Призыв API Espressif SDK из эскиза 201
Обработка исключений 202
Файловая система МАГАРЫЧЕЙ ПРИ ТОРГОВОЙ СДЕЛКЕ 202
Команда mkspiffs 202
Архитектура поддержки IDE Ардуино 203

Строительство ESP приложения Ардуино, используя Затмение IDE 211
Причины рассмотреть использование Затмения по Ардуино IDE 225
Примечания по использованию Затмения пакет Ардуино 226
Библиотеки ESP Ардуино 227

Библиотека WiFi 227
WiFi.begin 227
WiFi.beingSmartConfig 228
WiFi.beginWPSConfig 228
WiFi. BSSID 228
WiFi. BSSIDstr 228
Канал WiFi 228
WiFi.config 229
WiFi.disconnect 229
WiFi.encryptionType 229
WiFi.gatewayIP 229
WiFi.getNetworkInfo 229
WiFi.hostByName 230
WiFi.hostname 230
WiFi.isHidden 230
WiFi.localIP 230
WiFi.macAddress 230
WiFi.mode 231
WiFi.printDiag 231
WiFi. RSSI 231
WiFi.scanComplete 231
WiFi.scanDelete 232
WiFi.scanNetworks 232
WiFi.smartConfigDone 232
WiFi.softAP 232
WiFi.softAPConfig 233
WiFi.softAPdisconnect 233
WiFi.softAPmacAddress 233
WiFi.softAPIP 233
WiFi. SSID 233
WiFi.status 233
WiFi.stopSmartConfig 234
WiFi.subnetMask 234
WiFi.waitForConnectResult 234
WiFiClient 234
WiFiClient 234
WiFiClient.available 234
WiFiClient.connect 235

WiFiClient.connected 235
WiFiClient.flush 235
WiFiClient.getNoDelay 235
WiFiClient.peek 235
WiFiClient.read 235
WiFiClient.remoteIP 235

WiFiClient.remotePort 236
WiFiClient.setLocalPortStart 236
WiFiClient.setNoDelay 236
WiFiClient.status 236
WiFiClient.stop 236
WiFiClient.stopAll 236
WiFiClient.write 236
WiFiServer 237
WiFiServer 237
WiFiServer.available 237
WiFiServer.begin 237
WiFiServer.getNoDelay 237
WiFiServer.hasClient 237
WiFiServer.setNoDelay 238
WiFiServer.status 238
WiFiServer.write 238
IPAddress 238
ESP8266WebServer 238
ESP8266WebServer 241
ESP8266WebServer.arg 241
ESP8266WebServer.argName 241
ESP8266WebServer.args 241
ESP8266WebServer.begin 241
ESP8266WebServer.client 241
ESP8266WebServer.handleClient 242
ESP8266WebServer.hasArg 242
ESP8266WebServer.method 242
ESP8266WebServer.on 242
ESP8266WebServer.onFileUpload 243
ESP8266WebServer.onNotFound 243
ESP8266WebServer.send 243
ESP8266WebServer.sendContent 243
ESP8266WebServer.sendHeader 243
ESP8266WebServer.setContentLength 243
ESP8266WebServer.streamFile 244
ESP8266WebServer.upload 244

ESP8266WebServer.uri 244
Библиотека ESP8266mDNS 244
MDNS.addService 244
MDNS.begin 244
MDNS.update 244
I2C – Провод 245
Wire.available 245
Wire.begin 245

Wire.beginTransmission 246
Wire.endTransmission 246
Wire.flush 246
Wire.onReceive 247
Wire.onReceiveService 247
Wire.onRequest 247
Wire.onRequestService 247
Wire.peek 247
Wire.pins 247
Wire.read 248
Wire.requestFrom 248
Wire.setClock 248
Wire.write 249
Библиотека тикера 249
Тикер 249
будьте свойственны 249
attach_ms 250
отделите 250
однажды 250
once_ms 250
Библиотека EEPROM 250
EEPROM.begin 251
EEPROM.commit 251
EEPROM.end 251
EEPROM.get 251
EEPROM.getDataPtr 251
EEPROM.put 251
EEPROM.read 251
EEPROM.write 251
МАГАРЫЧИ ПРИ ТОРГОВОЙ СДЕЛКЕ 252
SPIFFS.begin 252
SPIFFS.open 252
SPIFFS.openDir 252
SPIFFS.remove 252

SPIFFS.rename 253
File.available 253
File.close 253
File.flush 253
File.name 253
File.peek 253
File.position 253
File.read 253
File.seek 254
File.size 254

File.write 254
Dir.fileName 254
Dir.next 254
Dir.open 254
Dir.openDir 254
Dir.remove 255
Dir.rename 255
Библиотека ESP 255
ESP.deepSleep 255
ESP.eraseConfig 255
ESP.getBootMode 255
ESP.getBootVersion 255
ESP.getChipId 255
ESP.getCpuFreqMHz 255
ESP.getCycleCount 255
ESP.getFlashChipId 255
ESP.getFlashChipMode 255
ESP.getFlashChipRealSize 256
ESP.getFlashChipSize 256
ESP.getFlashChipSizeByChipId 256
ESP.getFlashChipSpeed 256
ESP.getFreeHeap 256
ESP.getFreeSketchSpace 256
ESP.getResetInfo 256
ESP.getResetInfoPtr 256
ESP.getSdkVersion 256
ESP.getSketchSize 256
ESP.getVcc 257
ESP.reset 257
ESP.restart 257
ESP.updateSketch 257
ESP.wdtDisable 257

ESP.wdtEnable 257
ESP.wdtFeed 257
Библиотека последовательности 258
Конструктор 258
Последовательность c_str 258
String.reserve 258
String.length 258
String.concat 258
String.equalsIgnoreCase 258
String.startsWith 258
String.endsWith 259
String.charAt 259

String.setCharAt 259
String.getBytes 259
Последовательность toCharArray 259
String.indexOf 259
String.lastIndexOf 259
String.substring 259
String.replace 259
String.remove 259
String.toLowerCase 259
String.toUpperCase 259
String.trim 260
String.toInt 260
String.toFloat 260
Программирование с JavaScript 260
Smart.js 261
Smart.js GPIO 262
Подготовка сервера HTTP 263
Отладка 264
Espruino 265
Редактирование и развертывание кодекса 265
Работа с переменными 266
Загрузка Espruino 266
Доступ WiFi 266
Написание сетевых приложений гнезда, используя Espruino 267
Написание клиента ОТДЫХА, использующего Espruino 268
Написание веб-сервера, используя Espruino 269
Работа с GPIO 271
Работа с I2C и JavaScript 271
Отладка JavaScript 272
Редактирование JavaScript 272

Библиотеки Espruino ESP8266 273
Основные возможности JavaScript 274
Управление кодексом с промежутками 274
Работа с GPIO 275
SPI 275
Основные отличия из JavaScript 276
Строительство Еспруино 276
Программирование с Lua 277
ESPlorer IDE 277
GPIO с Lua 277
WiFi с Lua 278
Организация сети с Lua 278
Программирование с основным 278
Интеграция с веб-приложениями 278

Страница 10

REST Services 278
Протокол ОТДЫХА 279
ESP8266 как клиент ОТДЫХА 279
Создание запроса ОТДЫХА, используя Мангусту 279
ESP8266 как поставщик услуг ОТДЫХА 280
Tasker 280
AutoRemote 280
DuckDNS 282
Мобильные приложения 283
Blynk 283
Типовые отрывки 283
Формирование соединения по протоколу TCP 283
Примеры приложения 284
Образец – Освещает светодиод на основе прибытия дейтаграммы UDP 284
Образец – Сверхзвуковое измерение расстояния 286
Образец – сканер WiFi 289
Образец – Работающий с микро SD-картами 289
Образец – Игра аудио от события 289
Образец – изменчивый свет настроения 289
Образец – Улучшающий организацию сети 294
Типовые библиотеки 294
Список функции 294
authModeToString 294
checkError 295
delayMilliseconds 295
dumpBSSINFO 295
dumpEspConn 295
dumpRestart 295

dumpState 295
errorToString 296
eventLogger 296
eventReasonToString 296
flashSizeAndMapToString 296
setAsGpio 296
setupBlink 296
toHex 297
Использование FreeRTOS 297
Архитектура задачи в FreeRTOS 298
Блокирование и синхронизация в RTOS 300
Списки в RTOS 300
ESP8266 – Строительство приложений для RTOS 300
Пульты с RTOS 302
Отладка подсказок 303
Развитие решений на Linux 304

Строительство окружающей среды Linux 304
Ссылка API 313
Ссылка FreeRTOS API 313
eTaskGetState 313
pcTaskGetName 313
xEventGroupClear 314
xEventGroupCreate 314
xEventGroupSetBits 314
xEventGroupWaitBits 314
xTaskCreate 315
vTaskDelay 316
vTaskDelayUntil 316
vTaskDelete 316
xTaskGetCurrentTaskHandle 317
xTaskGetTickCount 317
vEventGroupDelete 317
vTaskList 317
vTaskPrioritySet 317
vTaskResume 317
xTaskResumeAll 317
vTaskResumeFromISR 317
vTaskSuspend 318
vTaskSuspendAll 318
xQueueCreate 318
vQueueDelete 318
xQueuePeek 318

xQueueReceive 318
xQueueSend 318
xQueueSendToBack 319
xQueueSendToFront 319
vSemaphoreCreateBinary 319
xSemaphoreCreateCounting 319
vSemaphoreGive 319
xSemaphoreGiveFromISR 319
vSemaphoreTake 319
pvPortMalloc 319
pvPortFree 319
Обработка списка 319
vListInitialise 319
vListInitialiseItem 319
vListInsert 320
vListInsertEnd 320
Ссылка lwip 320
Гнезда 320

Функции таймера 321
os_delay_us 321
os_timer_arm 321
os_timer_disarm 322
os_timer_setfn 322
system_timer_reinit 323
os_timer_arm_us 323
hw_timer_init 323
hw_timer_arm 323
hw_timer_set_func 323
Системные функции 323
system_adc_read 323
system_deep_sleep_set_option 323
system_get_boot_mode 323
system_get_boot_version 324
system_get_chip_id 324
system_get_cpu_freq 324
system_get_flash_size_map 324
system_get_rst_info 325
system_get_userbin_addr 325
system_get_vdd33 325
system_init_done_cb 325
system_os_post 326
system_os_task 326

system_phys_set_rfoption 327
system_phys_set_max_tpw 327
system_phys_set_tpw_via_vdd33 327
system_print_meminfo 327
system_restart_enhance 328
system_rtc_clock_cali_proc 328
system_set_os_print 328
system_show_malloc 328
system_rtc_clock_cali_proc 329
system_uart_swap 329
system_soft_wdt_feed 329
system_soft_wdt_stop 329
system_soft_wdt_restart 330
system_uart_de_swap 330
system_update_cpu_freq 330
os_memset 330
os_memcmp 330
os_memcpy 331
os_malloc 331
os_calloc 331

os_realloc 331
os_zalloc 332
os_free 332
os_bzero 332
os_delay_us 333
os_printf 333
os_install_putc1 333
os_random 334
os_get_random 334
os_strlen 334
os_strcat 334
os_strchr 335
os_strcmp 335
os_strcpy 335
os_strncmp 335
os_strncpy 335
os_sprintf 336
os_strstr 336
Вспышка SPI 336
spi_flash_get_id 336
spi_flash_erase_sector 336
spi_flash_read 337

spi_flash_set_read_func 337
system_param_save_with_protect 337
spi_flash_write 337
system_param_load 338
WiFi – ESP8266 338
wifi_fpm_close 338
wifi_fpm_do_sleep 338
wifi_fpm_do_wakeup 338
wifi_fpm_get_sleep_type 338
wifi_fpm_open 338
wifi_fpm_set_sleep_type 338
wifi_fpm_set_wakeup_cb 338
wifi_get_channel 338
wifi_get_ip_info 338
wifi_get_macaddr 339
wifi_get_opmode 339
wifi_get_opmode_default 339
wifi_get_phy_mode 340
wifi_get_sleep_type 340
wifi_get_user_fixed_rate 340
wifi_get_user_limit_rate_mask 340
wifi_set_broadcast_if 340

wifi_get_broadcast_if 341
wifi_set_sleep_type 341
wifi_promiscuous_enable 341
wifi_promiscuous_set_mac 341
wifi_register_rfid_locp_recv_cb 341
wifi_register_send_pkt_freedom_cb 341
wifi_register_user_ie_manufacturer_recv_cb 341
wifi_rfid_locp_recv_close 341
wifi_rfid_locp_recv_open 341
wifi_send_pkt_freedom 341
wifi_set_channel 341
wifi_set_event_handle_cb 341
wifi_set_ip_info 342
wifi_set_macaddr 342
wifi_set_opmode 342
wifi_set_opmode_current 343
wifi_set_phy_mode 343
wifi_set_promiscuous_rx_cb 343
wifi_set_sleep_type 344
wifi_set_user_fixed_rate 344

wifi_set_user_ie 344
wifi_set_user_limit_rate_mask 344
wifi_set_user_rate_limit 344
wifi_set_user_sup_rate 344
wifi_status_led_install 345
wifi_status_led_uninstall 345
wifi_unregister_rfid_locp_recv_cb 346
wifi_unregister_send_pkt_freedom_cb 346
wifi_unregister_user_ie_manufacturer_recv_cb 346
Станция WiFi 346
wifi_station_ap_change 346
wifi_station_ap_number_set 346
wifi_station_connect 346
wifi_station_dhcpc_start 347
wifi_station_dhcpc_status 347
wifi_station_dhcpc_stop 347
wifi_station_disconnect 347
wifi_station_get_ap_info 348
wifi_station_get_auto_connect 348
wifi_station_get_config 348
wifi_station_get_config_default 349
wifi_station_get_connect_status 349
wifi_station_get_current_ap_id 349
wifi_station_get_hostname 350

wifi_station_get_reconnect_policy 350
wifi_station_get_rssi 350
wifi_station_scan 350
wifi_station_set_auto_connect 351
wifi_station_set_cert_key 352
wifi_station_clear_cert_key 352
wifi_station_set_config 352
wifi_station_set_config_current 353
wifi_station_set_reconnect_policy 353
wifi_station_set_hostname 353
WiFi SoftAP 353
wifi_softap_dhcps_start 353
wifi_softap_dhcps_status 354
wifi_softap_dhcps_stop 354
wifi_softap_free_station_info 354
wifi_softap_get_config 355
wifi_softap_get_config_default 355
wifi_softap_get_dhcps_lease 356

wifi_softap_get_dhcps_lease_time 356
wifi_softap_get_station_info 356
wifi_softap_get_station_num 356
wifi_softap_reset_dhcps_lease_time 356
wifi_softap_set_config 357
wifi_softap_set_config_current 357
wifi_softap_set_dhcps_lease 357
wifi_softap_set_dhcps_lease_time 358
wifi_softap_dhcps_offer_option 358
WiFi WPS 358
wifi_wps_enable 358
wifi_wps_disable 359
wifi_wps_start 359
wifi_set_wps_cb 359
API модернизации 359
system_upgrade_flag_check 359
system_upgrade_flag_set 360
system_upgrade_reboot 360
system_upgrade_start 360
system_upgrade_userbin_check 360
wifi_promiscuous_enable 360
wifi_promiscuous_set_mac 360
wifi_promiscuous_rx_cb 360
wifi_get_channel 361
wifi_set_channel 361
Умные API конфигурации 361

smartconfig_start 361
smartconfig_stop 361
API SNTP 361
sntp_setserver 361
sntp_getserver 361
sntp_setservername 362
sntp_getservername 362
sntp_init 362
sntp_stop 363
sntp_get_current_timestamp 363
sntp_get_real_time 363
sntp_set_timezone 363
sntp_get_timezone 364
Универсальные API TCP/UDP 364
espconn_delete 364
espconn_dns_setserver 364

espconn_gethostbyname 365
espconn_port 366
espconn_regist_sentcb 366
espconn_regist_recvcb 366
espconn_send 367
espconn_sendto 367
ipaddr_addr 367
IP4_ADDR 368
IP2STR 368
API TCP 368
espconn_abort 368
espconn_accept 369
espconn_get_connection_info 369
espconn_connect 370
espconn_disconnect 370
espconn_regist_connectcb 371
espconn_regist_disconcb 371
espconn_regist_reconcb 371
espconn_regist_write_finish 372
espconn_set_opt 373
espconn_clear_opt 373
espconn_regist_time 374
espconn_set_keepalive 374
espconn_get_keepalive 375
espconn_secure_accept 375
espconn_secure_ca_disable 375
espconn_secure_ca_enable 375
espconn_secure_set_size 375

espconn_secure_get_size 375
espconn_secure_delete 375
espconn_secure_connect 375
espconn_secure_send 376
espconn_secure_disconnect 376
espconn_tcp_get_max_con 376
espconn_tcp_set_max_con 376
espconn_tcp_get_max_con_allow 376
espconn_tcp_set_max_con_allow 376
espconn_recv_hold 376
espconn_recv_unhold 377
API UDP 377
espconn_create 377
espconn_igmp_join 377

espconn_igmp_leave 377
API звона 378
ping_start 378
ping_regist_recv 378
ping_regist_sent 378
API mDNS 379
espconn_mdns_init 379
espconn_mdns_close 379
espconn_mdns_server_register 379
espconn_mdns_server_unregister 379
espconn_mdns_get_servername 379
espconn_mdns_set_servername 379
espconn_mdns_set_hostname 380
espconn_mdns_get_hostname 380
espconn_mdns_disable 380
espconn_mdns_enable 380
GPIO – ESP32 380
gpio_config 380
gpio_get_level 382
gpio_input_get 382
gpio_input_get_high 382
gpio_intr_enable 382
gpio_intr_disable 382
gpio_isr_register 382
gpio_output_set 383
gpio_output_set_high 383
gpio_set_direction 383
gpio_set_intr_type 384
gpio_set_level 384
gpio_set_pull_mode 384

GPIO – ESP8266 385
PIN_PULLUP_DIS 386
PIN_PULLUP_EN 387
PIN_FUNC_SELECT 387
GPIO_ID_PIN 387
GPIO_OUTPUT_SET 387
GPIO_DIS_OUTPUT 388
GPIO_INPUT_GET 388
gpio_output_set 388
gpio_input_get 389
gpio_intr_handler_register 389
gpio_pin_intr_state_set 389

gpio_intr_pending 390
gpio_intr_ack 390
gpio_pin_wakeup_enable 390
gpio_pin_wakeup_disable 391
API UART 391
UART_CheckOutputFinished 391
UART_ClearIntrStatus 391
UART_ResetFifo 391
UART_SetBaudrate 391
UART_SetFlowCtrl 391
UART_SetIntrEna 391
UART_SetLineInverse 391
UART_SetParity 392
UART_SetPrintPort 392
UART_SetStopBits 392
UART_SetWordLength 392
UART_WaitTxFifoEmpty 392
uart_init 392
uart0_tx_buffer 393
uart0_sendStr 393
uart0_rx_intr_handler 393
API владельца I2C 394
i2c_master_checkAck 394
i2c_master_getAck 394
i2c_master_gpio_init 394
i2c_master_init 394
i2c_master_readByte 395
i2c_master_send_ack 395
i2c_master_send_nack 395
i2c_master_setAck 395
i2c_master_start 395
i2c_master_stop 395

i2c_master_writeByte 395
API SPI 395
cache_flush 395
spi_lcd_9bit_write 396
spi_mast_byte_write 396
spi_byte_write_espslave 396
spi_slave_init 396
spi_slave_isr_handler 396
hspi_master_readwrite_repeat 396
spi_test_init 396

API PWM 396
pwm_init 396
pwm_start 397
pwm_set_duty 397
pwm_get_duty 397
pwm_set_period 398
pwm_get_period 398
get_pwm_version 398
set_pwm_debug_en (uint8 print_en) 398
Битовое жонглирование 398
ESP теперь 399
esp_now_add_peer 399
esp_now_deinit 399
esp_now_del_peer 399
esp_now_get_peer_key 399
esp_now_get_peer_role 399
esp_now_get_self_role 399
esp_now_init 399
esp_now_register_recv_cb 399
esp_now_register_send_cb 399
esp_now_send 399
esp_now_set_kok 399
esp_now_set_peer_role 399
esp_now_set_peer_key 399
esp_now_set_self_role 399
esp_now_unregister_recv_cb 399
esp_now_unregister_send_cb 399
МАГАРЫЧИ ПРИ ТОРГОВОЙ СДЕЛКЕ 399
esp_spiffs_deinit 400
esp_spiffs_init 400
SPIFFS_check 401
SPIFFS_clearerr 401
SPIFFS_close 401
SPIFFS_closedir 402

SPIFFS_creat 402
SPIFFS_erase_deleted_block 402
SPIFFS_errno 402
SPIFFS_fflush 402
SPIFFS_format 402
SPIFFS_fremove 403
SPIFFS_fstat 403
SPIFFS_gc 403

SPIFFS_gc_quick 403
SPIFFS_info 403
SPIFFS_lseek 404
SPIFFS_mount 404
SPIFFS_mounted 405
SPIFFS_open 405
SPIFFS_open_by_dirent 406
SPIFFS_opendir 406
SPIFFS_read 407
SPIFFS_readdir 407
SPIFFS_remove 407
SPIFFS_rename 408
SPIFFS_stat 408
SPIFFS_unmount 408
SPIFFS_write 408
Lib-C 408
atoi 409
атолл 409
bzero 409
calloc 409
свободный 409
malloc 409
memcmp 409
memcpy 409
memmove 409
memset 409
os_get_random 409
os_random 410
printf 410
помещает 410
рэнд 410
realloc 410
snprintf 410
sprintf 410
strcat 410
strchr 410

strcmp 411
strcpy 411
strcspn 411
strdup 411
strlen 411
strncat 411

strncmp 411
strncpy 411
strrchr 411
strspn 411
strstr 412
strtok 412
strtok_r 412
стертол 412
zalloc 412
Структуры данных 412
esp_spiffs_config 412
station_config 412
структура softap_config 413
структура station_info 413
структура dhcps_lease 414
структура bss_info 414
структура ip_info 415
структура rst_info 415
структура espconn 416
esp_tcp 417
esp_udp 417
структура ip_addr 417
ipaddr_t 418
структура ping_option 418
структура ping_resp 418
структура mdns_info 419
enum phy_mode 419
GPIO_INT_TYPE 419
System_Event_t 420
коды ошибок espconn 422
СТАТУС 422
Справочные материалы 424
C ++ программирование 424
Простое определение класса 424
Функции лямбды 424
Игнорирование предупреждений 424
Затмение 425
Расстройство ESPFS 425

EspFsInit 425
espFsOpen 425
espFsClose 425
espFsFlags 425

espFsRead 426
mkespfimage 426
Расстройство ESPHTTPD 426
httpdInit 426
httpdGetMimetype 427
httpdUrlDecode 427
httpdStartResponse 427
httpdSend 427
httpdRedirect 428
httpdHeader 428
httpdGetHeader 428
httpdFindArg 428
httpdEndHeaders 428
Makefiles 429
Форумы 431
Справочные документы 431
GitHub 432
GitHub быстрые обманы 432
SDK 433
Сравнения одноплатного компьютера 433
Герои 434
Макс Филиппов – jcmvbkbc – компилятор GCC для Xtensa 434
Иван Грохотков – igrr – Ардуино IDE для развития ESP8266 434
jantje – Затмение Ардуино 435
Ричард Слоан – Владелец Сообщества ESP8266 435
Михаил Григорев – ЧЕРТЫ – Затмение для развития ESP8266 435
Mmiscool – Основной переводчик 436
Области к исследованию 436

Введение

Привет люди,

Я работал в бизнесе программного обеспечения больше 30 лет, но до недавнего времени, не играл непосредственно с Микро Процессорами. Когда я купил ПИ Малины и затем Ардуино, я боюсь, что был завербован. В моем доме я окружен компьютерами всех форм, размеров и мощностей … любой из них с порядками величины больше власти, чем любое из этих маленьких устройств … однако, я все еще нашел меня очарованным.

Когда я споткнулся через ESP8266 в начале 2015, он возбудил мой интерес. Я не коснулся C, программирующего в десятилетиях (я - Питекантроп в эти дни). Когда я начал читать то, что было доступно в способе документации от превосходного сообщества, окружающего устройство, я нашел, что были только маленькие очаги знания. Лучший источник информации был (и все еще), официальный PDFs для SDK от Espressif (производители ESP8266), но даже который довольно «легок» на примерах и фоне. Когда я изучил устройство, я начал делать примечания, и мои страницы примечаний продолжали расти и расти.

Эта книга (если мы хотим назвать его, что) является моей сопоставленной и полированной версией тех примечаний. Вместо того, чтобы держать их мне, я предлагаю их всем нам в сообществе ESP8266 в надежде, что они будут иметь некоторую ценность. Мой план состоит в том, чтобы продолжить обновлять эту работу, поскольку все мы узнаем больше и разделяем то, что мы находим на общественных форумах. По сути, я повторно выпущу работу равномерно, поэтому, пожалуйста, перепроверьте в домашней странице книги для последнего.

Поскольку Вы читаете, удостоверьтесь, что Вы полностью понимаете, что есть, несомненно, погрешности, ошибки в моем понимании и ошибки в моем письме. Только обратной связью и время будет мы быть в состоянии исправить их. Пожалуйста, простите грамматические ошибки и орфографические ошибки, которые не поймал мой спеллчекер.

Для вопросов или комментариев к книге, пожалуйста,

отправьте этой нити

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

Домашняя страница

для

книги:http://neilkolban.co

m/tech/esp8266/

Пожалуйста, не посылайте мне по электронной почте непосредственно с техническими вопросами. Вместо этого давайте использовать форум и давайте спросим и ответим на вопросы, поскольку великое сообщество ESP8266 возражало против энтузиастов, людей, увлеченных своим хобби, и профессионалов.

Нил Колбэн

Техас, США

Обзор

Микро контроллер - интегральная схема, которая способна к бегущим программам. Есть много случаев тех, которые на рынке сегодня от множества производителей. Цены на эти микро контроллеры продолжают падать. На рынке человека, увлеченного своим хобби, общедоступная архитектура под названием «Ардуино», который использует ряд Atmel процессоров, поймала воображение бесчисленных людей. Правления, содержащие этот жареный картофель Atmel, объединенный с конвенцией для связей и также свободного набора средств разработки, понизили точку входа для игры с электроникой к фактически nill. В отличие от PC, эти процессоры - чрезвычайно нижний уровень с низкими суммами возможностей хранения и поршня. Они не будут заменять рабочий стол или ноутбук в ближайшее время. Для тех, кто хочет больше «физической привлекательности» в их процессорах, люди в ПИ Малины развивали очень дешевое (~ 45$) правление, которое основано на процессорах ARM, который имеет намного больше памяти и использует микро SD для постоянного хранения данных. Эти устройства управляют вариантом операционной системы Linux. Я не собираюсь говорить далее о ПИ Малины, как это находится в классе «компьютера» в противоположность микропроцессору.

Они микро диспетчер и архитектура великие и всегда будут местом для них. Однако есть выгода …, и это общается через Интернет. Эти устройства имеют удивительный набор возможностей включая прямые электрические входы и выходы (GPIOs) и поддерживают для множества протоколов включая SPI, I2C, UART и больше, однако, ни один из них до сих пор не идет с включенной беспроводной сетью.

Никакой вопрос (в моем уме), что Ардуино привлек общее внимание. Ардуино основан на жареном картофеле Atmel и имеет множество физических размеров в его открытых следах аппаратных средств. Основной микро диспетчер использовал, ATmega328. Можно найти случаи этих сырых процессоров на eBay за менее чем 2$ с полностью построенными досками, содержащими их за менее чем 3$. Это в 10-20

раз более дешево, чем ПИ Малины. Конечно, каждый получает существенно меньше, чем ПИ Малины, таким образом, сравнение может стать странным … однако, если то, что каждый хочет сделать, переделывают электронику или делают некоторые простые устройства, которые соединяются со светодиодами, выключателями или датчиками, тогда функциональные особенности должны были стать ближе.

Между ними у Ардуино и ПИ Малины, кажется, есть все удовлетворенные потребности. Если бы это имело место, то это было бы очень короткой книгой. Давайте добавим поворот, который мы начали с … беспроводной сети. Чтобы иметь устройство перемещают шасси робота или светодиодные образцы вспышки или издают некоторый шум или читают данные из датчика и звукового сигнала, когда температура получает слишком высокий …, это все замечательные и достойные проекты. Однако мы все очень знаем о ценности Интернета. Наши компьютеры - связанный Интернет, наши телефоны связаны, мы смотрим телевизор (Netflix) по Интернету, мы играем в игры по Интернету, мы социализируем (??) по Интернету … и так далее. Интернет стал

такой предмет первой необходимости, что мы смеялись бы, если бы кто-то предложил нам новый компьютер или телефон, который испытал недостаток в способности пойти «онлайн».

Теперь вообразите то, что микро диспетчер с родным беспроводным Интернетом мог сделать для нас? Это было бы процессором, который мог запустить приложения, а также или лучше, чем Ардуино, который будет иметь GPIO и поддержку протокола аппаратных средств, имел бы RAM и флэш-память …, но будет иметь новую возможность убийцы, что это также было бы в состоянии сформировать Подключения к Интернету. И это, … просто помещают …, - каково устройство ESP8266. Это - альтернативный микропроцессор тем уже упомянутым, но также и имеет WiFi и TCP/IP (протокол TCP/IP), поддержка уже встроила. Кроме того, это также не намного более дорого, чем Ардуино. Ища eBay, мы находим правления ESP8266 менее чем 3$.

ESP8266

ESP8266 - имя микро диспетчера, разработанного Espressif Systems. Espressif - китайская компания, базирующаяся из Шанхая. ESP8266

рекламирует себя как отдельное сетевое решение WiFi, предлагающее себя как мост от существующего микро диспетчера к WiFi …, и … также способен к управлению сам содержавшие заявления.

Производство объема ESP8266 не начиналось до начала 2014, что означает, что в схеме вещей это - совершенно новый вход в очереди процессоров. И … в нашей технологии голодный мир, новый обычно, равняется интересному. Спустя несколько лет после производства IC, 3ул. партийных OEMs берут этот жареный картофель и строят «доски резкого изменения цен на бумаги» для них. Если бы я должен был вручить Вам сырой ESP8266 прямо из фабрики, маловероятно, что мы знали бы, что сделать с одним. Они очень крошечные и фактически невозможные для людей, увлеченных своим хобби, приложить провода, чтобы позволить им быть включенными в макеты. К счастью, они оптовые закупки OEMs ICs, проектируйте принципиальные схемы, проектируйте печатные платы и постройте предварительно сделанные доски с ICs пред - приложенный немедленно готовый к нашему использованию. Именно эти правления захватывают наш интерес и что мы можем купить за несколько долларов на eBay.

Есть множество доступных стилей правления. Двум, что я собираюсь сосредоточиться на, дали ESP-1 имен и ESP-12. Важно отметить, что есть только один процессор ESP8266, и именно этот процессор найден на ВСЕХ правлениях резкого изменения цен на бумаги. Что различает, одно правление от другого - количество выставленных булавок GPIO, сумма флэш-памяти если, стиль булавок соединителя и различных других соображений, связанных со строительством. С программной точки зрения они все одинаковые.

Страница 26

Зрелость

ESP8266 - новое устройство на арене. Это было вокруг с тех пор только лета 2014 года, но уже отправляло объемы производства в десятках миллионов. Все и все должны начать где-нибудь. Это означает, что есть совершенно новое богатство территории, которая будет исследоваться и новые возможности и функции и образцы использования, которые будут обнаружены. На вниз стороне, у этого еще нет богатства обучающих программ, образцов и видео, которые сопровождают другие

микро системы диспетчера. Его документация не блестящая, и некоторые основные вопросы на его использовании все еще исследуются. То, как это сидит с Вами, является функцией Вашего намерения в лужении в этой области. Если Вы захотите идти по путям, которые сопровождались много раз прежде, то другие процессоры будут более привлекательными. Однако, если Вам нравятся страсть к приключениям и принятие участие в «первом этаже» только что прибывшего, проблемы, которые мы (сообщество ESP8266) пытаемся решить, могут активно взволновать Вас, а не отговорить Вас.

Это - также основная причина, что люди как я тратят многих, много часов, учась и документируя то, что мы находим …, таким образом, другие могут, надо надеяться, основываться на том, что было изучено без изобретения велосипед.

Мог волнение о фиаско процессоров ESP8266? Да … эти устройства могут просто быть подарком судьбы и несколько лет с этого времени, человек, увлеченный своим хобби, не уделит долгое внимание о них. Но то, что я спрашиваю Вас, должно приблизиться к устройству без предубеждения.

Спецификация ESP8266

Когда мы приближаемся к новому устройству электроники, нам нравится знать о его спецификации. Вот некоторые важные моменты:

Напряжение	3.3 В
Текущее потребление	10uA – 170mA
Присоединяемая флэш-память	16 МБ макс. (512K нормальный)
Процессор	Tensilica L106 32 бита
Скорость процессора	80-160MHz
RAM	32K + 80K
GPIOs	17 (мультиплексированный с другими функциями)
Аналог к цифровому	1 вход с 1 024 шагами (10 битов) резолюция
802,11 поддержки	b/g/n/d/e/i/k/r
Максимальные параллельные соединения по протоколу TCP	5

Вопросом определения, сколько времени ESP8266 может работать от батарей, является интересный. Текущее потребление совсем не постоянное. Передавая в полную силу, это

может потреблять 170mA, но когда в глубоком сне, этому только нужно 10uA. Это - настоящее различие. Это означает, что время выполнения ESP8266 на фиксированном текущем водохранилище не просто функция времени, но также и того, что оно делает в течение того времени …, и это - функция программы, развернутой на него.

ESP8266 разработан, чтобы использоваться с модулем памяти партнера, и это - обычно флэш-память. Большинство модулей идет с некоторой вспышкой, связанной с ними. Поймите, что у вспышки есть конечное количество, стирает за страницу, прежде чем что-то потерпит неудачу. Они оценены приблизительно в 10 000, стирает. Это обычно не проблема для изменения конфигурации, пишет, или ежедневная регистрация пишет …, но если Ваше заявление все время пишет новые данные чрезвычайно быстро, то это может быть проблемой, и Ваша флэш-память выйдет из строя.

Модули ESP8266

Интегральная схема ESP8266 прибывает в небольшой пакет, возможно, пятимиллиметровый квадрат. Очевидно, если Вы не первоклассный паяльщик, Вы не собираетесь делать много с этим. Хорошие новости - то, что много продавцов создали правления резкого изменения цен на бумаги, которые делают работу намного легче для Вас. Здесь мы перечисляем некоторые более общие модули.

ESP-12

Текущую самую популярную и гибкую конфигурацию, доступную сегодня, называют ESP-12. Это выставляет большинство булавок GPIO для использования. Основному модулю ESP-12 действительно нужен его собственный модуль расширителя, чтобы сделать это макетом и 0,1-дюймовым правлением полосы дружественный.

Вот то, на что похоже устройство ESP-12, когда оно установлено на правлении расширителя макета:

Булавка из правления расширителя смотрит следующим образом:

Страница 28

ESP-12 установили синюю поверхность, Вовлек верхний правый угол. Этот светодиод вспыхивает, когда есть трафик UART.

Вот описание различных булавок:

Имя:	**Описание**
VCC	3.3 В.
GPIO 13	Также используемый для SPI MOSI.
GPIO 12	Также используемый для МИСО SPI.
GPIO 14	Также используемый для Часов SPI.
GPIO 16	
CH_PD	Чип позволяет. Должно быть высоким для нормального функционирования. • 0 – Отключенный • 1 – Позволенный
ADC	Аналог к цифровому входу
ОТДЫХ	Внешний сброс. • 0 – Сброс • 1 – Нормальный

TXD	UART 0 передают.
RXD	UART 0 получают.
GPIO 4	Регулярный GPIO.
GPIO 5	Регулярный GPIO.
GPIO 0	Должно быть **высоким** на ботинке, низко для обновления вспышки.
GPIO 2	Должно быть **высоким** на ботинке.
GPIO 15	Должно быть **низким** на ботинке и вспышке.
GND	Земля.

Вот схематическое для соединения случая:

Затем мы видим изображение этой схемы, пристроенной на макете.

Если мы просто хотим использовать наше правление резкого изменения цен на бумаги, у нас есть следующее, когда мы установлены на макете, у нас может быть следующая установка:

Это дает нам два набора 8 соединителей булавки. Первый набор:

Установите 1	
Булавка	**Цвет**
GND	Оранжевый
GPIO15	Желтый
GPIO2	Зеленый
GPIO0	Синий
GPIO5	Фиолетовый
GPIO4	Серый
RXD	Белый
TXD	Черный

Второй набор:

Установите 2	
Булавка	**Цвет**
VCC	Оранжевый
GPIO13	Желтый
GPIO12	Зеленый
GPIO14	Синий

GPIO16	Фиолетовый
CH_PD	Серый
ADC	Белый
ОТДЫХ	Черный

ESP-1

Правление ESP-1 - ESP8266 на 8 правлениях булавки. Это нисколько не дружественный макет, но к счастью мы можем сделать адаптеры для него чрезвычайно легко. ESP-1 был одним из первых доступных правлений и кроме цены, должен почти быть обесценен. Это только обеспечивает маленькое подмножество схем контактов, обеспеченных другими правлениями. Однако ESP-1 был в широком масштабе произведенным объемом и все еще поддерживает один из самых маленьких физических следов. Если Вы только нуждаетесь в нескольких I/Os или нуждаетесь в очень крошечном размере, в этом выборе может все еще быть некоторая стоимость.

Булавка из устройства следующие:

Функция	Цвет	Описание
TX		Передать
RX		Получить. **Всегда** используйте конвертер уровня для поступающего данные. Это устройство **не** составляет терпимых 5 В.
CH_PD		Чип позволяет. Должно быть высоким для нормального функционирования.

		• 0 – Отключенный • 1 – Позволенный
RST		Внешний сброс. • 0 – Сброс • 1 – Нормальный
GPIO 0		Должно быть **высоким** на ботинке, низко для обновления вспышки.
GPIO 2		Должно быть **высоким** на ботинке.
VCC		3.3 В
GND		Земля

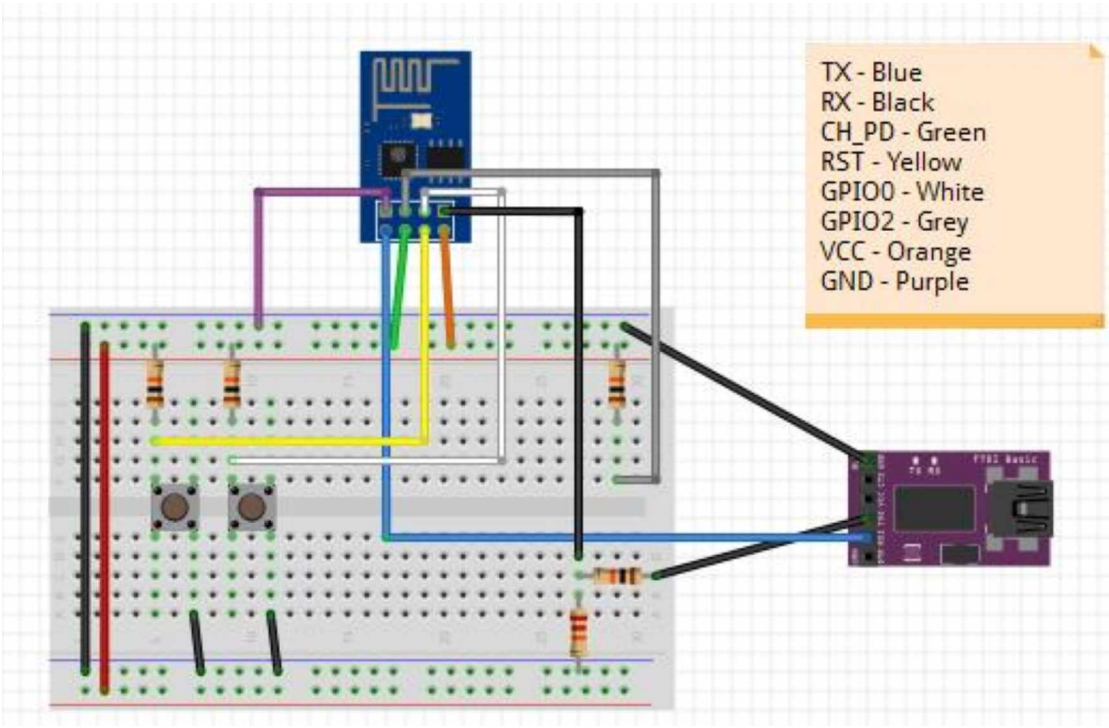
Простую схему показывают ниже. Обратите внимание, что TX и булавки RX показывают **не** связанные. Не забудьте **всегда** использовать конвертер уровня для булавки RX в устройство, поскольку это не терпимых 5 В.

вот альтернативная схема:

Вот схема на макете, который был продемонстрирован, чтобы работать просто великолепно.

Страница 34

Если мы хотим добавить кнопки основания для СБРОСА, и GPIO 0, следующее - некоторые схемы:



Страница 35

Когда мы нажимаем кнопку сброса, она имеет смысл для этого только, чтобы быть мгновенной прессой. Вот схема для этого:

Страница 36

Дефолт последовательная скорость соединения, кажется, 115200.См. также:

- YouTube:детали Булавки ESP8266 ESP-01, Начиная
- YouTube:ESP8266 ESP-01 и USB к последовательным связям конвертера (CP2102 Silicon Labs)

Adafruit УРА

Adafruit HAZZAH - управление резкого изменения цен на бумаги по ESP8266. Это - большая часть макета, дружественного из решений, с которыми я столкнулся до сих пор.

См. также:

- Adafruit УРА

NodeMCU devKit

Этот модуль идет с построенным в соединителе USB и богатом ассортименте схем контактов. Это - также немедленно дружественный макет (если Вы колеблетесь между двумя правлениями).

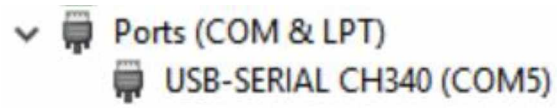
Отображение булавки на этом устройстве следующие:

Страница 38

Есть в настоящее время два аромата совета NodeMCU, названного v0.9 и v1.0. Вот первичные различия:

Функция	NodeMCU v0.9	NodeMCU v1.0
USB	CH340 базировался	CP2012 базировался
ESP8266	ESP-12E	ESP-12E

Когда он связан через USB на машине Windows, Последовательный соединитель показывает «ПОСЛЕДОВАТЕЛЬНЫМ USB CH340».



Страница 39

Если по некоторым причинам USB, который не разоблачает последовательный соединитель, ищут Интернет водителей для CH340G для операционной системы Вашего PC.

Это устройство не особенно единственный дружественный макет (хотя это может оставить один ряд выставленных булавок), но соответствует красиво на двух макетах, которые являются рядом. Я рекомендую добавить второй USB→UART, связанный следующим образом:

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

Кроме того, кнопка между GND и RST также обеспечит способность к сбросу.См. также:

- Домашняя страница NodeMCU
- GitHub: nodemcu/nodemcu-devkit-v1.0
- GitHub: nodemcu/nodemcu-devkit
- Доктора NodeMCU LUA
- ESP8266: NodeMCU Dev Kit V1.0 Review

узел. IT (иначе ESP-210)

См. также:

- ESP-210

Щит SparkFun WiFi – ESP8266

SparkFun произвели щит WiFi для Ардуино. Это - ESP8266, установленный на хорошо разработанном PCB что помощники с Ардуино. Это делает общение с ESP8266 через В командах чрезвычайно легким без проводки необходимого. Просто выдвиньте правление щита в гнезда Ардуино, и Вы сделаны.

См. также:

- Щит SparkFun WiFi – ESP8266

Облегченный эспрессо

См. также:

- Облегченный эспрессо

Wemos D1

Wemos D1 обеспечивает, Uno Ардуино разработал правление вместе с несколькими вариантами власти и женскими ударами головой в обеих сторонах правления. Если Вы - более удобная работа над измеренной платформой Uno Ардуино, это делает хорошего кандидата.

См. также:

- Wemos D1

Дуб digistump

См. также:

- Дуб digistump

Соединение с ESP8266

ESP8266 - устройство WiFi, и следовательно мы в конечном счете соединим с ним использование протоколы WiFi, но некоторая самонастройка требуется сначала. Устройство не знает что сеть соединиться с, который пароль использовать и другие необходимые параметры. Это, конечно, предполагает, что мы соединяемся как станция, если мы хотим, чтобы устройство было точкой доступа, или мы хотим загрузить наши собственные заявления в него, история становится глубже. Это подразумевает, что есть некоторый способ взаимодействовать с устройством кроме WiFi и есть …, ответ - (Последовательный) UART. У ESP8266 есть выделенное взаимодействие UART с маркированным TX и RX булавок. Булавка TX - передача ESP8266 (за границу от ESP8266), и булавка RX используется, чтобы получить данные (прибывающий в ESP8266). Эти булавки могут быть связаны с партнером UART. Безусловно самый легкий и самый удобный партнер для нас - USB → UART конвертер. Они обсуждены подробно позже в книге. На данный момент давайте предположим, что мы настроили их. Через UART мы можем приложить предельный эмулятор, чтобы послать нажатия клавиш и получить данные от ESP8266, показанного как знаки на экране. Это используется экстенсивно, работая с В командах. Вторая цель UART состоит в том, чтобы получить двоичные данные, используемые, чтобы «высветить» флэш-память устройства, чтобы сделать запись новых заявлений на выполнение. Есть множество технических инструментов в нашем распоряжении, чтобы достигнуть той задачи.

Когда мы используем UART, мы должны рассмотреть понятие скорости передачи в бодах. Это - скорость коммуникации данных между ESP8266 и его партнером. Во время ботинка ESP8266 пытается автоматически определить скорость передачи в бодах партнера и соответствовать ему. Это принимает дефолт 74 880 и если Вам приложат последовательный терминал, то Вы будете видеть сообщение как:


```
ets 8 января 2013, rst cause:2, загружают способ: (1,0)
```

если это настроено, чтобы получить в 74 880.

У ESP8266 есть второй UART, связанный с ним, который произведен только. Одна из основных целей этого второго UART состоит в том, чтобы произвести диагностику и отладочную информацию. Это может быть чрезвычайно полезно во время развития и как таково, я рекомендую приложить **два** USB → UART конвертеры к устройству. Второй UART мультиплексирован с булавкой GPIO2.

См. также:

- USB к конвертерам UART
- ПРИ программировании команды
- Погрузка программы в ESP8266

Теория WiFi

Когда работа с WiFi ориентировала устройство, важно, чтобы у нас было по крайней мере некоторое понимание понятий, связанных с WiFi. На высоком уровне WiFi - способность участвовать в связях TCP/IP по связи радиосвязи. WiFi - конкретно набор протоколов, описанных в архитектуре LAN Радио IEEE 802.11.

В рамках этой истории устройство назвало Точку доступа (точка доступа или AP) действиями как центр всех коммуникаций. Как правило, это связано с (или действует как) как маршрутизатор TCP/IP к остальной части сети TCP/IP. Например, в Вашем доме, Вам, вероятно, соединят точку доступа WiFi с Вашим модемом (кабель или DSL). Связи WiFi тогда сформированы к точке доступа (через устройства, названные станциями) и транспортные потоки TCP/IP через точку доступа к Интернету.

Устройства, которые соединяются с точками доступа, называют «станциями»:

Устройство ESP8266 может играть роль Точки доступа, Станции или обоих одновременно.

Очень обычно точка доступа также имеет сетевое подключение с Интернетом и действует как мост между беспроводной сетью и более широкой сетью TCP/IP, которая является Интернетом.

Коллекцию станций, которые хотят общаться друг с другом, называют Basic Service Set (BSS). Общая конфигурация - то, что известно как Инфраструктура BSS. В этом способе все коммуникации, прибывающие и за границу с отдельной станции, разбиты через точку доступа.

Станция должна связать себя с точкой доступа, чтобы участвовать в истории. Станция может только быть связана с единой точкой доступа в любой момент.

У каждого участника сети есть уникальный идентификатор, названный MAC-адресом. Это - 48-битная стоимость.

Когда у нас есть несколько точек доступа в беспроводном диапазоне, станция должна знать с который соединиться. У каждой точки доступа есть сетевой идентификатор, названный BSSID (или чаще всего просто SSID). SSID - **сервисный идентификатор набора**. Это - 32 стоимости характера, которые представляют цель пакетов информации, посланной по сети.

См. также:

- Википедия –Точка доступа
- Википедия –IEEE 802.11
- Википедия –WiFi защищенный доступ
- Википедия –IEEE 802.11i-2004

ПРИ программировании команды

Самый быстрый и самый легкий способ начать с ESP8266 состоит в том, чтобы получить доступ к нему через В интерфейсе команды.

Когда мы думаем об устройстве ESP8266, мы находим, что у него есть построенный в UART (Последовательная) связь. Это означает, что может и послать и получить данные, используя протокол UART. Мы также знаем, что устройство может общаться с WiFi. Что, если у нас было применение, которое работало на ESP8266, который взял «инструкции», полученные по последовательной связи, выполнило их и затем возвратило ответ? Это тогда позволило бы нам использовать ESP8266, никогда не имея необходимость знать языки программирования, которые являются родными к устройству. Это точно, что программа, которая, как до сих пор находили, была предварительно установлена на ESP8266, делает для нас. Программу называют «В процессоре команды», названном в честь формата команд, посланных через последовательную связь. Эти команды все снабжены префиксом «В» и следуют (примерно) за стилем, известным как «набор команд Хейза».

Страница 44

Если мы думаем о применении, желающем использовать услуги ESP8266 как клиент и ESP8266 как сервер, способный к обслуживанию тех команд как сервер, то клиент посылает ряды знаков через связь UART с сервером, и сервер отвечает результатом.

Espressif издают полный комплект В документации команды, которая может быть найдена на их странице форума в:

- http://espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf

Есть два основных документа:

- ESP8266EX В наборе команд
- ESP8266EX В примерах команды

Команды

Когда каждый телеграфировал ESP8266 к последовательному конвертеру, следующий вопрос будет, «Он работает?». Когда мы соединяем последовательный монитор, первая команда, которую мы можем послать, «B» котором должен ответить простым «хорошо».

Инструкция, переданная к устройству, следует за одним из следующих вариантов синтаксиса:

Напечатать	**Формат**	**Описание**
Тест	B + <x> =?	Подвергните сомнению параметры и его диапазон ценностей.
Вопрос	B + <x>?	Возвратите текущую стоимость параметра.
Набор	B + <x> = <...>	Установите значение параметра.
Выполнить	B + <x>	Выполните команду.

Все «B» инструкциях заканчиваются «\r\n» парой.

Команда	Описание
B	Прибыль хорошо
AT+RST	Перезапустите ESP8266.
AT+GMR	Версия встроенного микропрограммного обеспечения прибыли и для В процессоре команды и для SDK в использовать. В настоящее время ответ возвратился, похож: В version:0.21.0.0 SDK version:0.9.5
AT+GSLP = <время>	Поместите устройство в глубокий сон какое-то время в миллисекундах. Это проснется после этот период.
ПОЕЛ [0|1]	Эхо В командах. • ATE0 – Эхо командует прочь • ATE1 – Эхо командует на
AT+RESTORE	Восстановите дефолты параметров настройки во флэш-памяти.
AT+UART_CUR = <baudrate>, <databits>, <stopbits>, <паритет>, <поток контроль>	databits может быть 5, 6, 7 или 8. паритет может быть 0=none, 1=odd, 2=even flowcontrol может быть: 0 – отключить 1 – позвольте PTC 2 – позвольте CTS

	3 – позвольте и PTC и CTS
AT+UART_DEF = <baundrate> , <databits>, <stopbits>, <паритет>, <управление потоками>	
AT+SLEEP?	
AT+SLEEP = <режим ожидания>	
AT+RFPOWER = <власть TX>	
AT+RFVDD?	
AT+RFVDD = <VDD33>	
AT+RFVDD	
WI-FI	
AT+CWMODE_CUR = <способ>	Устанавливает текущий режим работы. • 1 – Станционный способ • 2 – способ AP • 3 – AP + Станционный способ

AT+CWMODE_CUR?	Получите текущий режим работы.
AT+CWMODE_CUR =?	Получите список доступных способов.
AT+CWMODE_DEF = <способ>	Устанавливает текущий режим работы. • 1 – Станционный способ • 2 – Способ AP • 3 – AP + Станционный способ
AT+CWMODE_DEF?	Получите текущий режим работы.
AT+CWMODE_DEF =?	Получите список доступных способов.
AT+CWJAP_CUR = <ssid> , <пароль> [<bssid>]	Присоединитесь к сети WiFi (ЯПОНЕЦ = Точка доступа Соединения).
AT+CWJAP_CUR?	Получите текущую информацию о связи.
AT+CWJAP_DEF = <ssid> , <пароль> [<bssid>]	Присоединитесь к сети WiFi (ЯПОНЕЦ = Точка доступа Соединения).
AT+CWJAP_DEF?	Получите текущую информацию о связи.

AT+CWLAP	Перечислите «Точки доступа Списка». Ответ: + CWLAP: <ecn>, <ssid>, <rssi>, <Mac>, <ch> где: ecn • 0 – ОТКРЫТЫЙ ◦ ◦ 1 – WEP ◦ 2 – WPA_PSK ◦ 3 – WPA2_PSK ◦ 4 – WPA_WPA2_PSK • ssid – SSID AP • rssi – сила Сигнала • Mac – MAC-адрес • ch – Канал
AT+CWLAP = <ssid> , <Mac>, <ch>	Перечислите фильтрованный набор точек доступа.
AT+CWQAP	Разъедините от AP.
AT+CWSAP_CUR?	Конфигурация softAP способа
AT+CWSAP_CUR = <ssid>, <pwd>, <chl>, <ecn>	
AT+CWSAP_DEF?	Конфигурация softAP способа

AT+CWSAP_DEF = <ssid>, <pwd>, <chl>, <ecn>	
AT+CWLIF	Список IP соединился в softAP способе
AT+CWDHCP_CUR?	
AT+CWDHCP_CUR = <способ> <en >	Позвольте или отключите DHCP. спос • об ◦ 0 – softAP ◦ 1 – станция ◦ 2 – softAP + станция • en 0 – Позволить ◦ ◦ 1 – Отключить
AT+CWDHCP_DEF?	
AT+CWDHCP_DEF = <способ> <en	Позвольте или отключите DHCP.

>	спос • об ◦ 0 – softAP ◦ 1 – станция ◦ 2 – softAP + станция • en 0 – Позволить ◦ ◦ 1 – Отключить
AP+CWAUTOCONN = <позволяют>	
AT+CIPSTAMAC_CUR?	Установите/получите MAC-адрес станции.
AT+CIPSTAMAC_CUR = <Mac>	Установите/получите MAC-адрес станции.
AT+CIPSTAMAC_DEF?	Установите/получите MAC-адрес станции.
AT+CIPSTAMAC_DEF = <Mac>	Установите/получите MAC-адрес станции.
AT+CIPAPMAC_CUR?	Установите/получите MAC-адрес softAP.
AT+CIPAPMAC_CUR = <Mac>	Установите/получите MAC-адрес softAP.
AT+CIPAPMAC_DEF?	Установите/получите MAC-адрес softAP.
AT+CIPAPMAC_DEF = <Mac>	Установите/получите MAC-адрес softAP.
AT+CIPSTA_CUR = <IP>	Установите IP-адрес станции.
AT+CIPSTA_CUR?	Получите IP-адрес станции. Например: +CIPSTA: «0.0.0.0»

Страница 49

AT+CIPSTA_DEF = <IP>	Установите IP-адрес станции.
AT+CIPSTA_DEF?	Получите IP-адрес станции. Например: +CIPSTA: «0.0.0.0»
AT+CIPAP_CUR?	Установите IP-адрес softAP.
AT+CIPAP_CUR = <IP> [<револьвер eway>, <netmask>]	Установите IP-адрес softAP.
AT+CIPAP_DEF?	Установите IP-адрес softAP.
AT+CIPAP_DEF = <IP> [<револьвер eway>, <netmask>]	Установите IP-адрес softAP.
AT+CIFSR	Возвращает IP-адрес ворот и IP-адрес.

Организация сети TCP/IP	
AT+CIPSTATUS	Информация о связи. Формат ответа: СТАТУС: <статистика> + CIPSTATUS: <id>, <тип>, <addr>, <порт>, <tetype> ста тис тик • а ◦ 2 – получил IP ◦ 3 – Связанный ◦ 4 – Разъединенный • id – Id связи • напечатайте – TCP или UDP • addr – IP-адрес • порт – Номер порта • tetype ◦ 0 – ESP8266 бежит как клиент ◦ 1 – ESP8266 бежит как сервер
AT+CIPSTART = <тип>, <addr>, <порт> [<местный порт>, <способ>]	Начните связь когда CIPMUX=0. • напечатайте – TCP или UDP • addr – Отдаленный IP-адрес • порт – Отдаленный порт • местный порт – Для UDP только • способ – Для UDP только ◦ 0 – предприятие ровесника назначения UDP зафиксировано ◦ 1 – предприятие ровесника назначения может измениться однажды ◦ 2 – предприятие ровесника назначения может измениться
AT+CIPSTART = <id>, <тип>, <addr>,	Начните связь когда CIPMUX=1. • id – ценность 0-4 связи

<порт> [<местный порт>, <способ>]	• напечатайте – TCP или UDP • addr – Отдаленный IP-адрес • порт – Отдаленный порт • местный порт – Для UDP только • способ – Для UDP только ◦ 0 – предприятие ровесника назначения UDP зафиксировано ◦ 1 – предприятие ровесника назначения может измениться однажды

	◦ 2 – предприятие ровесника назначения может измениться
AT+CIPSTART =?	???
AT+CIPSEND = <длина>	Пошлите знаки длины.
AT+CIPCLOSE	Закройте связь.
AT+CIFSR	Получите местный IP-адрес.
AT+CIPMUX = <способ>	Позвольте многочисленные связи. • 0 – Единственная связь. • 1 – Многочисленные связи.
AT+CIPMUX?	Возвращает текущую стоимость для CIPMUX. • 0 – Единственная связь. • 1 – Многочисленные связи.
AT+CIPSERVER = <способ> [<p орт>]	Настройте как сервер TCP. Если никакой порт не снабжен, дефолт 333. Сервер может только будьте созданы, когда CIPMUX=1 (позволяют многочисленные связи). • способ ◦ 0 – Удаляют сервер (нуждается в перезапуске после), ◦ 1 – Создают сервер
AT+CIPMODE = <способ>	Установите способ передачи. • 0 – Нормальный способ. • 1 – Неприкрашенный способ.
AT+CIPSTO = <время>	Перерыв сервера набора. Стоимость в диапазоне 0 – 7 200 секунд.
AT+CIUPDATE	???

См. также:

- YouTube –обучающая программа ESP8266 В командах

Установка последнего В процессоре команды

Последнее В процессоре команды может всегда загружаться в двухчастной форме с веб-сайта Espressif. Всегда рассматривайте README, который идет с файлами, и следуйте инструкциям, содержавшим в. Чтобы загрузить микропрограммные изображения, Вам будет нужен а

flashtool. Кроме того, Вам будут нужны файлы, поставляемые

загрузкой. С v0.50 необходимые файлы и адреса, которые будут загружены в:

Файл	Адрес
nonboot/eagle.flash.bin	0x00000
nonboot/eagle.irom0text.bin	0x40000
blank.bin	0x3E000
blank.bin	0x7E000

Эти инструкции для жареного картофеля вспышки 512K.

См. также:

- Espressif –Загрузка В процессоре команды – V0.50

Сборка схем

Так как ESP8266 - фактический электронный компонент, некоторая физическая сборка требуется. Эта книга не попытается касаться non-ESP8266 электроники, поскольку это - очень большой и широкий предмет самостоятельно. Однако то, что мы сделаем, описывают некоторые компоненты, которые мы нашли чрезвычайно полезными, строя решения ESP8266.

USB к конвертерам UART

Вы не можете программировать ESP8266, не поставляя его данные через UART. Самый легкий способ достигнуть этого с помощью USB к конвертеру UART. Я использую устройства, которые основаны на CP2102, который может быть найден дешево на eBay за менее чем 2$ каждым. Другой популярный бренд - устройства от Future Technology Devices International (FTDI). Вы захотите по крайней мере два. Один для программирования и один для отладки. Я предлагаю покупать больше чем два на всякий случай …

Заказывая, не забывайте получать некоторые кабели расширителя USB наружной и внутренней нарезки, поскольку маловероятно, что Вы будете в состоянии приложить свои USB-устройства и к макету и к PC одновременно через прямую связь и хотя кабели соединителя будут работать, включение макета именно так намного легче. Кабели

соединителя USB позволяют Вам легко соединяться с PC на гнездо USB к штепселю USB UART. Вот изображение

Страница 52

тип кабеля соединителя я рекомендую. Получите их с максимально короткой длиной кабеля. 12-24 дюйма должны быть предпочтены.

Когда мы включаем USB → UART в машину Windows, мы можем изучить COM-порт, что новый последовательный порт появляется на, открывая Windows Device Manager. Есть много способов сделать это, один путь состоит в том, чтобы начать его из окна команды DOS с:

```
mmc devmgmt.msc
```

Согласно разделу под названием Порты (COM & LPT) Вы найдете записи для каждого из COM-портов. COM-порты не предоставляют Вам отображение, что конкретное гнездо USB принимает конкретный COM-порт, таким образом, мое плохое предложение состоит в том, чтобы вынуть USB из каждого гнезда один за другим и обратить внимание, какой COM-порт исчезает (или появляется, если Вы вставляете USB).

См. также:

- Соединение с ESP8266
- Работа с сериалом

Макеты

Я нахожу, что у меня никогда не может быть слишком многих макетов. Я

предлагаю получить некоторых полный размер и половина правлений размера наряду с приблизительно 24 проводами соединителя AWG и хорошей парой инструментов для снятия изоляции. Держите мусорное ведро мусора рядом с иначе, Вы найдете себя по колено в раздетой изоляции и сокращать проводные части, прежде чем Вы будете знать это. Я также рекомендую некоторому мужскому мужчине Дюпона, предварительно сделанному проводами. Кабель ленты может также быть полезным.

Власть

Нам нужно электричество, чтобы получить эти устройства работа. Я выбираю макет MB102 присоединяемые адаптеры питания. Они могут быть приведены в действие от обычной стенной бородавки (адаптер сети) или от USB. Кажется, что штепсель для стены - власть бородавки составляет 2.1 мм и центр, уверенный однако, что я настоятельно рекомендую, что Вы читаете технические спецификации своего определенного поставщика очень тщательно. Есть также потенциальная озабоченность, что гнездо барреля телеграфировано параллельно с входом USB, который мог означать что, если Вы прилагаете высоковольтный вход (например, 12 В), также соединяя источник USB, Вы можете повредить свое USB-устройство. У устройств есть основной выключатель питания включения - выключения плюс прыгун, чтобы установить 3.3-вольтовую или 5-вольтовую продукцию. У Вас может

даже быть один рельс макета быть 3.3 В и другие 5 В …, но заботиться, чтобы не применить 5 В к Вашему ESP8266. При наличии двух рельсов власти, один на уровне 3.3 В и другого на уровне 5 В, Вы можете привести в действие и ESP8266 и устройства/схемы, которые требуют 5 В.

Когда ESP8266 начинает передавать по радио, которое может потянуть много тока, который может вызвать рябь в Вашем источнике питания. У Вас могут также быть другие датчики или устройства, связанные с Вашей поставкой также. Эти колебания напряжения могут вызвать проблемы. Сильно рекомендуется, чтобы Вы поместили, микро 10 жили конденсатор между +ve и-ve как близко к Вашему ESP8266, как Вы можете. Это обеспечит водохранилище права выровнять любую переходную рябь. Это - одна из тех подсказок, что Вы игнорируете в своей опасности. Все может работать просто великолепно без конденсатора …, пока он не делает или пока Вы не начинаете получать неустойчивые проблемы и затрудняетесь объяснять их. Позвольте мне поместить его этот путь за несколько центов, которых это стоит и нулевой вред, который это причиняет, почему нет?

Мультиметр / Логика исследует / Логика Анализатор

Когда Ваш круг не работает, и Вы уставились на него задающийся вопросом что не так, Вы будете благодарны, если у Вас будут мультиметр и логическое исследование. Если Ваш бюджет будет простираться, я также рекомендую основанный на usb логический анализатор, такой как сделанные Saleae. Они позволяют Вам контролировать сигналы, входящие или производимые Вашим ESP8266. Думайте об этом как о лучшем источнике отладки доступного Вам.

См. также:

- Логика Saleae анализаторы

Различные компоненты

Вы захотите обычную группу подозреваемых для различных компонентов включая светодиоды, резисторы, конденсаторы и т.д.

Физическое строительство

Когда у Вас есть макет Ваш круг и письменный Ваше заявление, там может прибыть время, где Вы хотите сделать свое решение постоянным. В том пункте Вам будут нужны паяльник, припой и некоторое правление полосы. Я также рекомендую некоторые женские гнезда заголовка так, чтобы Вы не спаивали свой ESP8266s непосредственно в схемы. Мало того, что это позволяет Вам снова использовать устройства (должен Вы желать), но на несчастье, которое Вы жарите один, будет легче заменить.

Рекомендуемая установка для программирования ESP8266

Очевидно, чтобы программировать ESP8266, Вы должны будете на самом деле получить ESP8266, но это не настолько легко. Сам фактический ESP8266 - крошечная интегральная схема, и Вы вряд ли будете в состоянии использовать его непосредственно. Вместо этого Вы купите один из многих стилей правлений резкого изменения цен на бумаги, которые уже существуют. Общие - ESP-1, который выставляет 2 булавки GPIO и ESP-12, который выставляет 9. Я рекомендую ESP-12, поскольку это только незначительно более дорого для дополнительных выставленных булавок.

Вам также будет нужно повышающееся правление, поскольку у ESP-12 отдельно нет булавок соединителя. Вы можете обычно покупать и ESP-12 и повышающееся правление вместе одновременно. Однако проверьте тщательно, повышающиеся доски могут быть куплены отдельно, и Вы должны утвердить это, когда Вы заказываете и предполагаете, что получаете обоих, что Вы *только* покупаете повышающиеся доски без ESP8266. Вы будете разочарованы.

ESP-12 тогда спаян на повышающееся правление, таким образом, Вам будут нужны паяльник и некоторое мелкое ручное управление. Запаивание не является самым легким в мире, как булавки чрезвычайно близко друг к другу. Поэтому и для других, я предложил бы покупать несколько 12 ESP и установить доски вместо всего один. Также не трудно пожарить Ваш ESP-12, если Вы понимаете некоторую проводку превратно. После того, как собранный, это должно посмотреть следующим образом:

Мои никогда не смотрят, это «убирает», когда строят, поскольку моя смола припоя, кажется, обесцвечивает оригинальную привлекательную белую основу повышающегося правления. Однако взгляды не важны.

У принятия Вас теперь есть установленный ESP-12 с булавками, Ваш следующий вопрос будет «теперь что»? Это - то, где Вы захотите несколько макетов и провода соединителя. Вы могли использовать соединители Дюпона с женскими гнездами, приложенными к ESP-12 и

мужским булавкам на другом, чтобы быть свойственными Вашему макету, но Вы найдете, что провода неизбежно высвобождаются в худшие возможные времена. Вы можете установить ESP-12 к макету, но я склонен находить, что есть недостаточно пространства для проводов соединителя под ним.

После того, как обеспеченный, я рекомендую **два** USB → UART соединители. Почему два? Один посвященный для высвечивания устройства и один для отладки.

Для власти я рекомендую использовать источники питания макета MB102 однако, удостоверьтесь, что Вы устанавливаете кабели прыгуна составлять 3.3 В. Вы разрушите свой ESP8266, при попытке привести его в действие на уровне 5 В.

Как только это все обеспечено электричеством, Вам будет нужен PC с двумя открытыми USB-портами.

Список частей

- Макеты – 2 половины размера – 3,50$ для 2
- ESP-12 плюс повышающиеся правления – 3 набора – 3,80$ каждый – 11,40$
- CP2102 USB → UARTs – 2 части - 3,10$
- Мужчина USB к женским расширителям – 2 частям – 1,00$ каждый – 2,00$
- 24 провода AWG – 5 метров за 1,12$
- 8pin 2.54-миллиметровые наращиваемые длинные женские заголовки на ножках – 10 частей за 3,95$
- Красные разбросанные светодиоды – горстка – 1,00$
- Резисторы – Некоторый 10K, некоторый 20K, приблизительно 330 Ом – горстка A – 1,00$
- Конденсаторы – приблизительно 10 микро жили – 1,00$

Все сказали, это достигает приблизительно 30$ + некоторая доставка. Я покупаю все свои компоненты через eBay от китайских поставщиков, которые дают мне цену/качество, которую я ищу. Название игры, хотя

терпение. Как только Вы приказываете, чтобы обычно потребовалось 2-3 недели для частей, чтобы прибыть также - быть терпеливым и использовать время, чтобы посмотреть видео YouTube на проектах электроники и соответствующих общественных форумах.

В конечном счете Вы, вероятно, собираетесь хотеть построить постоянную схему для своего развития. На правлении полосы схема, которую я построил, похожа:

Страница 58

Конфигурация для высвечивания устройства

Позже в книге Вы найдете, что, когда она прибывает время, чтобы высветить устройство с Вашими новыми заявлениями, Вы должны будете установить некоторые булавки GPIO быть низкими и затем

перезагрузка. Это - признак, что это теперь готово быть высвеченным. Очевидно, Вы можете построить схему, которую Вы используете для высвечивания Вашего встроенного микропрограммного обеспечения и затем помещаете устройство в его заключительную схему, но Вы найдете, что во время развития, захотите высветить и проверить довольно часто. Это означает, что Вы захотите использовать провода прыгуна и позволить Вам перемещать связи булавок на Ваших макетах от их положения «вспышки» до их положения «нормальной эксплуатации».

Программирование

ESP8266 позволяет Вам писать заявления, которые могут бежать с рождения на устройстве. Вы можете собрать кодекс языка C и развернуть его к устройству посредством процесса, известного как высвечивание. Для Ваших заявлений сделать что-то полезное, они должны быть в состоянии взаимодействовать с окружающей средой. Это могло устанавливать сетевые связи или посылать/получать данные из приложенных датчиков, входов и выходов. Чтобы заставить это произойти, ESP8266 содержит основной набор функций, о которых мы можем свободно думать как операционная система устройства. Услуги операционной системы подвергнуты, чтобы быть названными из Вашего заявления, предоставляющего контракт на услуги, которые Вы можете усилить. Эти услуги полностью зарегистрированы. Чтобы успешно написать заявления на развертывание, Вы должны знать о

существовании этих услуг. Они становятся обязательными инструментами в Вашем ящике для инструментов. Например, если Вы должны соединиться с точкой доступа WiFi, есть API для этого. Чтобы получить Ваш текущий IP-адрес, есть API для этого и получить время, так как устройство было запущено, есть API для этого. На самом деле есть МНОГО API, доступных для нас, чтобы использовать. Хорошие новости - то, что никто не ожидает, что мы запомним все подробности их использования. Скорее достаточно широко знать, что они существуют и должны где-нибудь пойти, когда Вы хотите искать детали того, как использовать их.

Чтобы разумно управлять числом и множеством этих выставленных API, мы можем собрать наборы их вместе в значащих группах связанных функций. Это дает нам еще один и лучший способ управлять нашим знанием и приобретением знаний о них.

Основной источник знания о программировании ESP8266 является Руководством API ESP8266 SDK. Прямые связи со всеми соответствующими документами могут быть найдены вСправочныхдокументах.

См. также:

- Системы Espressif – производители ESP8266
- Электронная доска объявлений Espressif – Место для SDKs, докторов и форумов

Способ ботинка

Когда ESP8266 загружает, ценности булавок, известных как `MTDO`, `GPIO0` и `GPIO2` исследованы. Комбинация высоких или низких ценностей этих булавок предоставляет 3-битному числу в общей сложности 8 возможных ценностей от 000 до 111. Каждой стоимости интерпретировало возможное значение устройство, когда это загружает.

Стоимость [15-0-2]	Десятичное число Стоимость	Значение

000	0	Переотображение … неизвестные детали.
001	1	Ботинок от данных получен от UART0. Также включает высвечивание флэш-памяти для последующего нормальные запуски.
010	2	Запустить от внешнего источника
011	3	Ботинок от флэш-памяти
100	4	Низкая скорость SDIO V2
101	5	Высокая скорость SDIO V1
110	6	Низкая скорость SDIO V1
111	7	Высокая скорость SDIO V2

С практической точки зрения, что это означает, то, что, если мы хотим, чтобы устройство обычно работало, мы хотим загрузить от вспышки с булавками, имеющими ценности 011, в то время как, когда мы хотим высветить устройство с новой программой, мы хотим поставлять 001, чтобы загрузить от UART0.

Обратите внимание, что `MTDO` также известен как `GPIO15`.

ESP8266 – Software Development Kit (SDK)

Включайте справочники

Язык программирования C использует основанный на тексте препроцессор, чтобы включать данные в компиляцию. У препроцессора C есть способность включать дополнительные исходные файлы C, которые, в соответствии с конвенцией, называют заголовочными файлами и концом с «`.h`» префиксом. В рамках этих файлов мы обычно находим определения типов данных и прототипов функции, которые используются во время компиляции. ESP8266 SDK обеспечивает, названный справочник «`включают`», который содержит включать файлы, поставляемые Espressif для использования с ESP8266. Список заголовочных файлов, которые мы можем использовать, как описан в следующей таблице:

Файл	Примечания
at_custom.h	Определения для таможенных расширений к В укладчике команды.
c_types.h	Определения языка С.
eagle_soc.h	Определения низкого уровня и макрос. В большой степени связанный с битовым жонглированием на уровне центрального процессора. Никакая идея, почему файл называют «орлом».
espconn.h	TCP и определения UDP. У этого есть pre-reqs c_types.h и ip_addr.h.
espnow.h	Функции, связанные с ESP теперь, поддерживают.
ets_sys.h	Неизвестный.
gpio.h	Определения для взаимодействий GPIO.
ip_addr.h	Определения IP-адреса и макрос.
mem.h	Определения для манипуляции памяти и доступа.
os_type.h	Определения типа OS.
osapi.h	Включает поставляемый удар головой **пользователя**, названный «user_config.h».
ping.h	Определения для способности звона.
pwm.h	Определения для PWM.
queue.h	Очередь и определения списка.
smartconfig.h	Определения для умной конфигурации.
sntp.h	Определения для SNTP.
spi_flash.h	Определения для вспышки.
upgrade.h	Определения для модернизаций.
user_interface.h	Определения для OS и WiFi. У меня нет объяснения того, почему этот файл называют «user_interface» как нет, очевидно, никакого UI, связанного с ESP8266s.

Компилирование

Программный код для программы ESP8266 обычно пишется в С. Прежде чем мы сможем развернуть применение, мы должны собрать кодекс в двойные инструкции по машинному коду. Перед этим, хотя, давайте проведем несколько минут, думая о кодексе.

Мы пишем код, используя редактора и идеально редактора, который понимает язык программирования, на котором мы работаем. Эти редакторы оказывают помощь синтаксиса, окраску ключевого слова и даже контекстные предложения. Когда мы сохраняем наш введенный код, мы собираем его и затем развертываем его и затем проверяем его.

Этот цикл повторяется так часто, что мы часто используем продукт, который охватывает редактирование, компиляцию, выполнение и тестирование как интегрированное целое. Родовое название для такого продукта - «Интегрированная Среда разработки» или «IDE». Есть случаи их обоих сбор и свободный. В свободном лагере мое предпочтительное оружие - Затмение и Ардуино IDE.

Затмение IDE является чрезвычайно богатой и сильной окружающей средой. Первоначально написанный IBM, это было открыто, поставил много лет назад. Это осуществлено на Яве, что означает, что это бежит и ведет себя тождественно через все общие платформы (Windows, Linux, OSx). Природа Затмения - то, что оно спроектировано как серия расширяемых программных расширений. Из-за этого многие участники через многие дисциплины расширили окружающую среду, и это - теперь связная структура для примерно всего. Включенный в это соединение ряд программных расширений, которые, на совокупности, называют «C Инструменты Разработчиков» или «CDT». Если Вы берете Затмение скелета и добавляете CDT, у каждого теперь есть первый уровень C IDE. Однако, что не поставляет CDT (и на серьезном основании) фактические компиляторы C и сами связанные инструменты. Вместо этого каждый «определяет» инструменты, которые каждый хочет использовать для CDT, и CDT берет его оттуда.

Для нашей истории ESP8266 это означает, что, если мы можем найти (который мы можем) ряд C инструменты компилятора, которые берут источник C и производят набор из двух предметов Xtensa, мы можем использовать CDT, чтобы построить наши программы.

Сделать вещи более интересными, хотя, мы должны понять, что C не единственный язык, который мы можем использовать для строительства приложений ESP8266. Мы можем также использовать C ++ и собрание. Вы можете быть удивлены, что я упоминаю собрание, как это как низкий уровень, поскольку мы можем возможно добраться однако есть странные времена, когда нам нужно просто, что (к счастью редко) … особенно, когда мы понимаем, что в значительной степени программируем непосредственно к металлу. У библиотек Ардуино (например), есть по крайней мере один файл ассемблера.

Для физических типов файлов суффиксы использовали для другого файла, с которым мы столкнемся во время развития, включайте:

- .h – C и C ++ языковой заголовочный файл
- .c – исходный файл языка C
- .cpp – C ++ исходный файл
- .S – исходный файл Ассемблера
- .o – файл Объекта (собранный источник)
- .a – библиотека Архива

Чтобы выполнить компиляции, нам нужен ряд средств разработки.

Мое личное предпочтение - пакет для Затмения, которому предварительно построили все и готовый к употреблению. Однако эти инструменты могут также быть загружены с Интернета как общедоступные проекты на части основанием части.

Макро-МЕСТНЫЙ ЖИТЕЛЬ - синоним для «статичного» ключевого слова языка C.

От чтения докторов не был найден никакой изданный пример того, как собрать. Однако, когда каждый использует проект открытого источника Затмения, каждый видит Мэкефайлов, которые используются, и это выставляет примеры компиляции.

Типичная компиляция похожа:

```
17:57:16 **** Строит из Дефолта конфигурации для проекта k_blinky ****
Файл mingw32-make.exe-f
C:/Users/IBM_ADMIN/Documents/RaspberryPi/ESP8266/EclipseDevKit/WorkSpace/k_
blinky/Make все
CC
user/user
_main.c
AR
build/app
_app.a  LD
build/app
.out
------------------------------------------------------------------------
---
Информация о разделе:


build/app.out:    формат файла elf32-xtensa-le
                  Размер    VMA        LMA      Файл      Alg
Разделы:                                       прочь     n
Имя Idx
```

```
                                                    2
                                                    **
  0 .data         0000053c  3ffe8000  3ffe8000 000000e0  4
  1 .rodata       Содержани ALLOC, ГРУЗ,        00000620  2
                  е,        ДАННЫЕ                      **
                  00000878  3ffe8540  3ffe8540           4
  2 .bss          Содержани ALLOC, ГРУЗ, ТОЛЬКО ДЛЯ     2
                  е,        ЧТЕНИЯ, ДАННЫЕ              **
                  00009130  3ffe8db8  3ffe8db8 00000e98  4
  3 .text         ALLOC     40100000  40100000 00000e98  2
                                                        **
                  00006f22                              2
  4 .irom0.text   Содержани ALLOC, ГРУЗ, ТОЛЬКО ДЛЯ     2
                  е,        ЧТЕНИЯ, КОДЕКС              **
                  00028058  40240000  40240000 00007dc0  4
                  Содержани ALLOC, ГРУЗ, ТОЛЬКО ДЛЯ
                  е,        ЧТЕНИЯ, КОДЕКС
------------------------------------------------------------------------
---
Информация о разделе:
   Section|                  Описание | Начало (ведьма) | Конец (ведьма)
   |Used пространство
------------------------------------------------------------------------
---
                Инициализированные данные                3FFE853C
    данные |                      (RAM) |   3FFE8000 |   |          1340
                                                        3FFE8DB8
    rodata |       Данные ReadOnly (RAM) |   3FFE8540 |   |          2168
                  Неинициализированные                  3FFF1EE8
       bss |              данные (RAM) |   3FFE8DB8 |   |         37168
                Припрятавший про запас                  40106F22
     текст |              кодекс (IRAM) |   40100000 |   |         28450
  irom0_text    Не припрятавший про запас                40268058
          |              кодекс (SPI) |   40240000 |   |        163928
Полная используемая RAM: 40676
Свободная RAM:
41244
Свободный IRam:
4336
------------------------------------------------------------------------
---
objcopy, которым управляют, пожалуйста, ждите...
сделанный objcopy
Управляйт
е
gen_appbi
n.exe
Никакой
необходим
ый
ботинок.
Произведите eagle.flash.bin и орла irom0text.bin successully во
встроенном микропрограммном обеспечении папки.
eagle.flash.bin--------> 0x00000
орел
irom0text.bin---->
0x40000 Сделанный
17:57:19 строит законченный (занял 3s.141 мс),
```

Мы можем построить решения, используя предварительно снабженный Makefiles, но, лично, мне не нравится тайна, таким образом, вот рецепт для создания решения с нуля.

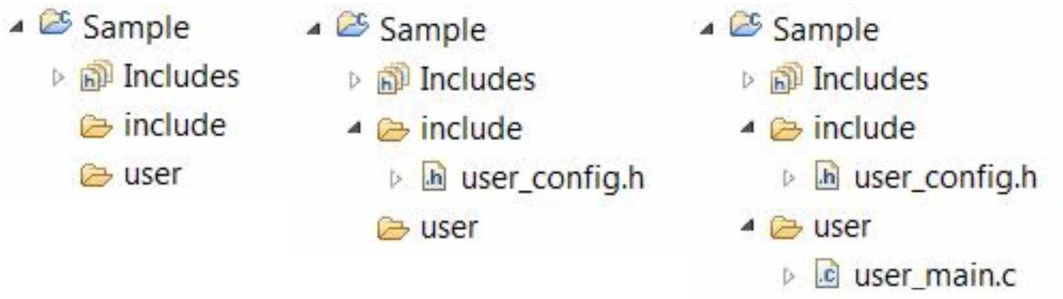
1. Создайте новый проект из Файла> Новый> С Проект
2. Выберите проект Makefile

3. Добавьте, что ESP8266 включают справочник

4. Создайте папки, названные «пользователем», и «включайте»

5. Создайте файл, названный «user_config.h» в, включают.

6. Создайте файл C, названный «user_main.c» в пользователе.



7. Создайте Makefile

```
#      Основной справочник для компилятора
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

# выберите который инструменты использовать в качестве компилятора,
библиотекаря и компоновщика
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

# использование флагов компилятора во время компиляции исходных файлов
ЦЕЛЬ          = myApp
CFLAGS        = - рот-g-o2-std=gnu90 - Wpointer-арифметика-Wundef-Werror-
Wl, - EL-fno-
действующие функции-nostdlib-mlongcalls-mtext-section-literals-mno-
serialize-volatile-D __ ets __-DICACHE_FLASH
МОДУЛИ        = пользователь
BUILD_BASE = строят
FW_BASE       = встроенное микропрограммное обеспечение
SDK_LIBDIR = lib
SDK_LDDIR     = ld

#
#      Ничто, чтобы настроить к югу от здесь.
1      #
#      флаги компоновщика раньше производили главный файл объекта
LDFLAGS       =-nostdlib-Wl, - «никакие клетчатые разделы»-u
```

```
call_user_start-Wl, - статичный
# библиотеки, пользовавшиеся в этом проекте, главным образом
предоставленном SDK
OCBOБОЖДАЕТ  = c gcc hal phy главные стр net80211 lwip wpa


# сценарий компоновщика, используемый для вышеупомянутого linkier шаг

LD_SCRIPT    = орел app.v6.ld

flashimageoptions = - flash_freq 40 м - flash_mode qio - flash_size 4 м
SDK_LIBDIR: = $ (addprefix $ (SDK_BASE)/, $ (SDK_LIBDIR))
LD_SCRIPT    : = $ (addprefix-T$ (SDK_BASE) / $ (SDK_LDDIR)/, $
(LD_SCRIPT))
OCBOБОЖДАЕТ  : = $ (addprefix-l, $ (OCBOБОЖДАЕТ)),
APP_AR       : = $ (addprefix $ (BUILD_BASE)/, $ (ЦЕЛЬ) _app.a)
TARGET_OUT: = $ (addprefix $ (BUILD_BASE)/, $ (ЦЕЛЬ) .out)
BUILD_DIRS = $ (addprefix $ (BUILD_BASE)/, $ (МОДУЛИ)) $ (FW_BASE)
SRC                = $ (foreach moduleDir, $ (МОДУЛИ), $ ($ группового
символа (moduleDir)/*.c))
# Замените весь x.c x.o
OBJS         = $ (patsubst %.c, $ (BUILD_BASE) / %.o, $ (SRC))

все: $ checkdirs
       (TARGET_OUT)
       повторяют
       «Построенный
       файл
       изображения!»
```



```
#      Постройте прикладной архив.
#      Это зависит от собранных объектов. $ (APP_AR): $ (OBJS)
       $ (AR)-cru $ (APP_AR) $ (OBJS)

#      Постройте объекты из $ исходных файлов C (BUILD_BASE) / %.o: %.c
       $ (CC) $ (CFLAGS)-I$ (SDK_BASE) / включают-Iinclude-c $ <-o $

#      Проверьте, что необходимые справочники присутствуют
checkdirs: $ (BUILD_DIRS)

# Создайте структуру каталогов, которая держится,
строение (собирает) $ (BUILD_DIRS):
       mkdir - родители - многословный $


$ (TARGET_OUT): $ (APP_AR)
       $ (LD)-L$ (SDK_LIBDIR) $ (LD_SCRIPT) $ (LDFLAGS)-Wl, - $
группы начала (OCBOБОЖДАЕТ) $ (APP_AR)-Wl, - группа конца-o $
       $ (OBJDUMP) - заголовки -
              раздел =. данные \-
              раздел =. rodata \-
              раздел =. bss \-
              раздел =. текст \-
              раздел =. $
              irom0.text
```

```
	$ (OBJCOPY) - единственный раздел .text - целевой продукцией двойной
	$ орлиный app.v6.text.bin $ (OBJCOPY) - единственный раздел .data -
	целевой продукцией двойной $ орлиный app.v6.data.bin $ (OBJCOPY) -
	единственный раздел .rodata - целевой продукцией двойной $
орел app.v6.rodata.bin
	$ (OBJCOPY) - единственный раздел .irom0.text -
целевой продукцией двойной $ орел app.v6.irom0text.bin
	$ (SDK_TOOLS) $ /gen_appbin.exe 0 0 0 0
	mv eagle.app.flash.bin $ (FW_BASE)/eagle.flash.bin
	орлиный app.v6.irom0text.bin $ mv
	(FW_BASE)/eagle.irom0text.bin орел app.v6 комнаты.
	*

#
# Высветите ESP8266
#
вспышка: все
	$ (ESPTOOL) - $ порта (ESPPORT) - $ бода (ESPBAUD) write_flash $
(flashimageoptions) 0x00000$ (FW_BASE) $/eagle.flash.bin 0x40000
(FW_BASE)/eagle.irom0text.bin
#
#	Уберите любой предыдущий строит
1	#
	чисто:
#	Удалите forceably и рекурсивно
	комната - рекурсивный - сила - многословный $ (BUILD_BASE) $
	(FW_BASE)

flashId:
	$ (ESPTOOL) - $ порта (ESPPORT) - $ бода (ESPBAUD) flash_id

readMac:
	$ (ESPTOOL) - $ порта (ESPPORT) - $ бода (ESPBAUD) read_mac
```



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

8. Добавьте Делают цели по крайней мере всех и вспышки

См. также:

- Программирование Затмения использования

Погрузка программы в ESP8266

Как только программа была собрана, она должна быть загружена в ESP8266. Эту задачу называют, «вспыхивая». Чтобы высветить ESP8266, он должен быть помещен в способ, где он примет, что новая

поступающая программа заменяет старую существующую программу. Путем это сделано, должен перезагрузить ESP8266 или удалив и повторно использовав власть или принеся остальным булавку низко и затем высоко снова. Однако просто перезагружение устройства недостаточно. Во время запуска устройство исследует стоимость сигнала, найденную на `GPIO0`. Если сигнал низкий, то это - признак, что программная сессия вспышки собирается произойти. Если сигнал на `GPIO0` будет высок, то он перейдет к своему режиму нормального функционирования. Из-за этого рекомендуется не позволить `GPIO0` плавать. Мы не хотим, чтобы это случайно перешло к вспыхивающему режиму, когда мы не желаемы. Резистор подтягивания 10k прекрасен.

Мы можем построить схему, которая включает несколько кнопок. Один для выполнения сброса и один для обеспечения `GPIO0` низко. Нажим кнопки сброса отдельно перезагрузит устройство. Это одно уже полезно. Однако, если мы держим кнопку «`GPIO0 low`», в то время как мы нажимаем сброс, тогда мы размещены в способ вспышки.

Вот является пример схематической диаграммой, иллюстрирующей ESP-12 включая кнопки:

Страница 69

Заметьте, что есть сепаратор напряжения от продукции USB к конвертеру UART булавка TX. Взгляды позади этого должны обращаться со случаем, где продукция напряжение TX больше, чем желаемые 3.3 В, требуемые на входе RX ESP8266. Это требуется? Есть уверенность, что **не** требуется, уверены ли Вы, что продукция напряжение TX составит 3.3 В. Это, кажется, имеет место для диапазона CP2102 USB к UARTs однако, я, не знают на других устройствах. То, чего я могу требовать, - то, что наличие сепаратора напряжения, который уменьшает 5 В до 3.3 В все еще, приводит к применимому напряжению уровня продукции, чтобы указать на высокий сигнал, когда оно питается 3.3-вольтовой фактической продукцией. Я не знаю, как близко я прихожу к минимальному входному напряжению RX на ESP8266, указывающем на верхний уровень.

Когда пристроено на макете, это может посмотреть следующим образом:

Это однако страдает от недостатка, что он требует, чтобы мы вручную нажали некоторые кнопки, чтобы загрузить новое применение. Это не ужасная ситуация, но возможно у нас есть альтернативы?

Когда мы высвечиваем наш ESP8266s, мы обычно соединяем их с USB-> конвертеры UART. Эти устройства в состоянии снабдить UART, используемую, чтобы программировать ESP8266. Мы знакомы с маркированным RX булавок, и TX, но что относительно булавок маркировал PTC и DTR …, что они могли бы сделать для нас?

PTC, который «Готов Послать», является продукцией от UART, чтобы сообщить нисходящему устройству, что он может теперь послать данные. Это обычно связывается с входным CTS партнера, который «Ясен Послать», который указывает, что теперь приемлемо послать данные. И PTC и CTS активны низко.

DTR, который является «Терминалом Данных, Готовым», используется в управлении потоками.

Высвечивая устройство, используя инструменты Затмения и рецепты следующее - команды вспышки, которыми управляют (как пример) и зарегистрированные сообщения:

```
22:34:17 **** Строит из Дефолта конфигурации для проекта k_blinky ****
c:/Espressif/utils/esptool.exe-p COM11-b 115200 write_flash вспышки
mingw32-make.exe-f C:/Users/User1/WorkSpace/k_blinky/Makefile - и
следующие 40 м - из qio - 0x00000 firmware/eagle.flash.bin 0x40000
```

```
firmware/eagle.irom0text.bin фс 4 м
```



```
Соединение...
Стирание вспышки...
голова: 8; общее количество: 8
сотрите размер: 16384

Письмо в 0x00000000... (3%)
Письмо в 0x00000400... (6%)
…
Письмо в 0x00007000... (96%)
Письмо в 0x00007400... (100%)
Письменные 30 720 байтов за 3,01 секунды (81,62 кбита/с)...
Стирание вспышки...
голова:
16; общее
количество
: 41
стирают
размер:
102400
Письмо в 0x00040000... (0%)
Письмо в 0x00040400... (1%)
…
Письмо в 0x00067c00... (99%)
Письмо в 0x00068000... (100%)
Письменные 164 864 байта за 16,18 секунд (81,53 кбита/с)...

Отъезд...

22:34:40 строит законченный (занял 23s.424 мс),
```

Как пример того, на что похожи сообщения, если мы не помещаем ESP8266 в способ вспышки, у нас есть следующее:

```
13:47:09 **** Строит из Дефолта конфигурации для проекта k_blinky ****
c:/Espressif/utils/esptool.exe-p COM11-b 115200 write_flash вспышки
mingw32-make.exe-f C:/Users/User1/WorkSpace/k_blinky/Makefile - и
следующие 40 м - из qio - 0x00000 firmware/eagle.flash.bin 0x40000
firmware/eagle.irom0text.bin фс 4 м
Соединение …
Traceback (новое требование в последний раз):
  Файл «esptool.py», линия 558, в
  <модуле> Файл «esptool.py»,
  линия 160, в соединяется
Исключение: Был не в состоянии соединиться
C:/Users/User1/WorkSpace/k_blinky/Makefile:313: рецепт для целевой
'вспышки' подвел mingw32-make.exe: *** [вспышка] Ошибка 255
13:47:14 строит законченный (занял 5s.329 мс),
```

Инструмент, названный `esptool.py`, обеспечивает превосходную окружающую среду для высвечивания устройства, но это может также использоваться для «чтения», что в настоящее время сохранено на него. Это может использоваться для того, чтобы сделать резервные копии из заявлений содержавшими в прежде, чем повторно высветить их с новой программой. Таким образом, Вы можете всегда возвращаться к тому, что Вы имели перед переписыванием. Например, на Unix:


```
esptool.py - порт/dev/ttyusb0 read_flash 0x00000 0xFFFF делает-
копию-0x00000.bin esptool.py - резервная-копия-0x10000.bin
порта/dev/ttyusb0 read_flash 0x10000 0x3FFFF
```

См. также:

- USB к конвертерам UART
- Рекомендуемая установка для программирования ESP8266
- Работа с памятью
- Что такое UART?
- esptool.py
- esptool-ck

Программирование окружающей среды

Мы можем программировать ESP8266, используя поставляемый SDK Espressif на Windows, используя Затмение. Поставляется отдельная глава по подготовке той окружающей среды. У нас также есть способность программировать ESP8266, используя IDE Ардуино. Это - потенциально меняющая правила игры история и это слишком данный ее собственную важную главу.

См. также:

- Программирование Затмения использования
- Программирование использования IDE Ардуино

Инструменты компиляции

Есть много инструментов, которые важны, когда строительство C основывало заявления ESP8266.

площадь

Инструмент архива привык к упакованному вместе собранные файлы объекта в библиотеки. Эти библиотеки заканчиваются «`.a`» (архив).

Библиотеку можно назвать, используя компоновщика, и объекты, содержавшие в, будут использоваться, чтобы решить внешний облик.

Некоторые наиболее распространенные флаги, используемые с этим инструментом, включают:

- `-c` – Создают библиотеку
- `-r` – Заменяют существующих участников в библиотеке
- `-u` – Обновление существующие участники в библиотеке

Синтаксис команды:

```
площадь-cru libraryName member.o member.o ….
```

См. также:

- GNU –площадь
- nm

Страница 73

esptool.py

Этот инструмент - общедоступное внедрение, используемое, чтобы высветить ESP8266 через последовательный порт. Это написано в Пайтоне. Версии, как замечалось, были доступны как исполняемые файлы окон, которые, кажется, были произведены «`.EXE`» файлы из кодекса Пайтона, подходящего для работы Windows без поддержки установка времени выполнения Пайтона.

- `-p порт | -порт` – Последовательный порт, чтобы использовать
- `-b бод | -бод` – скорость передачи в бодах, чтобы использовать для сериала
- `-h` – Помощь
- `{Команда}-h` – Помощь для той команды
- `load_ram {имя файла}` – Загрузка изображение к RAM и выполняет
- `dump_mem {адрес} {размер} {имя файла}` – Свалка произвольная память диску
- `read_mem {адрес}` – Прочитанное произвольное местоположение памяти
- `write_mem {адрес} {стоимость} {маска}` – Рид-модифи-райт к произвольной памяти

  местоположение
- `write_flash` – Пишут двойную каплю, чтобы вспыхнуть

- -flash_freq {40 м, 26 м, 20 м, 80 м} | - и следующие {40 м, 26 м, 20 м, 80 м} – частота Вспышки SPI
- -flash_mode {qio, qout, dio, dout} | - из {qio, qout, dio, dout} – способ Вспышки SPI

- -flash_size {4 м, 2 м, 8 м, 16 м, 32 м, 16m c1,32m c1,32m c2} | - фс {4 м, 2 м, 8 м, 16 м, 32 м, 16m c1,32m c1,32m c2} – размер Вспышки SPI в Мегабите
- {Адрес} {имя файла} – Адрес, чтобы написать, подайте, чтобы написать … повторяемый

- управляемый – программный код Пробега во вспышке
- image_info {файл изображения} – заголовки Свалки от прикладного изображения. Вот пример произвел:

```
Точка
входа:
40100004 3
сегмента
Сегмент 1: 25 356 байтов по телефону 40100000
Сегмент 2: 1 344 байта в 3ffe8000
Сегмент 3: 924 байта в 3ffe8540

Контрольная сумма: 40 (действительный)
```

- make_image – Создают прикладное изображение из бинарных файлов
  - - segfile SEGFILE,-f SEGFILE – Сегмент ввел файл

- -segaddr SEGADDR,-a SEGADDR – базовый адрес Сегмента
- -entrypoint ENTRYPOINT,-e ENTRYPOINT – Адрес точки входа
- продукция

- elf2image – Создают прикладное изображение из файла ЭЛЬФА
- -ПРОДУКЦИЯ продукции,-о ПРОДУКЦИЯ – префикс имени файла Продукции
- -flash_freq {40 м, 26 м, 20 м, 80 м}, - и следующие {40 м, 26 м, 20 м, 80 м} – вспышка SPI частота

- -flash_mode {qio, qout, dio, dout}, - из {qio, qout, dio, dout} – способ Вспышки SPI

- -flash_size {4 м, 2 м, 8 м, 16 м, 32 м, 16m c1,32m c1,32m c2}, -фс {4 м, 2 м, 8 м, 16 м, 32 м, 16m c1,32m c1,32m c2} – размер Вспышки SPI в Мегабите

- -символ входа ENTRY_SYMBOL,-es ENTRY_SYMBOL – символ Точки входа

  имя (дефолт 'call_user_start')

- read_mac – Прочитанный MAC-адрес от ROM OTP. Вот продукция в качестве примера:

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

- flash_id – Прочитанные SPI быстро показывают ID устройства и производитель. Вот пример

  продукция:

```
голова:
0; общее
количест
во: 0
стирают
размер:
0
Производ
ителей:
Устройст
во c8:
4014
```

- read_flash – Прочитанные SPI высвечивают содержание

- адрес – адрес Начала

- размер – Размер региона, чтобы свалить

- имя файла – Название разгрузки двоичных данных

- erase_flash – Выступают, Чип Стирают на вспышке SPI. Это - особенно полезное

  командуйте, заканчиваете ли Вы кого-то облицовывающего устройство кирпичом, поскольку оно должно перезагрузить устройство к своим дефолтам.

См. также:

- esptool-ck
- nodemcu-маяк-мигалка
- Погрузка программы в ESP8266
- Работа с памятью
- GitHub:themadinventor/esptool

esptool-ck

Другой инструмент, который также называют `esptool-ck`. Обозначение этих инструментов, являющихся настолько подобным, начинает становиться неудобным.

- `-eo <имя файла>` – Открывают объект ЭЛЬФА.
- `-es <раздел> <имя файла>` – Прочитанный названный раздел из объекта и пишет к названному файлу.
- `-EC` – Завершения файл ЭЛЬФА.
- `-филиал <имя файла>` – Готовит микропрограммный файл к ESP.
- `-BM <qio|qout|dio|dout>` – Набор чип вспышки соединяют способ.
- `-bz <512K|256K|1M|2M|4M|8M|16M|32M>` – Набор размер кристалла вспышки.
- `-bf <40|26|20|80>` – Набор частота чипа вспышки.
- `-бакалавр наук <раздел>` – Прочитанный раздел ELF и пишут микропрограммному изображению.
- `-до н.э` – Близко микропрограммное изображение.
- `-v` – Увеличение многословие продукции (`-v`,`-vv`,`-vvv`)
- `-q` – Отключают большую часть продукции
- `-CP <устройство>` – Последовательное устройство (например, `COM1`)
- `-CD <правление>` – Выбирают метод сброса для сброса правления.
  - ничего
  - ck
  - wifio
  - nodemcu
- `-cb <baudrate>` – Выбирают скорость передачи в бодах, чтобы использовать.
- `-приблизительно <адрес>` – Адрес флэш-памяти как цель закачки.
- `-cf <имя файла>` – Закачка названный файл, чтобы вспыхнуть.

Здесь, например, команда, чтобы высветить правление NodeMCU devKit:

```
esptool - CP COM15 - CD nodemcu-cb 115200 - приблизительно 0x00000-cf
myApp_0x00000.bin
```

Вот чистая регистрация закачки IDE Ардуино:

```
esptool v0.4.5 - (c) 2 014 Ch. Klippel правление
       урегулирования <ck@atelier-klippel.de> к ck
       урегулирование baudrate от

       115 200 до 115 200 портов

       урегулирования от COM1 до COM11
```



```
       урегулирование адреса от 0x00000000
       до 0x00000000 espcomm_upload_file
       статистика C:
       \Release/Test_ESP_RESTClient.bin
       успех, устанавливающий перерывы
       последовательного порта в 1 000 мс
открытие
правления
сброса
загрузчик
а
операцион
ной
системы,
пытающего
ся
соединить
ся
       начало потока
       урегулирование перерывов
       последовательного порта к 1
       мс, устанавливающей перерывы
       последовательного порта в
       1 000 мс, вспыхивает полный
       espcomm_send_command:      отправка
       заголовка                   команды
       espcomm_send_command:      отправка
       полезного       груза      команды,
       прочитанного 0, просила 1
попытка
       с
       о
       е
       д
       и
       н
       и
       т
       ь
       н
       а
       ч
       а
       л
       о
       п
```

```
        o
        т
        о
        к
        а
        урегулирование перерывов
        последовательного порта к 1
        мс, устанавливающей перерывы
        последовательного порта в
        1 000 мс, вспыхивает полный
        espcomm_send_command: отправка
        заголовка команды
        espcomm_send_command: отправка
        полезного груза команды
        espcomm_send_command: получение 2
        байтов данных espcomm_send_command:
        получение 2 байтов данных
        espcomm_send_command: получение 2
        байтов данных espcomm_send_command:
        получение 2 байтов данных
        espcomm_send_command: получение 2
        байтов данных espcomm_send_command:
        получение 2 байтов данных
        espcomm_send_command: получение 2
        байтов данных espcomm_send_command:
        получение 2 байтов данных
        espcomm_open
Загрузка 312 944 байтов от C:
\Release/Test_ESP_RESTClient.bin, чтобы
вспыхнуть в 0x00000000
        стирание вспышки
        размер: адрес
        04c670: 000000
        first_sector_index
        : 0
        total_sector_count
        : 77
        head_sector_count:
        16
        adjusted_sector_co
        unt: 61
        adjusted_size:
        03d000
        espcomm_send_command: отправка
        заголовка команды
        espcomm_send_command: отправка
        полезного груза команды,
        устанавливающего перерывы
        последовательного порта в 10 000 мс,
        устанавливающих перерывы
        последовательного порта в 1 000 мс
        espcomm_send_command: получение 2
        байтов вспышки написания данных
.....................................................................
...........
.....................................................................
...........
.....................................................................
...........
.....................................…
стартовое приложение без перезагрузки
        espcomm_send_command: отправка
        заголовка команды
        espcomm_send_command: отправка
        полезного груза команды
```

```
        espcomm_send_command: получение 2
        байтов данных
закрытие
        за
        гр
        уз
        чи
        ка
        оп
        ер
        ац
        ио
        нн
        ой
        си
        ст
        ем
        ы
        см
        ыв
        ае
        т
        на
        ча
        ло
```

Страница 77

```
        урегулирование перерывов
        последовательного порта к 1
        мс, устанавливающей перерывы
        последовательного порта в
        1 000 мс, вспыхивает полный
```

Наборы из двух предметов, соответствующие выпускам инструмента, могут быть найдены согласно разделу выпусков:

https://github.com/igrr/esptool-ck/releases Видят также:

- esptool.py
- nodemcu-маяк-мигалка
- GitHub:https://github.com/igrr/esptool-ck

gcc

Общедоступная Коллекция Компиляторов GNU включает компиляторы для C и C ++. Если мы смотрим тщательно на флаги, которые поставляются для компилирования и соединения кодекса для ESP8266, мы находим следующее:

Компилирование

- `-c` – Собирают кодекс к файлу объекта `.o`.
- `-pot` – Оптимизируют генерацию объектного кода для размера.
- `-O2` – Оптимизируют для работы, которые кодируют результат в большем кодовом размере. Для пример, вместо того, чтобы делать вызов функции, кодекс мог быть в-выровненном.
- `-ggdb` – Генерируют код отладки, который может использоваться `gdb` отладчиком..
- `-std=gnu90` – Диалект C поддержан.
- `-Werror` – Совершают все ошибки предупреждений.
- `-Wno-адрес` – не предупреждают о подозрительном использовании адресов памяти.
- `-Wpointer-арифметика` – Предупреждает, когда арифметика указателя предпринята, который зависит от `sizeof`.
- `-Wundef` – Предупреждают, когда идентификатор найден в #if директиве, которая не является макросом.
- `-fno-inline-functions` – не позволяют функциям быть замененными машинными командами.
- `-mlongcalls` – Переводят прямые требования ассемблера на косвенные требования.
- `-mtext-section-literals` – Позволяют опечаткам быть смешанными с текстовым разделом.
- `-mno-serialize-volatile` – Специальные инструкции для изменчивых определений.

Соединение:

- `-nostdlib` – не используют стандарт C или C ++ системные библиотеки запуска

См. также:

- GCC – коллекция компиляторов GNU

gen_appbin.py

Синтаксис этого инструмента:

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

- flash_mode
  - `0` – QIO
  - `1` – QOUT
  - `2` – DIO
  - `3` – DOUT
- flash_clk_div
  - `0` – 80 м / 2
  - `1` – 80 м / 3
  - `2` – 80 м / 4
  - `0xf` – 80 м / 1
- flash_size_map
  - 0 - 512 КБ (256 КБ + 256 КБ)
  - 1 - 256 КБ
  - `2` – 1 024 КБ (512 КБ + 512 КБ)
  - `3` – 2 048 КБ (512 КБ + 512 КБ)
  - `4` – 4 096 КБ (512 КБ + 512 КБ)
  - `5` – 2 048 КБ (1 024 КБ + 1 024 КБ)
  - `6` – 4 096 КБ (1 024 КБ + 1 024 КБ)

Следующие файлы, как ожидают, будут существовать:

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

Продукция этой команды - новый файл, названный `eagle.app.flash.bin`.

сделать

Сделайте двигатель компиляции, используемый, чтобы отследить то, что должно быть собрано, чтобы создать Ваше целевое приложение. Сделайте ведется Makefile. Хотя сильный и достаточно простой для простых проектов C, это может стать сложным довольно быстро. Если Вы изучаете Makefiles, написанный другими, захватываете превосходную GNU, делают документацию и изучают его глубоко.

nodemcu-маяк-мигалка

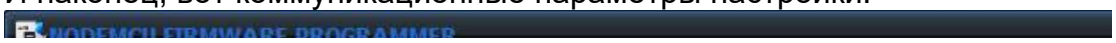
Этот инструмент - другой случай маяка мигалки ESP8266. В отличие от некоторых из других доступных инструментов, этот - базирующийся графический интерфейс пользователя. Из инструмента можно выбрать все варианты, которые можно было бы ожидать включая один или несколько файлов высвечивать, последовательная связь и информация и т.д.

Один введенный, можно нажать кнопку «Flash», и высвечивание начинается с привлекательного индикатора выполнения.

Следующее - то, на что инструмент похож после завершения вспышки:

Вот то, на что это похоже в его счете выбора файла вспышки:

И наконец, вот коммуникационные параметры настройки:



Хотя визуально привлекательный, у этого, кажется, есть большой недостаток. Это чувствует себя намного медленнее, чтобы вспыхнуть, чем некоторые из других инструментов. Это, конечно, предполагает, что каждый пытается вспыхнуть при той же скорости передачи в бодах.

Однако даже с этой небольшой слабостью, это - все еще один из самых легких, чтобы использовать доступные инструменты маяка мигалки и, кажется, идеально подходит для случайного маяка мигалки. Если бы я

должен был рекомендовать инструменту использоваться кем-то, кто только должен был установить приложение на их ESP8266 нечасто, это, вероятно, было бы им.

См. также:

- esptool.py
- esptool-ck
- GitHub:nodemcu/nodemcu-flasher
- YouTube:ESP8266, как быстро показать встроенное микропрограммное обеспечение NodeMcu
- Высвечивание встроенного микропрограммного обеспечения NodeMCU на ESP8266 (Windows) - Руководство

nm

Символы списка

из файлов

объекта.

Полезные флаги:

- `-только определенный` – Шоу только определило экспорт
- `-неопределено-единственный` – Шоу только неопределенный экспорт
- -номера линии

См. также:

- площадь
- GNU –nm

objcopy

См. также:

- GNU –objcopy

objdump

Команда - `эльф xtensa lx106 objdump` расположенный в `C:\Espressif\xtensa-lx106-elf\bin`. Некоторые более

важные флаги:

- `-syms` – Свалка символы в архиве.

См. также:

- Википедия –objdump
- GNU –objdump
- страница человека –objdump (1)

xxd

Это - обманчиво простой, но полезный инструмент. То, что это делает, свалить двоичные данные, содержавшиеся в рамках файла в отформатированной форме. Одно сильное использование его должно взять бинарный файл и произвести структуру данных языка C, которая представляет содержание файла. Это означает, что Вы можете взять двоичные данные и включать его в Ваши заявления. Копия `xxd.exe` распределена с SDK, поставляемым Espressif в папке `инструментов`. Следующее прочитает содержание `inFile` как двоичные данные и произведет заголовочный файл в `outFile`.

```
xxd - включайте <inFile> <outFile>
```

Страница 82

Соединение ESP8266

Когда C и C ++ исходные файлы, которые составляют Ваш проект, были собраны к их файлам объекта, пора связать их с библиотеками, чтобы завершить исполняемый файл, который будет развернут. Вот пример команды соединения, используемой, чтобы построить исполняемый файл.

```
эльф xtensa
   lx106
   gcc-g
   -
   P
   o
   t
   -
   n
   o
   s
   t
   d
   l
   i
   b
   - Wl, - «никакие клетчатые
   разделы»-u call_user_start-Wl, -
   статичный
      «-
      L??/tools/sdk//
      lib» «-L??
      tools/sdk//ld»
      «-
```

```
      teagle.flash.51
      2k.ld»
   - Wl, - обертка,
   system_restart_local-
   Wl, - обертка,
   register_chipv6_phy
   - o
   «Release/Test_ESP_RESTClient
   .elf»-Wl, - группа начала
   x.o
   y.o
   z
   .
   o
   -
   l
   m
   -
   l
   g
   c
   c
   -
   l
   h
   a
   l
   -
   l
   p
   h
   y
   -
   lne
   t80
   211
   -
   llw
   ip-
   lwp
   a-
   lma
   in-
   lpp

   -
   lsmar
   tconf
   ig-
   lwps-
   lcryp
   to
- Wl, -
группа
конца «-
LRelease
»
```

Заметьте, что некоторыми библиотеками пользуются, связываясь. Многие из этих библиотек снабжены Espressif SDK.

Название библиотеки	Описание
в	
crypto	
espnow	
hal	
json	
lwip	
lwip_536	
главный	
net80211	
phy	
стр	
pwm	
smartconfig	
ssc	
ssl	
модернизация	
wpa	
wps	

См. также:

- Работа с памятью
- Компоновщик GNU
- Введение в GNU – компилятор и компоновщик

Высвечивание по воздуху - ФОТА

Предположите, что Вы создали фантастическое приложение ESP8266, которое содержится во встроенном устройстве. Вы отправляете его своим клиентам, и все великое. Внезапно, Вы начинаете получать отчеты, что это периодически терпит неудачу. Вы диагностируете ошибку и к Вашему ужасу, Вы находите, что там кодировал ошибку …, но к счастью, это легко и быстро зафиксировано. Вы теперь находите, что у Вас есть проблема. Вашему встроенному устройству не выставили UART и, даже если бы оно сделало, то Ваши клиенты бунтовали бы, если бы они должны были включить его в компьютер. Хуже, это стало бы кошмаром поддержки, чтобы идти потребители через механику достижения перезагружения, когда мы не сделали предположений о технических навыках конечного потребителя.

Чтобы решить этот вопрос, мы вводим понятие высвечивания (или более конкретно ре - вспыхивающий) применение по Интернету через связь WiFi. Это понятие называют, «Вспыхивая По Воздуху» или ФОТА.

В общем это работает следующим образом.

Ваш ESP8266 имеет сумму в наличии флэш-памяти к нему.
Давайте разделим ту память на две равных половины.

Когда суда ESP8266, Ваше заявление будет загружено в $1^{Св.}$ половину вспышки и проигнорирует вторую половину. Время от времени это будет «звонить домой» через Интернет и спрашивать, есть ли набор замены встроенного микропрограммного обеспечения (новая версия). Если будет, то это загрузит то новое встроенное микропрограммное обеспечение в $2^{\text{без обозначения даты}}$ половины вспышки. Если это полностью будет иметь успех, то устройство перезагрузит и начнет управлять новым встроенным микропрограммным обеспечением от $2^{\text{без обозначения даты}}$ половин вспышки …, это теперь проигнорирует $1^{Св.}$ половину. Последующая замена встроенного микропрограммного обеспечения с еще одной версией будет загружена в $1^{Св.}$ половину и повторения истории.

Эффективно, мы таким образом в состоянии щелкнуть - шлепаются между двумя версиями. С этой теорией высокого уровня под нашими поясами давайте теперь выроем немного глубже. Очевидно, если мы следуем этой истории, мы видим, что имеем, эффективно уменьшают сумму вспышки, доступной, чтобы вести наши программы наполовину. Это не кажется хорошим …, почему мы делаем это? Ответ на самом деле довольно прост. Если мы рассматриваем действительность, что связи WiFi могут прерваться, доступ в Интернет может быть потерян, и власть может просто быть удалена из устройства, мы можем закончить в ситуации, где закачка нового кодекса во вспышку на самом деле терпит неудачу, прежде чем это закончит. Это оставило бы нас со взломанным кодом в области вспышки. Если бы мы попробовали к «оперативному», заменяют наше существующее заявление, и такая неудача произошла, мы "облицевали бы устройство кирпичом". Вероятно, что замененное применение, которое было только наполовину загружено, даже не загрузит. Чтобы обойти ту проблему, ESP8266 содержит флаг, который определяет, какая из двух возможных половин встроенного микропрограммного обеспечения является в настоящее время той, используемой, чтобы выполнить программу. Только то, когда новая версия встроенного микропрограммного обеспечения была утверждена как успешно загруженный, является флагом, переключенным на другую половину. Если ошибка происходит во время закачки замены, то флаг не переключен, и никакой вред не будет причинен, так как мы эффективно проигнорировали вторую половину вспышки во-первых.

Страница 85

Давайте теперь пойдем еще глубже. Espressif предоставляет код, названный «ботинком», который ответственен за загрузку ESP8266. Когда ESP8266 приведен в действие на, именно этот кодекс ботинка получает контроль. Это - ботинок, который определяет, как остаток от включения питания устройства продолжится. Когда мы высвечиваем ESP8266, мы должны обеспечить и применение ботинка и нашу

собственную прикладную логику. С точки зрения адресного пространства программа ботинка загружена в `0x00000` адреса вспышки для 4 кбайт. Наше заявление будет загружено от адреса `0x01000` вперед. Так как у ESP8266 может быть множество размеров вспышки, мы исследуем каждый из них в свою очередь. Вспышка 512 КБ

Содержание	**Адрес**	**Размер**
Ботинок	`0x0 0000 – 0x0 0FFF`	4 КБ
Приложение 1 (user1.bin)	`0x0 1000 – 0x3 BFFF`	236 КБ
Пользователь params	`0x3 C000 – 0x3 FFFF`	16 КБ
Не использовать	`0x4 0000 – 0x4 0FFF`	4 КБ
Приложение 2 (user2.bin)	`0x4 1000 – 0x7 BFFF`	236 КБ
Система params	`0x7 C000 – 0x7 FFFF`	16 КБ

Вспышка 1024 КБ

Содержание	**Адрес**	**Размер**
Ботинок	`0x0 0000 – 0x0 0FFF`	4 КБ
Приложение 1 (user1.bin)	`0x0 1000 – 0x7 BFFF`	492 КБ
Пользователь params	`0x7 C000 – 0x7 FFFF`	16 КБ
Не использовать	`0x8 0000 – 0x8 0FFF`	4 КБ
Приложение 2 (user2.bin)	`0x8 1000 – 0xF BFFF`	492 КБ
Система params	`0xF C000 – 0xF FFFF`	16 КБ

Вспышка 2048 КБ – выбор 1

Содержание	**Адрес**	**Размер**

Ботинок	0x00 0000 – 0x00 0FFF	4 КБ
Приложение 1 (user1.bin)	0x00 1000 – 0x07 BFFF	492 КБ
Пользователь params	0x07 C000 – 0x07 FFFF	16 КБ
Не использовать	0x08 0000 – 0x08 0FFF	4 КБ
Приложение 2 (user2.bin)	0x08 1000 – 0x0F BFFF	492 КБ
Пользовательские данные	0x0F C000 – 0x1F BFFF	1008 КБ
Система params	0x1F C000 – 0x1F FFFF	16 КБ

Вспышка 2048 КБ – выбор 2

Содержание	**Адрес**	**Размер**
Ботинок	0x00 0000 – 0x00 0FFF	4 КБ
Приложение 1 (user1.bin)	0x00 1000 – 0x07 BFFF	1004 КБ
Пользователь params	0x0F C000 – 0x0F FFFF	16 КБ
Не использовать	0x10 0000 – 0x10 0FFF	4 КБ
Приложение 2 (user2.bin)	0x10 1000 – 0x1F BFFF	1004 КБ
Система params	0x1F C000 – 0x1F FFFF	16 КБ

Вспышка 4096 КБ – выбор 1

Содержание	**Адрес**	**Размер**
Ботинок	0x00 0000 – 0x00 0FFF	4 КБ
Приложение 1 (user1.bin)	0x00 1000 – 0x07 BFFF	492 КБ
Пользователь params	0x07 C000 – 0x07 FFFF	16 КБ
Не использовать	0x08 0000 – 0x08 0FFF	4 КБ
Приложение 2 (user2.bin)	0x08 1000 – 0x0F BFFF	492 КБ
Пользовательские данные	0x0F C000 – 0x3F BFFF	3072 КБ
Система params	0x3F C000 – 0x3F FFFF	16 КБ

Вспышка 4096 КБ – выбор 2

Содержание	Адрес	Размер
Ботинок	0x00 0000 – 0x00 0FFF	4 КБ
Приложение 1 (user1.bin)	0x00 1000 – 0x07 BFFF	1004 КБ
Пользователь params	0x0F C000 – 0x0F FFFF	16 КБ
Не использовать	0x10 0000 – 0x10 0FFF	4 КБ
Приложение 2 (user2.bin)	0x10 1000 – 0x1F BFFF	1004 КБ
Пользовательские данные	0x1f C000 – 0x3F BFFF	2048 КБ
Система params	0x3F C000 – 0x3F FFFF	16 КБ

См. также:

- Документ Espressif: 99C – ESP8266 – Модернизация ОТЫ

Отладка

Сочиняя программы, мы можем найти, что они не всегда бегут как ожидалось. Выполнение отлаживающий на SOC может быть трудным, так как у нас нет легко доступных исходных отладчиков уровня.

Регистрация ESP-IDF

Структура ESP-IDF обеспечивает регистрирующийся набор особенностей. Эти пункты регистрации могут тогда быть вставлены в Ваше собственное заявление на диагностирование проблем или завоевание следов.

Чтобы использовать регистрирующиеся функции, мы должны включать «esp_log.h».

Функция регистрации высокого уровня вызвана «esp_log_write ()», у которого есть следующая подпись:

```
пустота esp_log_write (esp_log_level_t уровень, случайная работа константы *признак, случайная работа константы * формат, …)
```

Думайте о нем как специализированный printf лесоруб. Формат и после параметров следует конвенции стиля printf.

По умолчанию, когда регистрацию требуют, продукцию посылают в основной последовательный поток. Однако мы можем отвергнуть то место назначения при помощи функции, вызванной esp_log_set_vprintf

`()`. Это берет в качестве параметра ссылку на функцию C, у которой есть тот же синтаксис как `vprintf`. Конкретно:

```
интервал myPrintFunction (случайная работа константы *формат, va_list
аргумент)
```

Когда мы хотим зарегистрировать сообщение, мы выбираем уровень регистрации, чтобы написать. Доступные уровни регистрации:

- `ESP_LOG_NONE`

- `ESP_LOG_ERROR`
- `ESP_LOG_WARN`
- `ESP_LOG_INFO`
- `ESP_LOG_DEBUG`
- `ESP_LOG_VERBOSE`

Зарегистрированная продукция имеет формат:

```
<регистрируют уровень> (<отметка времени>) <признак>: <сообщение>
```

Где уровень регистрации - один из «`E`», «`W`», «`I`», «`D`» или «`V`».

Отметка времени - количество миллисекунд начиная с ботинка.

У нас также есть глобальное урегулирование, которое является максимальным уровнем регистрации, который мы должны зарегистрировать. Например, если мы установим `ESP_LOG_WARN` тогда сообщения на уровне `ESP_LOG_WARN`, `то ESP_LOG_ERROR` будет зарегистрирован, но `ESP_LOG_INFO`, `ESP_LOG_DEBUG` и `ESP_LOG_VERBOSE` будут исключены.

Параметр признака к регистрирующейся функции обеспечивает признак, которого логический компонент/модуль выпускает сообщение. Это предоставляет контекст тому, что иначе могло бы быть неоднозначными сообщениями.

Макрос языка C обеспечен, чтобы сделать использование регистрации более простым. Макрос:

- `ESP_LOGE (признак, формат, …)` - Регистрация ошибка.
- `ESP_LOGW (признак, формат, …)` - Регистрация предупреждение.
- `ESP_LOGI (признак, формат, …)` - информация о Регистрации.
- `ESP_LOGD (признак, формат, …)` - отладка Регистрации.

- `ESP_LOGV (признак, формат, …)` - Регистрация многословная информация.

Так как регистрация включена или исключена во время компиляции, мы можем определить, что регистрирующийся уровень, чтобы включать в наш строит. Во время компиляции это может исключить определенные заявления регистрации из источника. Флаг компиляции-`DLOG_LOCAL_LEVEL` управляет регистрирующимися включенными уровнями.

Для заявлений регистрации, которые остаются в кодексе после компиляции, которые не были исключены во время изготовления, мы можем управлять уровнем регистрации во времени выполнения, звоня `esp_log_level_set ()`. Подпись этой функции:

```
пустота esp_log_level_set (случайная работа константы *признак,
esp_log_level_t уровень)
```

Имена тега регистрирующиеся группы, которые мы покажем. Если специальный признак имени «*» поставляется, это соответствует всем признакам.

Если мы пишем режимы обработки перерыва, не используйте эти функции регистрации в тех.

- Ошибка: Справочный источник, не найденный

Регистрация к UART1

Мы можем вставить диагностические заявления, используя `os_printf ()`. Это заставляет текст и данные, связанные с этими функциями быть написанным UART1 (GPIO 2). Если мы прилагаем USB → UART устройство в схеме, мы можем тогда посмотреть на зарегистрированные данные. В моей среде разработки у меня всегда есть два USB → UART устройства в игре. Один, чтобы высветить новые заявления и один, чтобы использовать для диагностической продукции.

OS также в состоянии написать отладочную информацию. По умолчанию это идет, но может быть выключено с требованием к `system_set_os_print ()`.

См. также:

- USB к конвертерам UART
- Работа с сериалом
- system_set_os_print

Управляйте Blinky

Физически смотря на ESP8266 нет очень, чтобы видеть, что это говорит, что Вы все работаете хорошо в нем. Есть свет власти и сетевая передача активный свет …, но это об этом. Техника, которую я рекомендую, состоит в том, чтобы всегда иметь Ваше устройство, выступают, выполняют «мигание, ведомое», который обычно известен как «Blinky». Это может быть достигнуто, соединив булавку GPIO с резистором ограничения тока и затем со светодиодом. Когда сигнал GPIO идет высоко, светодиодные индикаторы. Когда сигнал GPIO идет низко, светодиод становится темным. Если мы определяем отзыв таймера, который называют (например), однажды секунда и переключается, сигнал булавки GPIO оценивают каждую просьбу, у нас будет простой дьявольский светодиод. Вы будете удивлены, как хороший чувство это даст просто знание, что *что-то* живо в устройстве каждый раз, когда Вы видите, что он мигает.

Стоимость управления таймером и изменения стоимости ввода/вывода, чтобы достигнуть мигания не должна быть проблемой в течение времени разработки, таким образом, я не волновался бы о побочных эффектах выполнения этого. Очевидно, для изданного применения, Вы не можете желать этого и можете просто удалить его.

Однако, хотя это - тривиальная схема, у нее есть большое использование во время развития. Во-первых, Вы будете всегда знать, что устройство работает. Если светодиод мигает, Вы знаете, что устройство имеет силу и контроль за обработкой логики. Если свет прекратит мигать, то Вы будете знать, что что-то заперлось, или Вы вошли в бесконечную петлю.

Другая полезная цель для включения Blinky состоит в том, чтобы утвердить это, Вы перешли к режиму вспышки, программируя устройство. Если мы понимаем, что устройство может загрузить в

нормальный или способ вспышки и мы загружаем его в способе вспышки, тогда Blinky прекратит выполнять. Это может быть полезно, если Вы используете кнопки или прыгунов, чтобы переключить способ ботинка, поскольку он представит свидетельства, что Вы *не*

находитесь в нормальном способе. При случае я неправильный нажал некоторые кнопки контроля, и быстро смог понять, что что-то неправильно прежде даже пыталось высветить его, поскольку Blinky все еще шел.

Вот некоторый простой кодекс для создания Blinky. В этом примере мы используем GPIO4 в качестве драйвера для светодиода. Во-первых, кодекс мы помещаем в `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, (пустота *) 0); os_timer_arm (&blink_timer,
1000, 1);
```

Это принимает глобальный названный `blink_timer`, определенный как:

```
МЕСТНЫЙ os_timer_t blink_timer;
```

Функцию обратного вызова в этом примере называют `blink_cb` и похожа:

```
МЕСТНАЯ пустота ICACHE_FLASH_ATTR blink_cb (пустота *аргумент)
{
       led_state =!
       led_state;
       GPIO_OUTPUT_SET (4,
       led_state);
}
```

Глобальная переменная, названная `led_state`, содержит текущее состояние светодиода (1=on, 0=off):

```
МЕСТНОЕ ОТДЕЛЕНИЕ UINT8_T LED_STATE=0;
```

Демпинг IP-адресов

Будучи WiFi и устройством TCP/IP, Вы предположили бы, что ESP8266 работает много с IP-адресами, и Вы были бы правы. Мы можем произвести представление последовательности использования IP-адреса:

```
os_printf (IPSTR, IP2STR (pIpAddrVar))
```

макрос IPSTR «`%d. % d. % d. % d`» так вышеупомянутое эквивалентен:

```
os_printf (» %d. % d. % d. % d», IP2STR (pIpAddrVar))
```

который может быть более полезным в определенных ситуациях.

См. также:

- ipaddr_t

Обработка исключений

Во времени выполнения вещи могут не всегда работать как ожидалось, и исключение может быть брошено. Например, Вы могли бы попытаться

получить доступ к хранению в недействительном местоположении или написать, чтобы прочитать только память или выполнить различие нолем.

Когда такое возникновение произойдет, устройство перезагрузит себя, но не прежде, чем написать некоторую диагностику UART1.

Диагностика может быть похожей:

```
Фатальное исключение (28):
epc1=0x40243182, epc2=0x00000000, epc3=0x00000000, excvaddr=
0x00000050, depc=0x00000000
```

Кодексы следующие:

- `exccause` – Кодекс, описывающий причину
- `epc1` – прилавок программы Исключения
- `excvaddr` – Виртуальный адрес, который вызвал новое усилие, груз или магазин
  исключение. Например, если писание памяти происходит, и та память не RAM, исключение будет брошено, и стоимость здесь будет адресом, который был предпринят, чтобы быть написанным.

Основные коды исключений:

Кодекс	Кодекс	Имя причины
0	0x00	IllegalInstructionCause
1	0x01	SyscallCause
2	0x02	InstructionFetchErrorCause
3	0x03	LoadStoreErrorCause
4	0x04	Level1InterruptCause
5	0x05	AllocaCause
6	0x06	IntegerDivideByZeroCause
7	0x07	Зарезервировано
8	0x08	PrivilegedCause
9	0x09	LoadStoreAlignmentCause
10	0x0a	Зарезервировано
11	0x0b	Зарезервировано
12	0x0c	InstrPIFDataErrorCause
13	0x0d	LoadStorePIFDataErrorCause
14	0x0e	InstrPIFAddrErrorCause

15	0x0f	LoadStorePIFAddrErrorCause
16	0x10	InstTLBMissCause
17	0x11	InstTLBMultiHitCause
18	0x12	InstFetchPrivilegeCause
19	0x13	Зарезервировано
20	0x14	InstFetchProhibitedCause

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

Если мы знаем местоположение исключения, мы можем проанализировать исполняемый файл `(app.out)`, чтобы выяснить, какая часть кодекса вызвала проблему. Например:

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

Используя отладчик (GDB)

GDB - Отладчик GNU и является превосходным инструментом для отладки собранного исходного кода C. Однако это, прежде всего, разработано, чтобы отладить OS, принял заявления, такие как собранные для Windows или Linux и не имел большой применимости для ESP8266. Это было, пока Espressif не выпустил их окурок GDB.

Версия отладчика должна быть `эльфом xtensa lx106 gdb` инструмент.

Чтобы подготовиться к использованию инструмента, нужно собрать со следующими дополнительными флагами:

- -ggdb
- -Og

Кроме того, мы должны инициализировать GDB с требованием к `gdbstub_init ()` где-нибудь рано в кодексе запуска в нашем заявлении. Наконец, мы связываемся в `gdbstub` библиотеке.

См. также:

- GitHub:espressif/esp-gdbstub

Отладка и тестирование TCP и связей UDP

Работая с TCP/IP, Вы, вероятно, захотите иметь некоторые приложения, которые Вы можете использовать, чтобы послать и получить данные так, чтобы Вы могли быть уверены, что ESP8266 работает. Там

много превосходных инструментов и доступных утилит, и они варьируются платформой и функцией.

Android - Протокол гнезда

Протокол Гнезда - бесплатное приложение для Android, доступное от App Store игры Google. См.:

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

Android - Отправитель/Приемник UDP

Отправитель/Приемник UDP - другое бесплатное приложение для Android, доступное от App Store игры Google. Что делает, этот интересный является его способностью быть UDP (в противоположность TCP) отправитель и получатель. См.:

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

Windows - Геркулес

Геркулес - более старое приложение для Windows, который все еще, кажется, работает просто великолепно на последних выпусках. Это выглядит немного старым в зубе теперь, но все еще, кажется, делает работу очень хорошо. См.:

- http://www.hw-group.com/products/hercules/index_en.html

Завиток

Завиток - мощный и всесторонний инструмент командной строки для

выполнения любого и всех связанных с URL команд. Это может передать запросы HTTP всех различных форматов и получить их ответы. Это имеет в наличии озадачивающий набор параметров к нему, который является оба благословением и проклятием. Вы можете быть вполне уверены, что, если это может быть сделано, Завиток может сделать это … однако быть готовым пробраться через большое количество документации.

См. также:

- Завиток

Затмение – TCP/MON

Один из самых мощных и полезных доступных инструментов называют Монитором TCP/IP, который снабжен как часть Затмения и распределен с «Инструментами веб-разработчика Затмения». Монитор TCP/IP открыт через представление Затмения, названное «Монитор TCP/IP».

Страница 94

Если Вы не можете найти его в видоискателе, возможности высоки, что Вы не установили «Инструменты веб-разработчика Затмения».

После того, как начатый, открытый его имущественное стекло:

Оттуда Вы можете добавить местных слушателей. Они будут слушателями TCP/IP, которые слушают на местном порте, куда Затмение бежит. Во время конфигурации Вы определяете другой IP-адрес и номер порта. Когда трафик TCP теперь прибывает в слушателя на который TCP/IP

Страница 95

Наставник смотрит, это передаст тот трафик партнеру, одновременно регистрируя его к экрану TCP/IP Monitor.

Например, здесь Наставник TCP/IP слушает на 192.168.1.2 (localhost), который является, куда Затмение бежит. Это слушает на порте 9999. Когда трафик TCP/IP прибудет в тот адрес, его пошлют вперед в 192.168.1.17 (который, оказывается, мое устройство ESP8266) держать в строевой стойке 80.

Вот пример регистрации, которую я видел, отправляя запрос браузера:

Как Вы видите, информацией, захваченной здесь, является сильный материал. Мы видим каждый транспортный запрос, его содержание и заголовки HTTP.

httpbin.org

Проверяя протоколы HTTP, соединяясь с веб-сайтом пohttp:// httpbin.org может быть неоценимым. Это обеспечивает массу услуг для тестирования запросов HTTP.

Архитектура ESP8266

Чтобы начать думать о написании заявлений на ESP8266, мы должны понять архитектуру высокого уровня устройства.

Таможенные программы

Таможенные программы - заявления, которые Вы можете написать и являетесь основным центром этой книги. Эти программы могут быть написаны в C или C ++ и затем собраны в бинарные файлы. Программы, как ожидают, определят «известные» функции в этом, служат спроектированными точками входа и отзывами.

Программисты пишут файл языка C с предложенным названием «`user_main.c`». Содержавший в функция с подписью:

```
пустота user_init (пустота)
```

Это обеспечивает начальный вход в программный код. Это называют однажды во время запуска. Выполняя в этой функции, поймите, что не вся окружающая среда все же готова к эксплуатации. Если Вы нуждаетесь в полностью функционирующей окружающей среде, регистрируете функцию обратного вызова, которая будет призвана, когда окружающая среда будет на 100% готова. Эта функция обратного вызова может быть зарегистрирована в требовании к `system_init_done_cb`.

Через инициализацию РФ нужно также обеспечить:

```
пустота user_rf_pre_init (пустота)
```

Бегая в пользовательском кодексе, мы должны быть чувствительными, что основная цель устройства - сетевые коммуникации. Так как они обработаны в программном обеспечении, когда пользовательский кодекс получает контроль, который просто означает, что организация сети кодекса не делает. Так как у нас только есть одна нить контроля, мы не можем быть в двух местах сразу. Рекомендуемая продолжительность, чтобы потратить в пользовательском кодексе в единственном заседании является меньше, чем 10msecs.

См. также:

- system_init_done_cb

WiFi при запуске

ESP8266 хранит информацию запуска WiFi во флэш-памяти. Это позволяет ему выполнять свои функции при запуске, не имея необходимость просить у пользователя любую специальную или дополнительную информацию. По-моему, это - больше проблемы, чем это стоит. Если я собираюсь написать применение ESP8266, я хочу управлять, когда, как и к тому, что оно соединит

или быть точкой доступа. К счастью есть функция, `вызванная wifi_station_set_auto_connect ()`, и его партнер назвал `wifi_station_get_auto_connect ()`. Они позволяют нам отвергать авто функции связи, когда мы - станция.

Работа с WiFi - ESP8266

ESP8266 может или быть станцией в сети, точке доступа для других устройств или обоих. Это - фундаментальное соображение, и мы захотим выбрать, как устройство ведет себя вначале в нашем дизайне. Как только мы выбрали то, что мы хотим, мы устанавливаем глобальную собственность способа, которая указывает, какой из эксплуатационных способов наше устройство выполнит (станция, точка доступа или станция И точка доступа).

Просмотр для точек доступа

Если ESP8266 выполнит роль станции, то мы должны будем соединиться с точкой доступа. Мы можем просить список доступных точек доступа, против которых мы можем попытаться соединиться. Мы делаем это использование `wifi_station_scan ()` функция. Эта функция берет указатель функции обратного вызова в качестве одного из его параметров. Этот отзыв будет призван, когда просмотр закончит. Отзыв необходим, потому что он может занять время (несколько секунд) для просмотра, который будет выполняться, и мы не можем позволить себе заблокировать операцию системы в целом до полный. Функция обратного вызова просмотра получает связанный список структур BSS. Содержавший в структуре BSS:

- SSID для сети
- BSSID для точки доступа
- Канал
- Сила сигнала
- … другие

Например:

```
пустота scanCB (пустота
   *аргумент, статус
   СТАТУСА) {структура
   bss_info *bssInfo;
   bssInfo = (структура bss_info *) аргумент;
   //пропустите первое в цепи …, это -
   недействительный bssInfo =
```

```
   STAILQ_NEXT (bssInfo, затем); в то
   время как (bssInfo! = ПУСТОЙ
   УКАЗАТЕЛЬ) {
      os_printf («ssid: %s\n»,
      bssInfo-> ssid); bssInfo =
      STAILQ_NEXT (bssInfo, затем);
   }
}

//...
```



```
{
   //Гарантируйте, что мы
   находимся в станционном
   способе
   wifi_set_opmode_current
   (STATION_MODE);
   //Просите просмотр сети, звоня «scanCB» на
   завершении wifi_station_scan (ПУСТОЙ УКАЗАТЕЛЬ,
   scanCB);
}
```

Отметьте использование `STAILQ_NEXT ()` макрос, чтобы провести к следующему входу в списке. Конец списка обозначен, когда это возвращает `ПУСТОЙ УКАЗАТЕЛЬ`.

См. также:

- Образец – сканер WiFi
- структура bss_info
- СТАТУС

Определение рабочего режима

ESP8266 может выполнить как Станция WiFi, точка доступа WiFi или и станция и точка доступа. Их считают тремя возможными глобальными рабочими режимами. Рабочий режим, который используется, когда устройство загружает, сохранен во флэш-памяти, но может быть изменен с требованием к `wifi_set_opmode ()`. Это изменит текущий способ, а также сделает запись способа, который будет использоваться на следующем перезапуске. Чтобы просто изменить способ, не изменяя следующий способ ботинка, мы можем использовать `wifi_set_opmode_current ()`. Чтобы восстановить текущий способ, мы можем использовать `wifi_get_opmode ()` и восстановить способ, используемый на ботинке, мы можем использовать `wifi_get_opmode_default ()`. Вполне, почему у нас есть выбор измениться, текущий способ, не экономя его во флэш-памяти является

тайной. По-видимому, есть некоторый случай, когда такая особенность была необходима и таким образом выставлена, но что когда-либо то, что причина может быть, не очевидно.

См. также:

- wifi_get_opmode
- wifi_get_opmode_default

Обработка событий WiFi

В ходе работы как устройство WiFi определенные события могут иметь место, о котором должен знать ESP8266. Они могут быть важными или заинтересовать к заявлениям, бегущим в нем. Так как мы не знаем, когда, или даже если, какие-либо события произойдут, у нас не сможет быть нашего прикладного блока, ждущего, чтобы они произошли. Вместо этого то, что мы должны сделать, определяют функцию обратного вызова, которая будет призвана, должен событие на самом деле происходить. Функция, `вызванная wifi_set_event_handler_cb ()`, делает просто это. Это регистрирует функцию, которая будет вызвана, когда ESP8266 обнаружит определенные типы связанных с WiFi событий. Зарегистрированная функция призвана и передана богатая структура данных, которая включает тип события и связанных данных, соответствующих тому событию. Типы событий, которые заставляют отзыв происходить:

- Мы соединились с точкой доступа
- Мы разъединили от точки доступа
- Способ разрешения изменился
- Мы добрались, DHCP выпустил IP-адрес
- Станция соединилась с нами, когда мы находимся в способе Точки доступа
- Станция разъединила от нас, когда мы находимся в способе Точки доступа

Вот пример функции обработчика событий, которая просто регистрирует название события, которое было замечено:

```
пустота eventHandler (System_Event_t *событие)
    {выключатель (событие-> событие) {случай
    EVENT_STAMODE_CONNECTED:
      os_printf («Событие:
      EVENT_STAMODE_CONNECTED\n»); разрыв;
    случай EVENT_STAMODE_DISCONNECTED:
```

```
      os_printf («Событие:
      EVENT_STAMODE_DISCONNECTED\n»); разрыв;
    случай EVENT_STAMODE_AUTHMODE_CHANGE:
      os_printf («Событие:
      EVENT_STAMODE_AUTHMODE_CHANGE\n»); разрыв;
    случай EVENT_STAMODE_GOT_IP:
      os_printf («Событие:
      EVENT_STAMODE_GOT_IP\n»); разрыв;
    случай EVENT_SOFTAPMODE_STACONNECTED:
      os_printf («Событие:
      EVENT_SOFTAPMODE_STACONNECTED\n»); разрыв;
    случай EVENT_SOFTAPMODE_STADISCONNECTED:
      os_printf («Событие:
      EVENT_SOFTAPMODE_STADISCONNECTED\n»); разрыв;
    дефолт:
      os_printf («Неожиданное событие: %d\n»,
    событие-> событие); разрыв;
  }
}
```

Функция обратного вызова может быть зарегистрирована в `user_init ()` следующим образом:

```
wifi_set_event_handler_cb (eventHandler);
```

Мы ограничены тем, что мы должны сделать в отзыве обработчика событий. А именно, кажется, что мы не должны пытаться сформировать новые связи. Вместо этого мы должны отправить задачу, что мы теперь в состоянии сделать дополнительную работу.

См. также:

- System_Event_t

`system_init_done_cb ()`

Станционная конфигурация

Когда мы будем думать о ESP8266 как о Станции WiFi, мы поймем, что в любой момент, это может только быть связано с одной точкой доступа. Помещая его иначе, нет никакого значения в высказывании, что устройство связано с **двумя или больше** точками доступа одновременно.

Идентичность точки доступа, с которой мы хотим быть связанными, известна как `«station_config»` и смоделирована как структура C, названная «`структура station_config`». Содержавший в той структуре две очень важных области, названные `«ssid»` и `«паролем»`. `ssid`

область - SSID точки доступа, с которой мы соединимся. Область `пароля` - ценность открытого текста пароля, который будет использоваться, чтобы подтвердить подлинность нашего устройства к целевой точке доступа, чтобы позволить связь.

Когда он загружен, ESP8266 помнит последний `station_config`, который мы устанавливаем. Мы можем явно установить `station_config` данные, используя функцию `wifi_station_set_config ()`. Это установит текущую конфигурацию **и** спасет ее для более позднего поиска после перезагрузки. Если мы только хотим установить текущую станционную конфигурацию и **не** иметь сохраненную информацию, мы может использовать `wifi_station_set_config_current ()`.

Мы не должны пытаться выполнить любые операции WiFi, пока устройство полностью не инициализировано.
Мы знаем, что инициализированы, регистрируя

отзыв, используя функцию.

Например:

```
пустота initDone ()
  {wifi_set_opmode_current
  (STATION_MODE); структура
  station_config stationConfig;
  strncpy (stationConfig.ssid,
  «myssid», 32);
  strncpy (stationConfig.password,
  «mypassword», 64);
  wifi_station_set_config
  (&stationConfig);
}
```

См. также:

- system_init_done_cb
- wifi_station_get_config_default
- wifi_station_set_config_current
- station_config

Соединение с точкой доступа

Как только ESP8266 был настроен со станционными деталями конфигурации, который включает SSID и пароль, мы готовы выполнить связь с целевой точкой доступа. Функция `wifi_station_connect ()` сформирует связь. Поймите, что это не мгновенно, и Вы не должны предполагать, что сразу после этой команды связаны. Ничто в блоках ESP8266 и как таковой ни один не делает требование к этой функции. Некоторое время спустя мы будем на самом деле связаны. Мы будем видеть два запущенные события отзыва. Первым является `EVENT_STAMODE_CONNECTED`, указывающий, что мы имеем

связанный с точкой доступа. Второе событие - `EVENT_STAMODE_GOT_IP`, который указывает, что нам назначил IP-адрес сервер DHCP. Только в том пункте может мы действительно участвовать в коммуникациях. Если мы будем использовать статические IP-адреса для нашего устройства, то мы будем только видеть связанное событие.

Есть одно дальнейшее соображение, связанное с соединением с точками доступа, и это - идея автоматической связи. Есть булев флаг, который сохранен во вспышке, которая указывает, должен ли ESP8266 попытаться автоматически соединиться с последней используемой точкой доступа. Если установлено в истинный, то после того, как устройство запущено и без Вас имеющий необходимость закодировать любые требования API, оно попытается соединиться с последней используемой точкой доступа. Это - удобство, которое я предпочитаю выключать. Обычно, я хочу, чтобы контроль в моем устройстве определил, когда я соединяюсь. Мы можем позволить или повредить автомобиль, соединяют особенность, звоня `wifi_station_set_auto_connect ()`.

См. также:

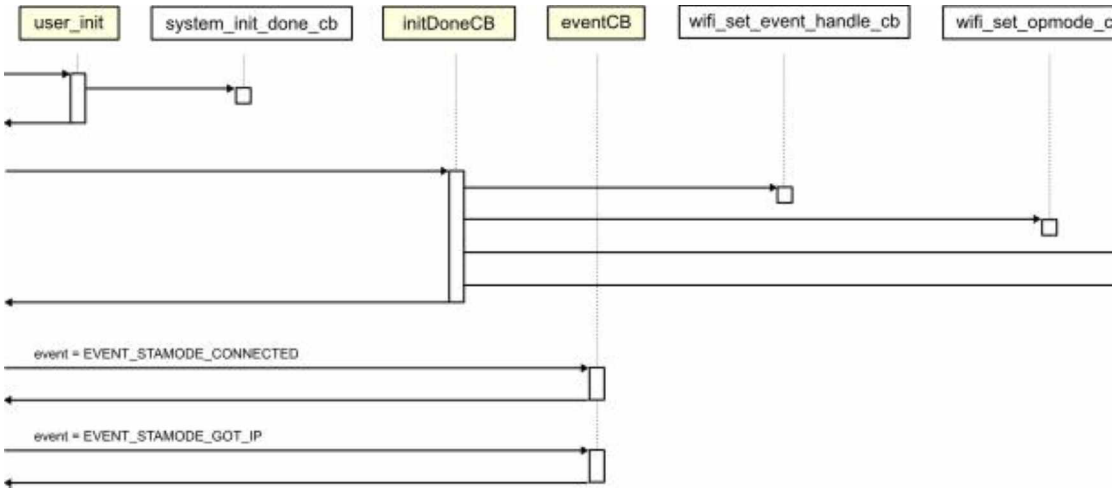
- Обработка событий WiFi

Контроль и потоки данных, соединяясь как станция

Мы теперь на стадии, где мы можем потянуть поток последовательности частей. Некоторые функции Вы ответственны и должны поставлять включая:

- `user_init` – Точка входа в применение
- `initDoneCB` – Отзыв, когда инициализация была закончена
- `eventCB` – Отзыв, когда связанное с WiFi событие обнаружено

Другие функции мы ответственны за запрос. Мы будем считать эту часть последовательности законченной, когда у нас будет признак, что у нас есть действительный IP-адрес.



Быть точкой доступа

До сих пор мы только рассмотрели ESP8266 как станцию WiFi к существующей точке доступа, но у него также есть способность **быть** точкой доступа к другим устройствам WiFi (станции) включая другой ESP8266s.

Чтобы быть точкой доступа, мы должны определить SSID это, которое позволяет другим устройствам отличать нашу сеть. Этот SSID может сигнализироваться, как скрытый, если мы не хотим, чтобы это было просмотрено. Кроме того, мы должны будем также поставлять способ идентификации, который будет использоваться, когда станция будет хотеть, соединяется с нами. Это используется, чтобы позволить разрешенные станции и отвергнуть не - разрешенные. Только станциям, которые знают наш пароль, позволят соединиться. Если мы используем

идентификацию, то мы должны будем также выбрать пароль, который соединяющиеся станции должны будут знать и поставлять, чтобы успешно соединиться.

Первая задача в том, чтобы быть точкой доступа состоит в том, чтобы сигнализировать использование как таковое ESP8266 `wifi_set_opmode ()` или `wifi_set_opmode_current ()` функции и пройти во флаге, который просит, чтобы мы были или посвятить точкой доступа или точкой доступа **и** станцией.

Вот отрывок кода, который может использоваться к установке и ESP8266 как точка доступа:

```
//     Определите наш способ как Точку доступа wifi_set_opmode_current
(SOFTAP_MODE);

//     Постройте наши детали конфигурации Точки доступа os_strcpy
(config.ssid, «ESP8266»); os_strcpy (config.password, «пароль»);
конфигурация ssid_len = 0;
       config.authmode = AUTH_OPEN; конфигурация ssid_hidden = 0; конфигурация
max_connection = 4; wifi_softap_set_config_current (&config);
```

Когда отдаленная станция соединится с ESP8266 как точка доступа, мы будем видеть сообщение отладки, написанное UART1, который может выглядеть подобным:

```
станция: соединение f0:25:b7:ff:12:c5, ПОМОЩЬ = 1
```

Это содержит MAC-адрес новой станции, присоединяющейся к сети. Когда станция разъединит, мы будем видеть, что соответствующая отладка регистрирует сообщение, которое может быть:

```
станция: отпуск f0:25:b7:ff:12:c5, ПОМОЩЬ = 1
```

Из ESP8266 мы можем определить, сколько станций в настоящее время связано с требованием к `wifi_softap_get_station_num ()`. Если мы хотим найти детали тех станций, мы можем назвать `wifi_softap_get_station_info ()`, который возвратит связанный список `структуры station_info`. Мы должны явно выпустить хранение, ассигнованное этим требованием с просьбой `wifi_softap_free_station_info ()`.

Вот пример отрывка кодекса, который перечисляет детали связанных станций:


```
uint8 stationCount =
wifi_softap_get_station_num ();
os_printf («stationCount = %d\n»,
stationCount);
структура station_info *stationInfo =
```

```
wifi_softap_get_station_info (); если (stationInfo! =
ПУСТОЙ УКАЗАТЕЛЬ) {
   в то время как (stationInfo! = ПУСТОЙ УКАЗАТЕЛЬ) {
      os_printf («Станционный IP: %d. % d. % d. % d\n», IP2STR
      (& (stationInfo-> IP))); stationInfo = STAILQ_NEXT
      (stationInfo, затем);
   }
   wifi_softap_free_station_info ();
}
```

Когда ESP8266 действует как точка доступа, это позволяет другим устройствам соединяться с ним и формировать связь WiFi. Однако кажется, что два устройства, связанные с тем же ESP8266, действующим как точка доступа, не могут непосредственно общаться друг между другом. Например, вообразите два устройства, соединяющиеся с ESP8266 как точка доступа. Они могут быть ассигнованы IP-адреса 192.168.4.2 и 192.168.4.3. Мы могли бы предположить, что 192.168.4.2 мог свистеть 192.168.4.3 и стих визы, но это не позволено. Кажется, что они только прямое разрешенное сетевое подключение между недавно связанными станциями и точкой доступа (ESP8266) сам.

Это, кажется, ограничивает применимость ESP8266 как точка доступа. Основное намерение ESP8266 как точка доступа состоит в том, чтобы позволить мобильным устройствам (например, Ваш телефон) соединяться с ESP8266 и разговаривать с применением, которое бежит на него.

См. также:

- wifi_softap_set_config_current
- wifi_softap_get_station_num

Сервер DHCP

Когда ESP8266 выполняет роль точки доступа, вероятно, что Вы захотите, чтобы это также вело себя как сервер DHCP так, чтобы соединение станций было в состоянии быть автоматически назначенными IP-адресами и изучить их маски подсети и ворота.

Сервер DHCP может быть запущен и остановлен в устройстве, используя API, названные `wifi_softap_dhcps_start ()` и `wifi_softap_dhcps_stop ()`. Текущее состояние (начался или остановился), сервера DHCP может быть найдено с требованием к `wifi_softap_dhcps_status ()`.

Диапазон дефолтов IP-адресов, предлагаемых сервером DHCP, 192.168.4.1 вверх. Первый адрес становится назначенным на сам ESP8266. Важно понять, что этот диапазон адресов не тот же диапазон адресов как Ваша LAN, где Вы можете работать. ESP8266 сформировал свое собственное сетевое адресное пространство и даже при том, что

они могут появиться с теми же видами чисел (192.168.x.x), они изолированы и независимые сети. Если Вы начинаете точку доступа на ESP8266 и соединяетесь с ним

с Вашего телефона не удивляйтесь, когда Вы попытаетесь свистеть он от своего связанного с Интернетом PC и не получите ответ.

См. также:

- wifi_softap_dhcps_stop

Текущий IP-адрес, netmask и ворота

Если нам нужен он, мы можем подвергнуть сомнению окружающую среду OS для текущего IP-адреса, netmask и ворот. Значения их обычно устанавливаются для нас сервером DHCP, когда мы соединяемся с точкой доступа. Функция, `вызванная wifi_get_ip_info ()`, возвращает нашу текущую информацию, в то время как функция, `вызванная wifi_set_ip_info ()`, позволяет нам устанавливать наши адреса.

Когда мы соединяемся с точкой доступа и приняли решение использовать DHCP, когда мы ассигнованы IP-адрес, событие произведено, который может использоваться в качестве признака, что у нас теперь есть действительный IP-адрес.

К правильно установке статические IP-адреса, в `init_done отзыве`, называют `wifi_station_dhcpc_stop ()`, чтобы искалечить клиента DHCP, бегущего в ESP8266. После этого требования `wifi_station_connect ()`, чтобы начать фазу связи точки доступа. Когда событие прибывает, который указывает, что мы связаны с точкой доступа как станция `(EVENT_STAMODE_CONNECTED)`, мы можем назвать `wifi_set_ip_info ()` и пройти в IP-адресе, воротах и netmask, который мы хотим использовать. Обратите внимание, что, когда мы используем статический IP-адрес, мы не получим событие отзыва, которое указывает, что мы получили IP-адрес `(EVENT_STAMODE_GOT_IP)`, поскольку у нас уже есть он.

См. также:

- Обработка событий WiFi
- wifi_station_dhcpc_stop
- структура ip_info

WiFi защищенная установка - WPS

ESP8266 поддерживает WiFi Защищенная Установка в станционном способе. Это означает, что, если точка доступа поддерживает его, ESP8266 может соединиться с точкой доступа, не представляя пароль. В настоящее время только «способ кнопки» связи осуществлен. Используя этот механизм, физическая кнопка нажата на точке доступа и, сроком на две минуты, любая станция в диапазоне может присоединиться к сети, используя протоколы WPS. Примером использования была бы нажимаемая кнопка WPS точки доступа и затем устройство ESP8266, звоня `wifi_wps_enable ()` и затем `wifi_wps_start ()`. ESP8266 тогда соединился бы с сетью.
См. также:

- wifi_wps_enable

- wifi_wps_start
- wifi_set_wps_cb
- Простые Вопросы: Что такое WPS (WiFi Защищенная Установка)
- Википедия:WiFi защищенная установка

Работа с TCP/IP

TCP/IP - сетевой протокол, который используется в Интернете. Это - протокол, который ESP8266 с рождения понимает и использует с WiFi в качестве транспорта. Книги по книгам были уже написаны о TCP/IP, и наша цель не состоит в том, чтобы попытаться воспроизвести детальное обсуждение того, как это работает, однако, есть некоторые понятия, которые мы попытаемся захватить.

Во-первых, есть IP-адрес. Это - 32-битная стоимость и должно быть уникально для каждого устройства, связанного с Интернетом. 32-битная стоимость может считаться четырьмя отличными 8-битными ценностями (4 x 8=32). Так как мы можем представлять 8-битное число как десятичное значение между 0 и 255, мы обычно представляем IP-адреса с примечанием <число>. <число>. <число>. <число>, например, 173.194.64.102. Эти IP-адреса обычно не вводятся в заявления. Вместо этого текстовое имя введено, такие как «`google.com`» …, но не будьте введены в заблуждение, эти имена - иллюзия на уровне TCP/IP. Вся работа выполняется с 32-битными IP-адресами. Есть система отображения, которая берет имя (такое как «`google.com`») и восстанавливает его соответствующий IP-адрес. Технологию, которая делает это, называют «Системой доменных имен» или DNS.

Когда мы думаем о TCP/IP, здесь есть на самом деле три отличных протокола в действии. Первым является IP (интернет-Протокол). Это -

основная дейтаграмма транспортного уровня мимолетный протокол. Выше IP слоя TCP (протокол TCP), который обеспечивает иллюзию связи по IP протоколу без установления соединения. Наконец есть UDP (Пользовательский Дейтаграммный Протокол). Это также живет выше IP протокола и предоставляет дейтаграмме передачу (без установления соединения) между заявлениями. Когда мы говорим, что TCP/IP, только говорим о переезжающем TCP IP, но на самом деле используем это в качестве сокращения от основных протоколов, которые являются IP, TCP и UDP и дополнительными связанными протоколами уровня приложения, такими как DNS, HTTP, FTP, Telnet и т.д.

espconn архитектура

Поскольку нам не разрешают заблокировать контроль в ESP8266 в течение никакого отрезка времени, мы должны зарегистрировать функции обратного вызова, которые будут призваны, когда некоторое долгое действие продолжительности закончило или, асинхронные события имеют место. Например, когда мы хотим получить поступающее сетевое подключение, мы не можем просто ждать той связи, чтобы прибыть. Вместо этого мы регистрируем функцию обратного вызова связи и затем возвращаем контроль назад к OS. Когда связь в конечном счете прибывает в будущее, функция обратного вызова, которую мы ранее зарегистрировали, призвана от нашего имени.

Страница 106

В следующей таблице перечислены функции обратного вызова, что ESP8266 обеспечивает соединения по протоколу TCP поддержки и события.

Функция регистра	Отзыв	Описание
espconn_regist_connectcb	espconn_connect_callback	TCP соединился успешно
espconn_regist_disconcb	espconn_disconnect_callback	TCP, разъединенный успешно
espconn_regist_reconcb	espconn_reconnect_callback	Обнаруженная ошибка или TCP разъединена
espconn_regist_sentcb	espconn_sent_callback	Посланный TCP или данные UDP
espconn_regist_recvcb	espconn_recv_callback	Полученный TCP или данные UDP
espconn_regist_write_finish	espconn_write_finish_callback	Напишите данные в TCP-send-buffer

См. также:

- espconn_regist_connectcb
- espconn_regist_disconcb
- espconn_regist_reconcb
- espconn_regist_sentcb
- espconn_regist_recvcb
- espconn_regist_write_finish

TCP

Соединение по протоколу TCP - bi - направленная труба, через которую данные могут течь в обоих направлениях. Прежде чем связь установлена, одна сторона действует как сервер. Это пассивно прислушивается к поступающим запросам связи. Это будет просто сидеть там столько, сколько необходимый, пока запрос связи не прибывает. Другая сторона связи ответственна за инициирование связи, и это активно просит связь быть сформированным. Как только связь была построена, обе стороны могут послать и получить данные. Для «клиента», чтобы просить связь, это должно знать информацию об адресах, на которой слушает сервер. Этот адрес состоит из двух отличных частей. Первая часть - IP-адрес сервера, и вторая часть - «номер порта» для определенного слушателя. Если мы думаем о PC, у Вас может быть много заявлений на нем, каждое из которых может получить поступающую связь. Просто знание IP-адреса Вашего PC не достаточно, чтобы обратиться к связи с правильным применением. Комбинация IP-адреса плюс номер порта обеспечивает все необходимое обращение.

Как аналогия с этим, думайте о своем сотовом телефоне. Это пассивно сидит там, пока кто-то не называет его. В нашей истории это - слушатель. Адрес, который кто-то использует, чтобы сформировать связь, является Вашим номером телефона, который состоит из кода области плюс остаток. Например, номер телефона (817) 555-1234 достигнет конкретного телефона. Однако код области 817 является для Форт-Уэрта в Техасе … запросом, который отдельно не достаточен, чтобы достигнуть отдельного …, полный номер телефона требуется.

Нет мы посмотрим на то, как ESP8266 может собраться как слушатель для поступающей связи TCP/IP.

Страница 107

Мы начинаем, вводя абсолютно жизненную структуру данных, которую называют «структурой espconn». Эта структура данных содержит большую часть «состояния» нашей связи и передана в большинство наших API TCP.

Мы инициализируем его, устанавливая много его областей:

• напечатайте – Это - тип связи, которую мы собираемся использовать. Так как мы хотим использовать
соединение по протоколу TCP в противоположность связи UDP,
мы поставляем ESPCONN_TCP как стоимость.

• государство – состояние связи будет изменяться со временем, но мы инициализируем его к
имейте начальное пустое государство,

поставляя ESPCONN_NONE. Например:

```
структура espconn conn1;

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

Теперь мы вводим другую структуру, названную «esp_tcp». Эта структура содержит определенные параметры настройки TCP. Для нашей истории это - то, где мы поставляем номер порта, на который послушает наше соединение по протоколу TCP для связей клиента. Это поставляется в названной собственности
«local_port».

```
esp_tcp tcp1;

пустота init ()
  {tcp1.local_po
  rt = 25867;
}
```

В структуре espconn тип данных, есть область, названная «первичной», который является указателем на протокол определенная структура данных. Для соединения по протоколу TCP это будет указателем на «esp_tcp» случай …, и это - то, где мы добираемся, чтобы склеить историю. Полный кодекс становится:

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

Мы можем теперь запустить наш сервер, прислушивающийся к поступающим соединениям по протоколу TCP, используя `espconn_accept ()`. Это берет `структуру espconn`, как введено, который используется, чтобы указать, на каком порте мы должны послушать (среди прочего). Вот пример:


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

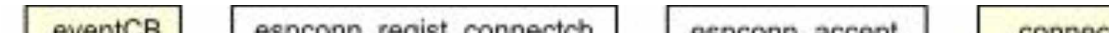
После запроса этого ESP8266 будет теперь пассивно прислушиваться к поступающим соединениям по протоколу TCP на порте, определенном в `local_port` области. Важно отметить, что Ваш кодекс не блокирует ожидание поступающего запроса. Где-нибудь в самом центре ESP8266 это теперь знает, чтобы принять связи на том порте. Следующий вопрос - простой …, что происходит, когда связь в конечном счете прибывает?

Ответ на это - часть основной архитектуры устройства и вращается вокруг понятия отзывов. В Вашем собственном программном коде именно будет призвана Ваша обязанность зарегистрировать функцию обратного вызова, когда связь прибудет. Это - то, где `espconn_regist_connectb ()` функция играет роль. Эта функция регистрируется, пользователь поставлял функцию обратного вызова, которую назовут, когда связь прибудет.

```
пустота connectCB (пустота *аргумент) {
   структура espconn *pNewEspConn =
   (структура espconn *) аргумент; … Делают
   что-то с новой связью
}

{
   ...
   espconn_regist_connectcb (&conn1,
   connectCB); espconn_accept
   (&conn1);
}
```

Рассматриваемый как блок-схема последовательности, мы видим отношения между некоторыми компонентами. Мы предполагаем, что в конечном счете отзыв, когда мы были ассигнованы IP-адрес, мы тогда, регистрирует это, мы интересуемся связями и что мы готовы принять поступающие новые связи. Затем в некоторое время в будущем мы получаем новый запрос связи, и отзыв связи призван.



Содержание `структуры espconn` прошло в отзыв, будет включать отдаленный IP-адрес партнера, который соединился с нами. Мы можем использовать ту информацию для регистрации или для разрешения. Например, если IP-адрес не тот, мы хотим позволить, мы можем разъединить в этом пункте, используя `espconn_disconnect ()`. Поймите, что эта структура данных представляет **новую** связь с партнером, который просто призвал и **не** является тем же как `структурой espconn`, который использовался, чтобы зарегистрировать это, мы хотели принять новые связи. Новая `структура espconn` будет передана в для каждой новой сформированной связи.

`espconn_regist_time ()`
Это покрывает ESP8266, получающий поступающие запросы связи, но что, если он должен желать сформировать связь, за границу к отдаленному TCP-приложению? Чтобы выполнить связь за границу просят, чтобы мы могли использовать `espconn_connect ()` требование. До совершения этого звонка мы должны настроить структуру TCP. Область `remote_port` должна содержать номер порта прикладного партнера, с которым мы хотим соединиться. Кроме того, `remote_ip` область должна содержать IP-адрес машины, принимающей партнера. `local_port` нужно назначить неиспользованный местный порт, используя `espconn_port (). local_ip` должен также быть закончен, используя местный IP-адрес. Точно так же, как получение прибывающей связи, устанавливая связь за границу приведет к просьбе к отзыву связи, когда связь будет установлена. Как только связь была сформирована, еще раз, два конца связи будут ровесниками друг друга. **Жизненно важно** понять, что просто издание `espconn_connect ()` **не** приводит к непосредственной связи. Вместо этого только после того, как `connectCB` был получен, может мы на самом деле использовать связь.

Например:

```
структура
espconn
conn1;
esp_tcp
```

```
tcp1;

пустота init ()
  {tcp1.remote_port
  = 25867;
  tcp1.remote_ip =
  ipAddress;
  tcp1.local_port = espconn_port
  (); структура 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;
}
```

Если партнер в нашем разговоре должен закрыть связь, нам сообщат об этом через функцию, которую мы регистрируем в `espconn_regist_disconcb ()`. Государственная область `структуры espconn` будет содержать `БЛИЗКО`. Обнаружение изящное закрытие партнера позволяет нам выполнять логику, в которой мы, возможно, нуждаемся, такие как высвобождение средств или сохраняющихся данных.

Если соединение по протоколу TCP сформировано и никакие транспортные потоки по связи в течение по крайней мере 10 секунд (дефолт), то связь автоматически закрыта от конца ESP8266.

Неработающая собственность перерыва

связи может быть установлена с функцией.

ESP8266 поддерживают максимум 5 параллельных

соединений по протоколу TCP.См. также:

- espconn_accept

- espconn_connect
- espconn_disconnect
- espconn_regist_connectcb
- espconn_regist_disconcb
- espconn_regist_time
- структура espconn
- esp_tcp

Отправка и получение данных TCP

На данном этапе давайте теперь предположим, что у нас есть связь между ESP8266 и заявлением партнера. Наличие связи большое, но

теперь мы должны разговаривать. Информация и данные должны течь в одном или обоих направлениях. Есть два соображения …, мы можем получить данные от партнера, или мы можем хотеть послать данные партнеру. Важно отметить, что в TCP, связь двунаправлена. Как только связь была установлена, любая сторона может послать данные в любое время. Нет никакого понятия одной стороны, имеющей исключительную отправку или получение прав. Выбор того, кто получатель и кто передатчик, просто до дизайна применения.

Например, предположите, что у нас был проект включить светодиод в ESP8266, когда он получает «1» характер, и выключите его, когда он получает «0» характер. В той истории ESP8266 был бы исключительно приемником и, просто нашим выбором, не должен передавать данные. Партнер был бы исключительно передатчиком.

Теперь давайте рассмотрим второй пример. В этом случае ESP8266 связан с датчиком температуры и каждые несколько секунд, он посылает текущую температуру партнеру. В той истории ESP8266 - исключительно передатчик и партнер только приемник.

Наконец, мы можем изображение ESP8266, связанный с несколькими датчиками. Это получает команды от партнера, как введено, которого это интерпретирует. На основе полученных данных правильный датчик выбран, его прочитанная стоимость и результаты, переданные назад. В этой истории ESP8266 - сначала приемник и затем становится передатчиком, в то время как партнер - противоположное.

Чтобы получить данные от партнера, мы регистрируем функцию обратного вызова, используя `espconn_regist_recvcb ()`. Мы проходим в `структуре espconn`, который поставлялся в связанном отзыве, который определяет нашу связь. Эта зарегистрированная функция обратного вызова призвана, когда новые данные становятся доступными от партнера. Функция обратного вызова передана буфер, содержащий данные и индикатор того, сколько данных было получено.

Ниже приведен пример регистрирующихся данных, которые получены по сети:

```
пустота recvCB (пустота *аргумент, случайная
   работа *pData, короткое целое без знака
   len) {структура espconn *pEspConn =
   (структура espconn *) аргумент; os_printf
   («Полученные данные!! - длина = %d\n»,
   len);
   интервал i=0;
```

```
espconn_send ()
   для (i=0; я <len; я
      ++) {os_printf (»
      %c», pData [я]);
   }
   os_pri
ntf (»
\n»);}//К
онец
recvCB
```

Функция, вызванная `recvCB ()`, зарегистрирована как отзыв, когда данные доступны для связи. Принимая это во внимание, мы можем начать управлять некоторыми экспериментами, и результаты будут интересны.

Если мы посылаем данные, мы видим, что отзыв призван как ожидалось. Однако как размер переданных данных, который получен ESP8266, увеличениями, на уровне приблизительно 1 460 байтов, странная вещь происходит. Вместо `recvCB ()` называемый однажды, мы видим, что он назван дважды. В первый раз, когда это получает первые 1 460 байтов и во второй раз, это получает то, что остается. Это повторено для приращений 1 460-байтовых размеров передачи. Например, если мы посылаем 5 000 байтов, `recvCB ()` назван 4 раза. Первые три раза с 1 460 байтами данных и последнего с 620 байтами, дающими в общей сложности 5 000.

Почему это было бы? Часть ответа - то, что ESP8266 имеет только очень небольшое количество RAM, доступной ему, и должен быть в состоянии поддержать параллельные связи. По сути, это может, по-видимому, задушить данные, посылаемые от отправителя, пока место не свободное, чтобы обработать его.

Это не может быть подчеркнуто достаточно важность этого понятия. Данные, посланные из сервера по соединению по протоколу TCP, «текутся» к ESP8266. Нет никакого понятия единицы передачи данных. Вместо этого данные, посланные в трубе в отправителе, прибудут в ESP8266, но это может прибыть в различные ставки. Заказ переданных данных сохранен (очевидно). В принципе делая две передачи в отправителе 5 байтов каждый мог привести к, каждый получает в ESP8266 10 байтов, или так же, как легко каждый получает 1 байта, и каждый получает 9 байтов. Не делайте предположения о заключении в скобки данных TCP.

Чтобы передать данные партнеру, мы используем функцию, вызванную `espconn_send ()`.

Эта команда берет структуру `espconn`, который определяет который

связь послать данные через. Функция также берет указатель на буфер данных и длину данных, чтобы послать. Жизненное соображение состоит в том, что данные, которые пошлют, немедленно не посылают. То, когда мы называем `espconn_sent ()`, что мы делаем, передает буфер данных, которые будут переданы в некоторое время в будущем. Мы ожидаем, что это будет несколькими миллисекундами, но это могло быть длиннее. Мы должны соблюдать контракт. Когда ESP8266 действительно успешно передаст данные, отзыв будет сделан к функции, которая была зарегистрирована в `espconn_regist_sentcb ()`. Только видя подтверждение, что последнее запрос передачи был закончен, должен мы выполнять другой запрос.

Когда мы просим данные быть переданными, мы обеспечиваем указатель на буфер, который содержит данные. Важно понять, что мы должны утверждать, что данные, пока мы не уверены его содержание, послали. Например, мы не можем просить передачу и затем немедленно расположить прочь или изменить буфер. То, что мы передаем к OS, является указателем на буфер и пока OS не говорит нам, что закончил потреблять его, мы должны поддержать его целостность.

См. также:

- espconn_regist_recvcb
- espconn_send

Управление потоками

Рассмотрите понятие ESP8266 в связи с партнером, и партнер посылает 5K данных в секунду. Теперь предположите, что ESP8266 только обрабатывает 1K данных в секунду. Как Вы видите, что-то пойдет не так, как надо вполне быстро. Мы сокрушим ESP8266 со слишком большим количеством данных. То, что мы действительно хотим, должно установить механизм управления потоками, таким образом, что отправителю данных говорят задушить назад его доставку данных к уровню, который может приспособить ESP8266. Посылание может буферизовать данные в его конце или иначе может быть сказано не послать так много данных за единицу времени, пододвинув обратно к оригинальной передающей логике.

См. также:

- espconn_recv_hold

- espconn_recv_unhold

Обработка ошибок TCP

Когда связь сформирована между двумя партнерами, важно, чтобы мы поняли, что нет фактической специальной основной связи между ними. Вместо этого есть только логическая связь, которая, кажется, присутствует по ориентированному протоколу дейтаграммы IP. То, что это могло бы означать, - то, что, если один конец связи неправильно заканчивается, другой конец не будет немедленно знать об этом. Как пример, если в реальном мире я делаю телефонный звонок Вам тогда, Ваш телефон указывает Вам, что у нас есть связь. Если батарея по моему телефону перестает работать, телефонная сеть обнаруживает, что и пропускает связь. Ваш телефон также вешает трубку, и Вы знаете, что мы больше не находимся в коммуникации. В мире TCP, которого не происходит. Если мой телефон «TCP» перестает работать, Ваш телефон «TCP» не сказан, что моего не стало. Вас можно оставить, сидя там неопределенно слушание тишины и ожидание меня, чтобы сказать что-то.

Чтобы решить, что ситуация, TCP вводит понятие, названное, «сохраняют - живым». Понятие очень просто. Со сторожевой башней - живой, эти два партнера периодически обменивают связь сердцебиения друг с другом. Пока каждый из них слышит сердцебиение о другом, они оба все еще присутствуют. Однако, если одна сторона связи потеряна, сердцебиение

запрос будет отправлен, но никакой ответ не прибудет, в котором пункте, тот, посылая сердцебиение предположит, что партнер пошел, и мы можем принять соответствующие меры очистки и закрытия.

Есть API, доступный нам, чтобы управлять сторожевой башней - живые параметры настройки. Это называют `espconn_set_keepalive ()`. У этого есть много свойств включая:

- Сколько времени мы должны ждать с прошлого раза мы получили известие от партнера прежде, чем послать сердцебиение?
- Если никакой ответ, сколько времени между последующим сердцебиением?
- Сколько раз мы должны послать сердцебиению, пока мы не объявляем партнера мертвым?

Рекомендуется, чтобы, если держат - живая обработка использовалась тогда сторожевая башня - живые параметры настройки, которые будут сделаны в укладчике отзыва соединить отзыва. Сторожевая башня - живой выбор должен также быть явно позволен, используя `espconn_set_opt ()` требование до урегулирования сторожевой башни - живые свойства.

Если связь партнера потеряна, мы можем обнаружить это, регистрируя функцию обратного вызова в `espconn_reconnect_callback ()`.

См. также:

- espconn_set_keepalive
- espconn_get_keepalive
- espconn_set_opt
- espconn_clear_opt

UDP

Если мы думаем о TCP как о формировании связи между двумя сторонами, подобными телефонному звонку, то UDP похож на отправку письма через почтовую систему. Если бы я должен был послать Вам письмо, то я должен был бы знать Ваше имя и адрес. Ваш адрес необходим так, чтобы письмо могло быть поставлено правильному дому, в то время как Ваше имя гарантирует, что заканчивается в Ваших руках в противоположность кому-то еще, кто может жить с Вами. В терминах TCP/IP адрес - IP-адрес, и имя - номер порта.

С телефонным разговором мы можем обменять столько или такую небольшую информацию, как нам нравится. Иногда я говорю, иногда Вы говорите …, но нет никакого максимального ограничения на то, сколько информации мы можем обменять в одном разговоре. С письмом однако, есть только столько страниц бумаги, которая поместится в конверты, которые я имею в моем распоряжении.

Понятие почтовой аналогии - то, как мы могли бы думать о UDP. Акроним обозначает Пользовательский Дейтаграммный Протокол, и это - понятие дейтаграммы, которая сродни письму. Дейтаграмма - множество байтов, которые переданы от отправителя приемнику как единица. Максимальный размер дейтаграммы, используя UDP составляет 64 кбайта. Никакая потребность связи быть установкой между этими двумя сторонами перед данными не начинает течь. Однако

есть вниз сторона. Отправитель данных не будет сделан знающий об отказе получателя восстановить данные. С TCP у нас есть подтверждение связи между двумя сторонами, которое позволяет отправителю знать, что данные были получены и, в противном случае могут автоматически повторно передать, пока это не было получено, или мы решаем сдаться. С UDP, и точно так же, как письмо, когда мы посылаем дейтаграмму, мы теряем из виду то, прибывает ли он на самом деле безопасно в место назначения.

Если мы хотим получить поступающие дейтаграммы, мы должны зарегистрировать, на какой номер порта мы интересуемся получением их. Мы достигаем этого через плохо названный `espconn_create ()` функция. Эта функция заставляет ESP8266 начинать прислушиваться к поступающим дейтаграммам на местном порте, определенном в `структуре espconn`. После вызывания этой функции Вы должны тогда назвать `espconn_regist_recvcb ()`, чтобы зарегистрировать функцию обратного вызова, которая будет призвана, когда дейтаграмма прибудет.

Вот пример высокого уровня подготовки слушателя UDP, как только IP-адрес был ассигнован:

```
структура
espconn
conn1;
esp_udp
udp1;
пустота
   setu
   pUDP
   ()
   {sin
   t8
   допу
   скае
   т
   ошиб
   ку;
   conn1.type =
   ESPCONN_UDP;
   conn1.state =
   ESPCONN_NONE;
   udp1.local_port =
   25867;
   conn1.proto.udp =
   &udp1;

   доп
   уст
   ить
   оши
   бку = espconn_create (&conn1);
```

```
    доп
    уст
    ить
    оши   espconn_regist_recvcb (&conn1,
    бку  = recvCB);
}//Конец из setupUDP
```

Если мы хотим мешать ESP8266 прислушаться к дейтаграммам, мы можем вызвать функцию, `вызванную espconn_delete ().`

Теперь хорошее время, чтобы возвратиться к IP-адресам и номерам портов. Мы должны начать знать, что на PC, только одно применение может слушать на любой данный порт. Например, если мое заявление слушает на порте 25867, то никакое другое применение не может также слушать на том же самом порте … не Ваше заявление, ни другая моя копия/случай. Когда поступающая связь или дейтаграмма прибывают в машину, она прибыла, потому что IP-адрес посланных данных соответствует IP-адресу устройства, в которое она прибыла. Мы тогда маршрут в устройстве на основе номеров портов. И вот то, где я хочу разъяснить деталь. Мы направляем в машине на основе **пары** и протокола и номера порта.

Так, например, если запрос прибывает в машину для порта 25867 по соединению по протоколу TCP, это разбито к TCP-приложению, наблюдая порт 25867. Если запрос прибывает в ту же машину для порта 25867 по UDP, это разбито к применению UDP, наблюдая порт 25867. То, что это означает, - то, что у нас **может** быть два заявления, слушающие на том же

порт, но на различных протоколах. Помещая это более официально, пространство распределения для номеров портов - функция протокола, и это не позволено для двух заявлений одновременно зарезервировать тот же порт в том же пространстве распределения протокола. Хотя я использовал историю PC, запускающего несколько приложений, в нашем ESP8266 история подобна даже при том, что мы просто запускаем одно приложение на устройстве. Если Ваше отдельное приложение должно должно быть послушать на нескольких портах, не пытайтесь использовать тот же порт с тем же протоколом, поскольку второй вызов функции найдет, что первый уже ассигновал порт. Это - деталь, которую я рад за Вас забыть, поскольку Вы будете редко сталкиваться с нею, но я хотел поймать ее здесь для полноты.

Теперь давайте посмотрим на то, что это берет, чтобы послать дейтаграмму. Подобный другим функциям, нам нужна `структура espconn` блок управления. Это должно быть настроено, чтобы использовать UDP и назвать отдаленный IP-адрес и порт. После того, как населенный, мы

можем тогда инициализировать структуру данных с требованием к `espconn_create ()`, и теперь мы готовы послать данные. Мы используем `espconn_sent ()` функция. Когда мы послали все наши данные, мы можем завершить с `espconn_delete ()`, чтобы высвободить средства, которые ESP8266 поддерживает для отправки данных.

Вот пример:

```
структура espconn
sendResponse;
esp_udp udp;
пустота sendDatagram (случайная
   работа *дейтаграмма, uint16
   размер) {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;//Отдаленный порт
   допустите ошибку = espconn_create (&sendResponse);
   допустите ошибку = espconn_send
   (&sendResponse, «hi123», 5);
   допустите ошибку = espconn_delete
   (&sendResponse);
}
```

См. также:

- espconn_create
- espconn_delete
- espconn_send
- espconn_regist_recvcb
- espconn_regist_sentcb
- структура espconn

Передача с UDP

Одной из особенностей, доступных нам с UDP, является понятие передачи. Это - понятие, что отправитель данных может построить дейтаграмму и передать их таким образом, что все устройства на той же подсети могут получить копию их. Получатели выбирают порт UDP и начинают слушать на него, как они обычно были бы. Передающее применение передает а

сообщение на том же порте, но с IP-адресом, где часть хозяина IP-адреса - все двоичные единицы. Например, если у нас будет netmask `255.255.255.0`, и наша сеть `192.168.1.x`, то затем передавание на IP-адресе `192.168.1.255` будет передачей. Специальный IP-адрес

`255.255.255.255` представляет трансляцию на нашей местной сети.

Для ESP8266 есть API, названный `wifi_set_broadcast_if ()`, который определяет, какие интерфейсы будут доступны для передачи. Выбор - станция, точка доступа или и станция и точка доступа.
Соответствующий API, названный `wifi_get_broadtcast_if ()`, может использоваться, чтобы восстановить текущее состояние конфигурации вещания.

См. также:

- wifi_set_broadcast_if
- wifi_get_broadcast_if

Запрос звона

На уровне TCP/IP устройство с IP-адресом может «свистеть» другое устройство с IP-адресом. То, что это означает, - то, что сообщения переданы между ними, который позволяет им знать, что у них есть маршрут через сеть друг другу. Если место назначения или не бежит или никакой маршрут, доступно, нам также сообщат, что была неудача.

ESP8266 обеспечивает структуру, названную `структурой ping_option`, который содержит детали запроса звона. Это передано в в качестве параметра к функции, `вызванной ping_start ()`, который начинает звон.

Прежде, чем вызвать эту функцию, целевой IP-адрес и количество запросов звона должны быть установлены в `структуре ping_option`.

Две функции обратного вызова могут быть зарегистрированы в `ping_regist_recv ()` и `ping_regist_sent ()`. Первое называют, когда ответ звона получен, и другой назван, когда запрос звона отправлен.

См. также:

- ping_start
- ping_regist_recv
- ping_regist_sent
- структура ping_option

Служба имен

В Интернете машины сервера могут быть найдены их названиями Domain Name Service (DNS). Это - обслуживание, которое решает человекочитаемое представление машины, такой как «`google.com`» в необходимую стоимость IP-адреса (например, `216.58.217.206`). Для этого преобразования, чтобы произойти, ESP8266 должен знать IP-адрес одного или нескольких серверов DNS, которые это будет тогда использовать, чтобы выполнить имя к IP-адресу

отображение. Если мы используем DHCP тогда, ничему иному не нужно, сделаны, поскольку сервер DHCP автоматически обеспечивает адреса сервера DNS. Однако, если мы не должны использовать DHCP, тогда мы должны проинструктировать ESP8266 местоположений серверов DNS вручную. Мы можем сделать это использование `espconn_dns_setserver ()` функция. Это берет множество одного или двух IP-адресов, как введено и от того пункта вперед, эти серверы будут использоваться для резолюции DNS. Если два адреса будут поставляться, и первое безразлично, то второе будет использоваться.

Google публично делает доступным два сервера имени с адресами 8.8.8.8 и 8.8.4.4.

Как только мы имеем, определяют nameservers, мы можем искать адрес имени хоста, используя `espconn_gethostbyname ()` функция. Код возврата для этого требования должен быть тщательно исследован. У нас может немедленно быть адрес из-за тайника, или мы, возможно, должны выполнить сетевой запрос и обеспечить отзыв для более позднего поиска. Если позже, `ipAddr` возвращен как ПУСТОЙ УКАЗАТЕЛЬ … однако, Ваш поставщик DNS может обеспечить IP-адрес поисковой системы, и следовательно Вы вернете адрес …, но не один хозяину, которого Вы ожидали!!

См. также:

- espconn_dns_setserver
- espconn_gethostbyname
- Википедия:система доменных имен
- Google:общественный DNS

Системы доменных имен передачи

На локальной сети с динамическими приходящими и уходящими устройствами мы можем хотеть, чтобы одно устройство нашло IP-адрес другого устройства так, чтобы они могли взаимодействовать друг с другом. Проблема, хотя то, что IP-адреса могут быть динамично ассигнованы сервером DHCP, работающим на точке доступа WiFi. Это означает, что IP-адрес устройства, вероятно, не собирается быть статичным. Кроме того, это не большое удобство использования к истории, чтобы относиться к устройствам их IP-адресами. То, в чем мы нуждаемся, является некоторой формой динамической службы имен для нахождения устройств по имени, где их IP-адреса не настроенный администратор. Это - то, где Система доменных имен Передачи (mDNS) играет роль.

На высоком уровне, когда устройство хочет найти другое устройство с именем, оно передает запрос всем членам сети, просящей ответ от устройства, у которого есть то имя. Если машина полагает, что у нее

есть та идентичность, она отвечает своей собственной передачей, которая включает ее имя и IP-адрес. Мало того, что это удовлетворяет оригинальный запрос, но и другие машины в сети, видят это взаимодействие и прячут ответ про запас для себя. Это означает, что это должно, они должны решить того же хозяина в будущем, у них уже есть ответ.

Используя Систему доменных имен Передачи (mDNS) ESP8266 может попытаться решить имя хоста машины в местной сети к ее IP-адресу. Это делает это, передавая пакет, просящий машину с той идентичностью ответить.

Демоны службы имен осуществлены Добрый день и nss-mdns (Linux).

Обычно, расположенное использование хозяев этой техники

принадлежит области, заканчивающейся в «.local».

Чтобы определить, участвует ли Ваш PC в mDNS, Вы можете исследовать, слушает ли он на порте UDP 5353. Это - порт, используемый для mDNS коммуникаций.

См. также:

- Википедия –передача DNS
- IETF RFC 6762:передача DNS
- Передача DNS
- Новые технологии DNS в LAN
- Avahi – Внедрение исходного проекта mDNS … для машин Unix
- Adafruit –добрый день (Zeroconf), общающийся через Интернет для Windows и Linux
- chrome.mdns – описание API для Хромового API для mDNS
- Android –браузер ZeroConf
- espconn_mdns_init
- espconn_mdns_close
- espconn_mdns_server_register
- espconn_mdns_server_unregister
- espconn_mdns_get_servername
- espconn_mdns_set_servername
- espconn_mdns_set_hostname
- espconn_mdns_get_hostname
- espconn_mdns_disable
- espconn_mdns_enable

Установка добрый день

Начните Добрый день инсталлятор:

Страница 119

Страница 120

Если все подходило, мы найдем новое управление службы Windows названным «Добрый день Обслуживание»:

Работа с SNTP

SNTP - Простой Сетевой Протокол Времени и позволяет устройству,

связанному с Интернетом изучать текущее время. Чтобы использовать это, Вы должны знать о по крайней мере одном разе, когда сервер определил местонахождение в Интернете. Американский Национальный Институт Науки и техники (NIST) поддерживает много они, которые могут быть найдены здесь:

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

Другие серверы времени могут быть найдены во всем мире, и я призываю Вас к Google, ищут Ваше ближайшее или страну определенный сервер.

Как только Вы знаете идентичность сервера его именем хоста или IP-адресом, Вы можете вызвать любую из функций, названных `sntp_setservername ()` или `sntp_setserver ()`, чтобы объявить, что мы хотим использовать тот случай сервера времени. ESP8266 может быть настроен максимум с тремя различными серверами времени так, чтобы, если один или два не доступны, мы могли бы все еще получить результат.

ESP8266 нужно также сказать местный часовой пояс, в котором он бежит. Это установлено с требованием к `sntp_set_timezone ()`, который берет количество погашения часов от UTC. Например, я нахожусь в Техасе, и мое погашение часового пояса становится `«-5»`.

С ними настроенными, мы можем начать обслуживание SNTP на ESP8266, звоня `sntp_init ()`. Это заставит устройство определять свое текущее время, посылая пакеты по сети к серверам времени и исследуя их ответы. Важно отметить, что немедленно после запроса `sntp_init ()`, Вы еще не будете знать, каково текущее время может быть. Это вызвано тем, что может потребоваться несколько секунд для ESP8266 к, отправляет запросы времени, и получите их ответы, и это все произойдет асинхронно с Вашими текущими командами и не закончит до когда-то позже.

Когда готовый, мы можем восстановить текущее время с требованием к `sntp_get_current_timestamp ()`, который возвратит число секунд с 1 января 1970$^{Св.}$ UTC. Мы можем также вызвать функцию, `вызванную sntp_get_real_time ()`, который возвратит представление последовательности времени.

См. также:

- sntp_setserver
- sntp_setservername
- sntp_init
- sntp_set_timezone
- sntp_get_current_timestamp
- sntp_get_real_time
- IETF RFC5905: сетевая версия 4 протокола времени: протокол и спецификация алгоритмов

ESP теперь

Понятие ESP теперь состоит в том, чтобы достигнуть частного протокола между наборами ESP8266s. Думайте о нем свободно как об отношениях диспетчера/раба, где у нас есть один диспетчер и потенциально несколько рабов. Рабы формируют «постоянные» связи с диспетчером. То, что это означает, - то, что, когда раб ESP8266 приведен в действие на, он фактически немедленно в состоянии передать диспетчеру. Сравните это с понятием ESP8266, двигающегося на большой скорости на, соединившись с точкой доступа и затем соединившись с ведущим устройством. Эти потоки берут таймер, в то время как ESP теперь намного быстрее от запуска.

Чтобы начаться, ESP8266, который хочет участвовать в использовании этого протокола, призовет `esp_now_init ()`. Если это больше не хочет быть частью этого вида сети, это может назвать `esp_now_deinit ()`. Прежде чем коммуникация может продолжиться, устройство будет, должны были добавить ровесники в сети. Это достигнуто посредством требования к `esp_now_add_peer ()`.

Каждое устройство ESP8266 в сети объявит себя как наличие роли или раба или диспетчера посредством требования к `esp_now_set_self_role ()`.

Соответствующий `esp_now_delete_peer ()` может использоваться, чтобы забыть о ранее зарегистрированном ровеснике. Когда готовый, чтобы передать данные, звонок может быть сделан `esp_now_send ()` поставка адреса получателя, а также данных, которые будут переданы. Максимальный объем данных, который можно в настоящее время послать как единица, составляет 256 байтов.

Два отзыва доступны, которые призваны, когда или новое сообщение было передано или новое сообщение, был получен.

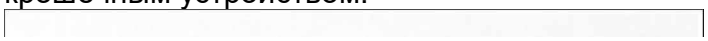
См. также:

- esp_now_init
- esp_now_send
- esp_now_register_recv_cb

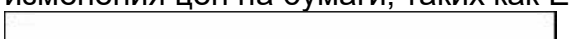
GPIOs

У ESP8266 есть 17 булавок GPIO. Когда мы думаем о GPIO, мы должны понять, что в любой момент, у каждого случая есть два эксплуатационных способа. Это может или быть вход или продукция. Когда это - вход, мы можем прочитать стоимость от него и определить логический уровень подарка сигнала в физической булавке. Когда это - продукция, мы можем написать логический уровень ему, и это появится как физическая продукция.

Не забудьте различать интегральную схему ESP8266, которая является крошечным устройством:



который отличается от различных моделей правления резкого изменения цен на бумаги, таких как ESP-1:



у которого есть 8 выставленных булавок, 4 из которых являются GPIO. Модуль назвал ESP-12:

у которого есть 16 выставленных булавок, 11 из которых являются GPIO. Для GPIO вот выставленные отображения:

Булавка	**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		●
Общие количества	**4**	**11**

Также хорошо напомнить нам о схемах контактов устройства.

Поскольку Вы видите, что нет никакого очевидного образца к расположению булавок и как таков, Вы должны проявить большую заботу, обеспечивая электричеством схему. Легко сделать ошибку.

Другое жизненное соображение, когда работа с GPIOs - напряжение. ESP8266 - 3.3-вольтовое устройство. Вы должны быть чрезвычайно осторожными, если Вы работаете с 5 В (или выше) партнер MCUs или датчики. К сожалению, устройства как Ардуино, как правило - 5 В, как USB → UART конвертеры и много датчиков. Это означает, что Вы должны, скорее всего, работать в смешанной окружающей среде напряжения. Ни при каких обстоятельствах не должны Вы думать, что Вы можете привести ESP8266 в действие с постоянным напряжением больше чем 3.3 В. Очевидно, Вы можете преобразовать более высокие напряжения вниз в 3.3 В, но никогда не пытаться соединить большее напряжение непосредственно. Другое более тонкое соображение, используя GPIOs для входа сигнала и поставки, больше, чем 3.3 В как высокая стоимость сигнала. Я настоятельно рекомендую не делать его. Некоторые люди могут утверждать, что Вы можете «выйти сухим из воды» и если Вы экспериментируете, это, может (казаться), работает, но Вы берете на себя ненужный риск ни на каком очевидно серьезном основании. Если это будет работать … тогда, то это будет работать, пока это не сделает, в котором пункте будет слишком поздно, и Вы можете приготовить свое устройство.

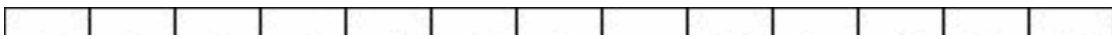
В моих собственных экспериментах я случайно пересилил ESP8266s, полностью измените напряжение, привел ESP8266s в действие и применил слишком высокое напряжение, как введено. В каждом случае результатом был неисправный чип и в нескольких случаях, пытаясь

видеть, привел ли он все еще работавший, применяя нормальное напряжение к устройству не только не работа, но и получение, настолько горячее на ощупь, он сжег мои пальцы.

Поскольку несчастные случаи происходят, когда строительство GPIO основывало схемы, я рекомендую покупать больше случаев ESP8266, чем Вам нужно. Тот путь, если Вам, действительно оказывается, требуется секунду (или третий или четвертый) у Вас будут они в Вашем распоряжении.

До использования любых функций ESP8266 GPIO Вы должны, назвал поставляемый `gpio_init ()` функцией. То, что это на самом деле делает, неизвестно однако, в правилах говорится, называют его и называют его, мы должны.

Способ, которым ESP8266 думает о GPIOs, состоит в том, как будто каждый GPIO был немного в 16-битном множестве.



(Мы возвратимся к тому, как 17 GPIOs наносят на карту к 16 битам в более позднее время),

Одно множество содержит признак того, введен ли GPIO или произведен. Мы назовем это множеством направления. Второе множество указывает на ценности GPIOs. Для входа GPIOs стоимость - стоимость на булавке. Для продукции GPIOs стоимость - стоимость, которая будет написана булавке. Мы назовем это множеством стоимости.

Функция поставляется ESP8266, названным `gpio_output_set ()`. Эта функция берет **четыре** 16-битных ценности, которые будут использоваться в качестве масок против двух 16-битных множеств.

Первую маску называют «`set_mask`». `1` стоимость в маске набора устанавливает соответствующее битовое значение быть `1` во множестве стоимости.

Вторую маску называют «`clear_mask`». `1` стоимость в ясной маске устанавливает соответствующее битовое значение быть `0` во множестве стоимости.

Заметьте, что в обоих случаях, если у масок есть `0`

стоимостей, первоначальные ценности неизменны.

Третью маску называют маской «enable_output». 1 стоимость в позволить маске продукции устанавливает соответствующий GPIO быть в способе продукции.

Четвертую маску называют маской «enable_input». 1 стоимость в позволить входной маске устанавливает соответствующий GPIO быть во входном способе.

Всего хорошего, чтобы не установить GPIO быть оба входом и выходом или иметь ценность и 1 и 0. Результаты будут не определены.

Константы определены для каждой из позиций двоичного разряда. Те константы:

- BIT0 – 2^0
- BIT1 – 2^1

1 • …

- BIT31 – 2^31

Так, например. Если мы хотим установить GPIO 5 быть введенным, мы могли бы закодировать:

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

чтобы установить GPIO 4 быть произведенным и иметь высокую стоимость, мы могли бы закодировать:

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

установить GPIO 0 и 1 и быть произведенным и первое, чтобы быть 1 и второе, чтобы быть 0:

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

Страница 127

Если мы хотим восстановить ценности GPIOs, мы можем использовать `gpio_input_get ()` метод. Это возвращает немного маски, содержащей все биты.

У нас есть некоторый макрос помощника, которые доступны. Это полезные обертки вокруг `gpio_output_set ()` и `gpio_input_get ()`.

- `GPIO_OUTPUT_SET (GPIO_NUMBER, стоимость)` – Наборы соответствующий GPIO, чтобы быть продукция и устанавливает свое значение.
- `GPIO_DIS_OUTPUT (GPIO_NUMBER)` – Наборы соответствующий GPIO,

который будет введен
(отключенная продукция).

- `GPIO_INPUT_GET (GPIO_NUMBER)` – Получает ценность входа GPIO

Так как булавки на ESP8266 могут служить нескольким целям, мы должны сначала объявить то, что функционирует, который будет иметь булавка. Чтобы сделать это, мы используем макрос, который устанавливает функцию логической булавки:

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

Например, чтобы определить GPIO2 как GPIO прикрепляют и установить его значение, мы могли бы закодировать:

```
PIN_FUNC_SELECT (PERIPHS_IO_MUX_GPIO2_U,
FUNC_GPIO2); GPIO_OUTPUT_SET (2, 1);
```

Вот заполненная таблица отображений.

Имя булавки	Функция 1	Функция 2	Функция 3	Функция 4	Физическая булавка	Устройства
MTDI_U	MTDI	I2SI_DATA	МИСО HSPIQ	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	SPID		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

Следующее - ключи к некоторым ценностям в столе:

- Колонка устройств
  - 1=ESP-1

- 12=ESP-12

Вот булавки GPIO, нанося на карту:

GPIO	Имя булавки	NodeMCU	Примечания	Риск
GPIO0	GPIO0_U	D3	Булавка управляет государством ESP8266 в ботинке. Осторожность, когда использовался как булавка продукции.	
GPIO1	U0TXD_U	D10	Булавка обычно используется для высвечивания устройства.	
GPIO2	GPIO2_U	D4	Используемый для продукции UART1 и, как таковой, вероятно, будет использоваться в течение времени разработки для отладки. Написанный, к когда высвеченный с новым встроенным микропрограммным обеспечением.	
GPIO3	U0RXD_U	D9	Булавка обычно используется для высвечивания устройства.	
GPIO4	GPIO4_U	D2	Только использование как GPIO.	
GPIO5	GPIO5_U	D1	Только использование как GPIO.	
GPIO6	SD_CLK_U		Не выставленный на текущих устройствах.	
GPIO7	SD_DATA0_U		Не выставленный на текущих устройствах.	
GPIO8	SD_DATA1_U		Не выставленный на текущих устройствах.	
GPIO9	SD_DATA2_U	SD2	Не выставленный на текущих устройствах.	
GPIO10	SD_DATA3_U	SD3	Не выставленный на текущих устройствах.	
GPIO11	SD_CMD_U		Не выставленный на текущих устройствах.	
GPIO12	MTDI_U	D6		
GPIO13	MTCK_U	D7		
GPIO14	MTMS_U	D5		
GPIO15	MTDO_U	D8	Используемый, чтобы управлять UART0 RTS и следовательно может иметь влияние на встроенное микропрограммное обеспечение, вспыхивающее начиная с микропрограммных данных прибывает через UART0.	
GPIO16	???	D0	???	

Максимальный выходной ток от булавки GPIO только 12mA.

Учитывая выбор, если Вы используете GPIO0, используйте его в качестве входной булавки в противоположность булавке продукции. Причина этого состоит в том, что, когда Вы развиваете решения, Вы должны принести GPIO0 низко, чтобы поместить ESP8266 в способ вспышки, где он читает новые программы от UART. Это означает, что Вы будете изменять входной сигнал на GPIO0. Если Вы используете булавку в качестве продукции, есть возможность, что, когда Вы изменяете свою проводку, чтобы принести ее низко или нажать кнопку, чтобы принести ее низко, если сигнал высок в то время, Вы закоротите схему.

Однако, если булавка будет введена тогда, то это не будет проблемой. Идеально, избегайте использования GPIO0 в целом и оставьте его специально для самонастройки устройства в различных способах.

См. также:

- PIN_FUNC_SELECT
- GPIO_OUTPUT_SET
- GPIO_DIS_OUTPUT
- GPIO_INPUT_GET
- gpio_output_set
- gpio_input_get

Усилие и сбрасывает параметры настройки

Мы обычно думаем о входе булавка GPIO как имеющий или высокий или низкий сигнал, поставляемый ему. Это означает, что связано с +ve или землей. Но что, если это не связано ни с одним? В этом случае булавка считается в плавающем государстве. Есть времена, где мы желаем к определенному не связанной булавки, как являющейся высоким или низким. Не связанную булавку, которую нужно считать высокой, называют «потянувшей», в то время как не связанную булавку, которую нужно считать низкой, называют «со спуском». Это прибывает из физической практики аппаратных средств бывших свойственных резисторов, чтобы остановиться или сбросить сигнал, когда это иначе плавало бы.

В ESP8266 SDK мы можем определить GPIO, как потянувший при помощи макроса под названием `PIN_PULLUP_EN`, и мы можем определить GPIO как больше не потянувший, используя макро-`PIN_PULLUP_DIS`.

Обработка Перерыва GPIO

Если мы полагаем, что сигнал на булавке может переместиться от высоко до низкого или от низко до высокого, такое изменение могло бы быть чем-то, что наше заявление будет интересоваться знанием. Чтобы определить, когда такое изменение происходит, мы можем все время получать голоса стоимости и обнаруживать изменение перехода. Однако это в не лучшее решение по ряду причин. Во-первых, мы должны деловито выполнить работу, проверяющую, изменилась ли стоимость. Во-вторых, будет время ожидания со времени, случай

происходит ко времени, когда мы проверяем. В-третьих, возможно полностью пропустить изменение сигнала, если продолжительность изменения коротка. Например, если мы проверяем ценность булавки и находим его высоко и затем немедленно после того, как она идет низко и затем высоко снова, в следующий раз, когда мы голосуем, мы будем все еще видеть булавку высоко и никогда не знали, что это было когда-либо низко в течение короткого периода.

Решение всех этих проблем - понятие перерыва. Перерыв подобен Вашему дверному звонку в Вашем доме. Без дверного звонка (или прислушивание к кому-то стук) Вы должны были бы периодически проверять, чтобы видеть, есть ли кто-либо у двери. Это тратит впустую Ваше время для большинства случаев, где нет никого там и также не удостоверяется, что, когда есть кто-то там, Вы проявляете внимание к ним своевременно.

Страница 130

На земле ESP8266s мы можем определить функцию обратного вызова перерыва, которую назовут, когда булавка изменит свою стоимость сигнала. Мы можем также определить то, что составляет причину призыва отзыва. Мы можем настроить укладчика отзыва (технически названный укладчиком перерыва) на булавке основанием булавки.

Во-первых, давайте рассмотрим функцию обратного вызова перерыва. Это зарегистрировано в требовании к `gpio_intr_handler_register ()`.

Мы можем позволить или отключить обработку перерыва на глобальном уровне. Требование к `ETS_GPIO_INTR_ENABLE` позволяет обработку перерыва, в то время как требование к `ETS_GPIO_INTR_DISABLE` отключает глобальную обработку перерыва.

Чтобы позволить перерыв для определенной булавки, мы используем функцию, `вызванную gpio_pin_intr_state_set ()`. Это позволяет нам устанавливать причину, что перерыв мог бы произойти. Причины включают:

- Отключите – не называют перерыв на изменении сигнала.
- PosEdge – Требование укладчик перерыва на изменении от низко до высоко.
- NegEdge – Требование укладчик перерыва на изменении от высоко до низко.

- AnyEdge – Требование укладчик перерыва или на изменении от низко до высокого или на изменении от высоко до низко.
- Привет – Требование укладчик перерыва, в то время как сигнал высок.
- Ло – Требование укладчик перерыва, в то время как сигнал низкий.

Примечания: Когда я пошел, чтобы осуществить укладчика перерыва в проекте, я нашел, что теория и практика не встретились. На данный момент единственная история, у меня есть работа для укладчика перерыва, смотрит следующим образом:

```
статическая пустота intrHandlerCB (
    uint32 interruptMask,//! <Маска, указывающая, какие GPIOs изменились.
    пустота *аргумент    //! <Дополнительный аргумент.
  ) {
  uint32 gpio_status = GPIO_REG_READ
  (GPIO_STATUS_ADDRESS); os_printf («статус:
  0x%x\n», gpio_status); gpio_intr_ack
  (interruptMask);
  международная булавка;
  для (pin=0; булавка <16; прикрепите ++) {
    если ((interruptMask & (1
      <<булавка))! = 0)
      {//Делают что-то
      gpio_pin_intr_state_set (GPIO_ID_PIN (булавка),
      GPIO_PIN_INTR_ANYEDGE);
    }
  }
}
```

См. также:

- gpio_intr_handler_register
- gpio_pin_intr_state_set

- gpio_intr_pending
- gpio_intr_ack

Расширение количества доступного GPIOs

Хотя у устройств ESP только есть ограниченное число булавок GPIO, которые не должны быть ограничением для нас. У нас есть способность расширить количество GPIOs, доступного нам через некоторые относительно недорогие интегральные схемы. Один из них называют PCF8574. (PFC8574A - то же, но имеет другой набор адресов).

Это - ^{2}устройство ^{2IC} и следовательно работает только по двум проводам. Используя этот IC мы поставляем 3-битный адрес (000-111),

который используется, чтобы выбрать рабский адрес устройства. Так как у каждого адреса есть 8 iOS, и у нас может быть до 8 устройств, это означает в общей сложности 64 дополнительных булавки.

Кажется, что устройство будет использовать резистор подтягивания для верхнего уровня и приносить булавку, чтобы основать для низко. Это означает, что, если мы хотим использовать какую-либо из булавок для входа, мы должны установить их писать способ высокому сначала. Это позволит или высокому или низкому входному сигналу быть обнаруженным. Казалось бы, что, если бы мы устанавливаем выходной сигнал низко и тогда питаться сырье высоко, сигнализируют в устройство, у нас было бы короткое.

Вот диаграмма булавки для устройства:

Вот описание булавок:

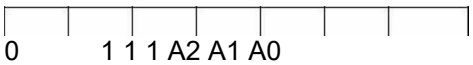
Символ	Булавка	Описание
A0-A2	1, 2, 3	Обращение
P0-P7	4, 5, 6, 7, 9, 10, 11, 12	Висмут направленный ввод/вывод
INT	13	Перерыв произведен
SCL	14	Последовательная линия часов

SDA	15	Последовательная линия данных
$_{VDD}$	16	Напряжение питания (2.5 В - 6 В)
Vss (Земля)	8	Земля

Адрес, на который может быть найдено рабское устройство, конфигурируем через булавки `A0-A2`. Это появляется по следующему адресу:

PCF8574

PCF8574A



Булавки `A0-A2` не должны плавать.

Это приводит к следующей таблице:

A2	A1	A0	Адрес PCF8574	Адрес 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

Вот программа в качестве примера, которая заставляет светодиоды создавать эффект Cylon.

```
#include
<Wire.h>
#include
<Ticker.h
>
//     SDA - желтый – 4
//     CLK - белый – 5

#define SDA_PIN 4
```



```
#define CLK_PIN 5
```

```
Тикер тикера; международный прилавок = 0;
международный директор = 1;

пустота timerCB () {Wire.beginTransmission(0x20); Wire.write (~ ((uint8_t) 1
       <<прилавок)); Wire.endTransmission ();
       возразите + = директор;
       если (возражают == 8) {прилавок = 6; директор =-1;
       } еще, если (возражают ==-1) {прилавок = 1;
              директор = 1;
       }
}


недействительная установка ()
{
       Wire.begin (SDA_PIN, CLK_PIN); ticker.attach (0.1, timerCB);
}

недействительная петля ()
{
}
```

Соответствующая схема:

И на макете:

См. также:

- Видео обучающая программа по этой теме
- Класс для PCF8574 (RobTillaart/Github)https://github.com/RobTillaart/Arduino/tree/master/libraries/PCF8574
- 8-БИТНЫЙ РАСШИРИТЕЛЬ IO (PCF8574)

- skywodd –библиотека PCF8574 Arduino
- Спецификация – NXP
- Страница продукта – TI
- Работа с I2C

Библиотека ESP_PCF8574 C

Класс под названием ESP_PCF8574 был написан, чтобы пользоваться библиотеками Ардуино. Библиотека может быть найдена наGitHub.

Это предоставляет следующие методы:

ESP_PCF8574.begin

Начните контроль за PCF8574.

```
пустота начинается (uint8_t адрес, uint8_t sda, uint8_t clk)
```

Параметр адреса - адрес I2C PCF8574 …, как правило, `0x20` – `0x27`.

`sda` и `clk` параметры - ПИН-коды, используемые для I2C SDA и CLK.

ESP_PCF8574.getBit

Восстановите вход данного бита.

```
bool getBit (uint8_t бит)
```

Если мы думаем о 8 GPIOs, поставляемых PCF8574, как являющимся 8 битами данных, этот метод восстанавливает ценность данных по данной входной булавке.

ESP_PCF8574.getByte

Восстановите все 8 битов входа.

```
uint8_t getByte ()
```

Если мы думаем о 8 GPIOs, поставляемых PCF8574, как являющимся 8 битами данных, этот метод восстанавливает ценность всех исходных данных.

ESP_PCF8574.setBit

Установите значение данной булавки продукции.

```
пустота setBit (uint8_t бит, bool стоимость)
```

Установите значение данной булавки продукции.

ESP_PCF8574.setByte

Установите значение всех булавок продукции.

```
пустота setByte (uint8_t стоимость)
```

Установите значение всех булавок продукции.

Библиотека PCF8574 JavaScript

Библиотека JavaScript была построена для установления связи PCF8574 из приложения JavaScript.

Призыв из JavaScript - сама простота. Так как PCF8574 - просто простое правление I2C, использование интерфейса I2C является всем, что необходимо.

Например:

```
вар sda =
NodeMCU.D1;//Желтый вар
scl = NodeMCU.D2;//Белый
I2C1.setup ({scl: scl,
sda: sda}); I2C1.writeTo
(0x20, стоимость);
```

Предположение, что адрес для нашего PCF8574 - `0x20`.

Работа с I2C

Интерфейс I2C - последовательная интерфейсная технология для доступа к устройствам. У этого есть две сигнальных линии под названием SDA (Данные) и SCL (Часы). ESP8266 может действовать как владелец и связанный с устройствами нисходящий акт как рабы. До 127 отличных рабов теоретически присоединяемые. У каждого рабского устройства есть уникальный адрес, и владелец решает, какой раб должен получить данные или быть разрешен говорить затем.

Все связанное с рабами использование «открытая утечка» связь с автобусом. Это означает, что, когда они соединяются, их приложение - или разомкнутая цепь или земля как продукция. Из-за этого для там невозможно быть электрическим конфликтом, как для одного устройства было бы невозможно утверждать высокий сигнал, в то время как другой пытался утверждать низкий сигнал. Присутствие логического высокого сигнала происходит, когда текущее рабское устройство идет разомкнутая цепь. Это означает, что нам нужны резисторы подтягивания на линиях, таким образом, что, когда никто активно не утверждает низкий сигнал, их тянут - до логического высокого сигнала. Ценность резистора 4.7K Ω рекомендуется.

Начало передачи обозначено, когда SCL оставляют высоким, и SDA тянут низко. Это сообщает всем рабским устройствам, что адрес собирается быть выпущенным. Когда адрес замечен всеми рабами, только один из них должен соответствовать, и другие устройства игнорируют запрос.

Страница 137

Адрес раба следует за начальным признаком начала и состоит из 7 битов с самым значительным битом сначала. После 7-битного адреса заключительные$^{th\ 8\ битов}$, который указывает, является ли это прочитанным или написать запросом. Ценность 1 указывает на прочитанный от раба, в то время как ценность 0 указывает на писание от владельца.

Немедленно после 8 битов адреса, прибывает, признание укусило. Этот бит **не** передан от владельца рабу, но вместо этого передан от раба владельца. Убедитесь, что Вы понимаете, что, смотря на диаграммы, показывающие данные по проводу SDA, те диаграммы, как правило, не показывают происхождение данных, только их последовательность. Поворот во время от последней части 8-битных данных об адресе/направлении, посланных от владельца к биту признания, посланному от раба, происходит, не пропуская тактов, так должно быть быстрым. Ценность 0 в признании указывает, что раб обработает или ответит. Ценность 1 в признании указывает, что никто не отвечает, или раб не присутствует.

После этого обращения структура прибывает структура данных или структуры. Поскольку владелец пишет запрос, владелец будет посылать 8 битов данных и ожидать единственную часть признания. Для прочитанного от раба раб пошлет 9 битов данных (8 битов данных и признание).

Владелец наконец пошлет конец коммуникации (или остановка) признак, который является переходом к высоко на часах без соответствующего перехода к низкому и **затем** перехода от низко до высоко на линии SDA.

Чтобы использовать I2C, мы сначала передаем запрос начала, используя i2c_master_start (). Мы следуем за этим адресом и флагом чтения-записи. Так как адрес составляет 7 битов, и флаг чтения-записи составляет один бит, это составляет 8 битов, и следовательно мы можем написать байт:

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

Затем мы можем прочитать и проверить использование флага признания:

```
если (i2c_master_checkAck () == верный) {
      ...
}
```

И отсюда мы можем или закончить или выполнить

следующую часть прочитанных или написать. Когда мы

хотим закончить, мы выполняем `i2c_master_stop ()`.

Мы можем послать вопрос адреса для каждого из возможных адресов устройства и видеть, получаем ли мы признание. Если мы делаем, то у нас есть устройство I2C по тому адресу. Это может использоваться, чтобы создать карту устройств.

Вот пример заявления, который делает просто что:

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

Страница 138

```
#include
<user_interface.h>
#include
<espconn.h>
#include <mem.h>
#include "driver/uart.h"
#include
"driver/i2c_master.h"
пустота user_rf_pre_init (пустота) {
}

os_timer_t scanTimer;

пустота scanTimerCB (пустота
   *pArg) {os_printf (»---
   Исследующий---\n»); uint8
   _t i;
   для (i=1; я <127; я ++)
      {i2c_master_start ();
      i2c_master_writeByte (я
      <<1);
      если (i2c_master_checkAck ())
```

```
         {os_printf («Найденный устройством
         в: 0x%2x\n», i);
      }
      i2c_master_stop ();
   }
   os_printf («Сделанный! \n»);

}//Конец timerCallback

пустота init ()
   {i2c_master_gpi
   o_init ();
   os_timer_setfn (&scanTimer,
   scanTimerCB, ПУСТОЙ УКАЗАТЕЛЬ);
   os_timer_arm (&scanTimer, 10000, 1);
}

пустота user_init (пустота) {uart_init
   (BIT_RATE_115200, BIT_RATE_115200);
   system_init_done_cb (init);
}//Конец user_init
```

См. также:

- Автобус I2C
- Sparkfun –обучающая программа: I2C
- API владельца I2C
- wget http://<IP-адрес> - тихий - документ продукции =-

Работа с SPI - последовательный периферийный интерфейс

SPI - последовательный протокол, используемый, чтобы общаться между владельцами и рабами. Все рабы соединяются с тем же автобусом, но только рабу с его булавкой `SS` низко разрешают передать. SPI - полный двойной протокол. То, что это означает, - то, что, в то время как данные выдвигаются из владельца рабу, раб одновременно передает данные обратно владельцу. Булавка MOSI содержит последовательные данные от владельца рабу, в то время как булавка МИСО содержит данные от раба владельца.

Как правило, три булавки:

- МИСО – **Владелец В**, **Раб** – Отправка данных владельцу от раба
- MOSI – **Владелец**, **Раб В** – Отправка данных рабу от владельца
- SCK (SCLK) – Последовательные Часы – Синхронизируют данные из отношений владельца/раба

Есть также дополнительный сигнал:

- SS (CSN (Сигнал выбора кристалла HE), NSS) – **Избранный Раб** – Используемый, чтобы позволить/отключить рабу так, чтобы могло быть несколько рабов. SS низко означает, что раб - активный раб.

Так как это - последовательный протокол, и мы получим данные в байтах, мы должны быть осведомлены о том, прибудут ли данные LSB сначала или MSB сначала. Будет выбор управлять этим.

Для часов мы будем запирать данные, и мы должны будем знать, какие края и параметры настройки важны. Будет выбор способа часов управлять этим. В SPI есть два признака, названные фазой и полярностью. Фаза `(CPHA)` - запираем ли мы данные по высокому или низкому, и Полярность `(CPOL)` или высока, или низкий означает, что часы неработающие.

`CPOL=0` означает, что часы - дефолт низко, `CPOL=1`

означает, что часы - дефолт высоко. Когда `CPOL=0`, тогда

следующее - ценности для CPHA

`CPHA=0` означает, что данные собраны на часах возрастающий край,
`CPHA=1` означает, что данные собраны на часах, падающих край.

Когда `CPOL=1`, тогда следующее - ценности для CPHA

`CPHA=0` означает, что данные собраны на часах, падающих край, `CPHA=1` означает, что данные собраны на часах возрастающий край.

SPI обертывает эти два флага в четыре определенных и названных способа:

Способ	**Полярность часов – CPOL**	**Фаза часов – CPHA**
`SPI_MODE0`	0 (Дефолт часов низко)	0
`SPI_MODE1`	0 (Дефолт часов низко)	1
`SPI_MODE2`	1 (Дефолт часов высоко)	0
`SPI_MODE3`	1 (Дефолт часов высоко)	1

Также для часов, что скорость, мы должны будем знать, какая скорость данные должны быть перемещены. Будет выбор скорости контроля за часами управлять этим.

У ESP8266 есть аппаратные средства поддержка SPI, которой управляют, устанавливая регистры, и использование SDK поставляло макрос. Есть определенные физические булавки, которые зарезервированы для аппаратных средств SPI. Это:

GPIO	NodeMCU	Имя:	Функция
GPIO12	D6	HMISO	МИСО
GPIO13	D7	HMOSI	MOSI
GPIO14	D5	HSCLK	CLK
GPIO15	D8	HCS	CS

Для начала мы будем думать о часах. Это - скорость, на которой ESP8266 переключает коммуникационный поток с партнером SPI. Мы можем установить это быть той же скоростью как часы процессора (80 МГц), в письме к периферийному регистру.

```
WRITE_PERI_REG (SPI_CLOCK (HSPI), SPI_CLK_EQU_SYSCLK)
```

Это инструктирует ESP8266, что аппаратные средства, тактовая частота SPI должна быть равна системной тактовой частоте, которая обычно является 80 МГц.

Если эта скорость слишком быстра, мы можем изменить тактовую частоту с делителем под названием `SPI_CLKDIV_PRE`. Это делит тактовую частоту на стоимость. Если Вы хотите разделиться на X, тогда поставляют ценность X-1.

Например, чтобы разделить тактовую частоту на 10, предоставьте ценность 9.

Теперь у нас есть сырая тактовая частота. Однако есть больше. Есть урегулирование под названием `SPI_CLKCNT_N`, который определяет, сколько из этих сырых клещей часов должно составить один такт SPI. Помните, что такт SPI начинается высоко, переходы к низкому и затем возвращается к высокому снова. `SPI_CLKCNT_N` определяет, сколько сырых тактов соответствует такту SPI. Например, определение ценности 9 (требуемое значение - `n+1`) означает, что 10 сырых тактов будут одним тактом SPI. Поймите, что это - делитель сырого такта.

Есть другие параметры настройки, что часы эффекта и их называют `SPI_CLKCNT_H` и `SPI_CLK_CNT_L`. Взятый вместе, они определяют количество тактов, против которых часы высоки низко. Это формирует сигнал часов. Ценности здесь закодированы, чтобы означать количество сырых тактов, где часы SPI высоки против низко. Различие между ними дает результат. Например, урегулирование `SPI_CLKCNT_H=6` и `SPI_CLKCNT_L=1` приводит к различию 5 подразумевать, что 5 сырых тактов будут высоки, и остающиеся такты (5) будут низкими.

Функция	Биты	Маска	Изменение
SPI_CLK_EQU_SYSCLK	31	N/A	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

Есть три регистра, которые управляют вводом данных и производят. Их называют `SPI_USER`, `SPI_USER1` и `SPI_USER2`.

`SPI_USER` содержит следующие флаги:

- `SPI_CS_SETUP` – Когда он позволен, линия сигнала выбора кристалла, тянут низко несколько циклов перед передачей, разрешающей партнеру устройство SPI стать готовым к передаче.
- `SPI_CS_HOLD` – Когда он позволен, линия сигнала выбора кристалла, остается низким для нескольких циклов после передачи, разрешающей партнеру устройство SPI продолжиться для немного.

Функция	Бит
SPI_USR_COMMAND	31
SPI_USR_ADDR	30
SPI_USR_DUMMY	29
SPI_USR_MISO	28
SPI_USR_MOSI	27
SPI_USR_MOSI_HIGHPART	25
SPI_USR_MISO_HIGHPART	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

SPI_USER1 содержит четыре параметров настройки для числа битов в различных параметрах настройки SPI. Это:

- `SPI_USR_ADDR_BITLEN` – Сколько битов в адресе SPI.
- `SPI_USR_MOSI_BITLEN` – Сколько битов в данных MOSI.
- `SPI_USR_MISO_BITLEN` – Сколько битов в данных о МИСО.
- `SPI_USR_DUMMY_CYCLELEN` – Сколько битов в фиктивных циклах.

Дополнительный регистр под названием `SPI_ADDR` содержит адрес.

Функция	Биты	Маска	Изменение
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

Функция	Биты	Маска	Изменение
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

Детали этих регистров и констант определения макроса могут быть найдены в `spi_register.h` файле, поставляемом в водителях, включают справочник.

См. также:

- Аппаратные средства регистры часов SPI
- Аппаратные средства SPI (HSPI) команда & регистры данных
- Википедия –последовательный периферийный интерфейсный автобус

MetalPhreak/ESP8266_SPI_Driver

Взаимодействуя с аппаратными средствами SPI может быть сложным. К счастью, общедоступный проект на GitHub предоставил простое в использовании решение проблемы. Проект известен как `MetalPhreak/ESP8266_SPI_Driver`, который состоит из исходного файла C и нескольких заголовочных файлов.

Источник выставляет следующие функции:

- пустота spi_init (uint8 spi_no)
- пустота spi_init_gpio (uint8 spi_no, uint8 sysclk_as_spiclk)
- пустота spi_clock (uint8 spi_no,
  uin
  t16
  pre
  div,
  uin
  t8
  cnt
  div
)

Страница 143

Чтобы вычислить эффективную тактовую частоту, возьмите частоту центрального процессора (80 МГц) и затем разделите ее на preDiv. Это дает основную частоту. Затем мы делим это на

- пустота spi_tx_byte_order (uint8 spi_no, uint8 byte_order)
- пустота 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_bi
  ts)

Кроме того, следующий макрос также обеспечены:

- spi_busy (spi_no)
- spi_txd (spi_no, биты, данные)
- spi_tx8 (spi_no, данные)
- spi_tx16 (spi_no, данные)
- spi_tx32 (spi_no, данные)
- spi_rxd (spi_no, биты)
- spi_rx8 (spi_no)
- spi_rx16 (spi_no)
- spi_rx32 (spi_no)

В предыдущих функциях `SPI` и `HSPI` - допустимые ценности для `spi_no`.См. также:

- GitHub:MetalPhreak/ESP8266_SPI_Driver

Работа с сериалом ESP8266

Есть два UARTs в системе, известной как UART0 и UART1. У UART0 есть свой собственный посвященный TX и булавки RX, в то время как UART1 мультиплексирован с GPIO2. UART1 произведен только, и следовательно только имеет линию TX.

Последовательный интерфейс к ESP8266 может быть инициализирован с требованием к функции `uart_init ()`.

Например,

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

Чтобы написать последовательность последовательному порту, мы можем тогда использовать `os_printf ()`. Это имеет тот же формат как `printf`, но пишет последовательному порту.

Чтобы работать с UART, Вы должны включать `uart.c`, `uart.h` и `uart_register.h` файлы от `examples/driver_lib`. В Вашем заявлении Вы должны тогда включать `«driver/uart.h»`.

Чтобы передать данные, используя UART0, у нас есть функция, `вызванная uart0_tx_buffer ()`, который принимает указатель на

данные и длину и передает их.

В SDK есть буфер передачи. Поскольку передача UART, как правило - медленная операция, заявлениям, которые хотят передать данные, сохранили их данные в буфере TX, который тогда истощается передачей со временем. Данные, написанные UART, гарантируют, чтобы быть написанными в заказе, в котором это поставлялось. Если буфер TX становится полным, никакие новые данные не могут быть приняты.

Точно так же данные, полученные ESP8266 по UART, помещены в получить буфер. Применение, работающее на ESP8266, должно получить данные своевременно. Если буфер RX становится полным нет никакого места, чтобы поместить новые данные и что от новых данных откажутся. В терминах UART и TX и буфера RX называют «FIFO», что означает метод «первым пришел - первым вышел». Буфера составляют 128 байтов каждый (128 байтов для TX и второй 128-байтовый буфер для RX).

Чтобы узнать, сколько байтов находится на различных очередях, мы должны пойти довольно низкий уровень. Например, числом байтов на очереди TX дают:

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

в то время как числом байтов на очереди RX дают:

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

См. также:

- Соединение с ESP8266
- USB к конвертерам UART
- Ошибка: Справочный источник, не найденный
- Ошибка: Справочный источник, не найденный
- Ошибка: Справочный источник, не найденный
- os_printf

Обработка Задачи ESP8266

Предположите, что мы хотим выполнить задачу для нас асинхронно. То, что мы могли бы хотеть сделать, отправить это, мы хотим, чтобы это

произошло и затем продолжало наш бизнес. Когда мы сделаны и оставили контроль назад OS, мы предполагаем, что задача в конечном счете начнет выполнять. Это - функция, обеспеченная функциями задачи ESP8266. Есть две функции интереса для нас. Первое называют `system_os_task ()`, который настраивает процессор задачи.

Когда мы хотим отправить это, задача имеет право начаться, мы можем использовать вторую функцию, `вызванную system_os_post ()`, который размещает сообщение.

Функция задачи, которую мы зарегистрировали, будет тогда «призвана» в какой-то момент после почтового запроса и будет дана параметры, поставляемые на почте. Приоритет определяет относительный приоритет двух постов, которые были выпущены. Тот с самым высоким приоритетом выполнит сначала.

Важно отметить, что **только три** приоритета позволены, которые являются 0, 1 и 2 с 0 наличием самого низкого приоритета. Также важно отметить, что может только быть **один** укладчик для каждой регистрации задачи приоритетом. Таким образом, если мы выполняем `system_os_task ()` дважды использование того же приоритета в обоих случаях, только последний помнят и выполнят, когда задача того приоритета отправлена.

С этим фоном, какова цель этого набора функции? Почему мы заботились бы об этом?

Примечание: Библиотека Ардуино для ESP

использует приоритет 0 наборов задачи. Вот

образец простого укладчика задачи.

```
пустота taskHandler
       (os_event_t
       *событие)
       {выключатель
       (событие-> сигнал) {
       слу
              ч
              а
              й
              1
              :
              р
              а
              з
              р
              ы
              в
              ;
       слу
              ч
              а
              й
              2
```

```
                :
                p
                a
                з
                p
                ы
                в
                ;
        }
}
```

```
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) «Привет»);
```

См. также:

- Ошибка: Справочный источник, не найденный
- Ошибка: Справочный источник, не найденный

Страница 146

Таймеры и время

В рамках нашего кодекса мы можем хотеть задержаться сроком на время. Мы можем использовать `os_delay_us ()` функция, чтобы приостановить обработку за установленный срок, измеряемый в микросекундах. Есть 1 000 микросекунд в миллисекунде и 1000 миллисекунды за секунду.

Мы можем настроить таймер, который назовут на периодической основе со степенью детализации отзыва миллисекунд. Структура данных, названная `os_timer_t`, держит государство таймера.

Мы можем определить пользовательскую функцию, которую назовут, когда огни таймера, используя `os_timer_setfn ()` функционируют. Обратите внимание, что мы можем только установить функцию обратного вызова, когда таймер разоружен.

Когда готовый, мы можем вооружить таймер так, чтобы он начал тикать и стрелял, когда готовый. Мы делаем это использование `os_timer_arm ()` функция.

Повторный флаг указывает, должен ли таймер перезапустить после того, как он стрелял. Мы можем приостановить или отменить увольнение таймера,

используя `os_timer_disarm ()`. Вот пример:

```
os_timer_t myTimer;

пустота timerCallback
       (пустота *pArg)
       {os_printf
       («Тиканье!»);
}//Конец timerCallback

пустота user_init (пустота) {
       uart_init (BIT_RATE_115200,
       BIT_RATE_115200); os_timer_setfn
       (&myTimer, timerCallback, ПУСТОЙ
       УКАЗАТЕЛЬ); os_timer_arm (&myTimer,
       1000, 1);
}//Конец user_init
```

Другой аспект работы со временем - вычисления времени и измерение. Функция `system_get_time ()` возвращает 32-битное неподписанное целое число (unit32) стоимость, которая является микросекундами, так как устройство загрузило. Эта стоимость перевернется после 71 минуты.

Что, если нам нужна степень детализации времени, меньшего, чем микросекунда? Надо надеяться, Вам часто не будет нужно это … однако, некоторые решения, такие как работа со светодиодами WS2812 действительно на самом деле требуют сверхчеткой точности.

Одно возможное решение состоит в том, чтобы опуститься к программированию ассемблера. Есть специальный регистр, управляемый ESP8266, который называют «`ccount`», который измеряет циклы операции. Ценность этого увеличена каждый раз, когда эксплуатационный цикл заканчивает.

Ценность этого регистра может быть восстановлена со следующим кодексом C:


```
статический действующий
  uint32_t getCycleCount ()
  {uint32_t ccount;
  __ asm __ __ изменчивый __ («rsr %0,
  количество»: «=a» (количество));
  возвратите количество;
}
```

Этот фрагмент использует действующий ассемблер, чтобы преобразовать заявление ассемблера в его соответствующую эксплуатационную инструкцию.

См. также:

- Ошибка: Справочный источник, не найденный
- os_timer_arm
- os_timer_disarm
- os_timer_setfn

Работа с памятью

Работая в C, Вы должны думать с точки зрения памяти компьютера. Сумма доступной RAM, вероятно, составит меньше чем 45 кбайт.

Мы можем ассигновать память, используя `os_malloc ()` или `os_zalloc ()`. Первая функция ассигнует и возвращает память, и второе делает точно то же, но ноли память перед возвращением. Когда Вашей логике больше не нужна память, она может возвратить его назад к куче с `os_free ()`. Чтобы определить, сколько размера кучи доступно, мы можем назвать `system_get_free_heap_size ()`. Как только у нас есть указатель памяти, мы можем начать управлять им через серию команд памяти. `os_memset ()` команда установит блок памяти определенной стоимости. `os_memcpy ()` скопирует блок памяти другому блоку. `os_bzero ()` функция установит значения блока памяти нолю.

Память на ESP8266 составлена из многих компонентов. Мы имеем:

- данные
- rodata
- bss
- куча

Ценности их могут быть найдены через `system_print_meminfo ()` функцию.

Когда ESP8266 должен прочитать инструкцию по памяти, чтобы выполнить его, та инструкция может прибыть из одного из двух мест. Инструкция может быть во флэш-памяти (также названный `irom`), или это может быть в RAM (также названо `iram`). Требуется меньше времени для процессора, чтобы восстановить инструкцию от RAM, чем это делает от вспышки. Считается, что усилие инструкции от вспышки берет в четыре раза дольше, чем та же инструкция

`ICACHE_FLASH_ATTR`
принесенный от RAM. Однако на ESP8266 есть намного меньше RAM, чем есть вспышка. То, что это означает, - то, что у Вас, намного более вероятно, закончится RAM путь, прежде чем у Вас закончится вспышка. Сочиняя нормальные заявления, мы не должны фиксировать при наличии инструкций в RAM, а не вспышке для исполнительной выгоды. Скорости выполнения ESP8266 так быстры, что, если стоимость восстановления инструкции от RAM ослепляюще быстро тогда восстанавливает, инструкция от более медленной вспышки **все еще** ослепляюще быстра.

Есть однако определенные классы инструкций, что мы могли бы хотеть поместить в RAM, а не вспышке. Примеры их - укладчики перерыва, где время, проведенное в них, должно всегда быть максимально коротким и также функционировать, которые пишут, чтобы вспыхнуть.

Когда мы определяем функции C, мы можем добавить признак под названием `ICACHE_FLASH_ATTR`. То, что это делает, поместить эту функцию в адрес флэш-памяти
пространство в противоположность RAM. А
именно, ослабевая функция с признаками это как
являющийся в «`.irom0.text`» разделе кодекса.

Примечание: С сырой технической точки зрения `ICACHE_FLASH_ATTR` - `#define`, который наносит на карту к:

```
__ припишите __ ((раздел (» .irom0.text»)))
```

Одна из областей, которые мы еще не обсудили, - то, как память населяется и используется, когда ESP8266 загружает. Есть два файла, обычно загружаемые во вспышку. Первое в погашении `0x00000`, и это содержит данные, которые будут загружены в RAM основанным на ROM загрузчиком операционной системы. Эти данные содержат серию разделов и адресов в RAM того, где данные будут загружены. Второй бинарный файл обычно загружается во вспышку в `0x40000`. Это содержит двоичные данные раздела, названного `.irom0.text`, который содержит кодекс. RAM, которой должен соответствовать нагруженный кодекс, где эта вспышка хранила данные, обращена.

Для заинтересованных деталями низкого уровня, был расшифрован формат памяти, написанной файлам вспышки. Считается, что низкая

память похожа:

```
структура
       rom_head
       er
       {uint8
       волшебст
       во;
       uint8
       sect_cou
       nt;
       uint8
       flags1;
       uint8
       flags2;
       uint32
       entry_ad
       dr;
};
```

Волшебная собственность - константа `0xe9`. `sect_count` содержит количество разделов, чтобы загрузить в поршень. Это не будет включать `irom.text` раздел. `flags1` и `flags2` используются, чтобы указать на размер вспышки, тактовую частоту и способ IO. Наконец, `entry_addr` - точка входа, чтобы начать выполнять поставляемый код пользователя.

Сразу после заголовка, записи раздела (должен быть `sect_count` их), где каждый вход:


```
структура
       sect_h
       eader
       {uint3
       2
       адрес;
       длина
       uint32
       ;
}
```

Адреса должны быть в `.iram` адресном пространстве, начинающемся в `0x40100000`. После того, как каждый из заголовков - стоимость контрольной суммы. Контрольная сумма вычислена от каждого из разделов и утвердила это, это соответствует тому, что, как предполагается, присутствует.

Вся вспышка также нанесена на карту к адресному пространству `0x40200000`.

Второй файл содержит `irom.text` данные о разделе. По умолчанию

адресное пространство для этого раздела - `0x40240000`, что означает, что это должно быть написано, чтобы вспыхнуть в `0x40000`. Превосходная карта памяти ESP8266 сохраняется наESP8266 Wiki. Основная сущность его здесь:

Адрес	Раз мер	Примечания
0x0000 0000 <		Не может быть прочитан.
0x2000 0000 <		Ненанесенный на карту
0x3FF0 0000 <		Память нанесла на карту ввод/вывод.
0x3FF1 0000 <		Ненанесенный на карту.
0x3FFE 8000 <	80K – 81920 (0x14000)	Пользовательская RAM данных (глоток)
0x3FFF C000 <	16K – 16384 (0x4000)	Системная RAM данных ETS.
0x4000 0000 <		Внутренний ROM
0x4010 0000 <	32K – 32768 (0x8000)	RAM инструкции (iram/sram)
0x4010 8000 <		Ненанесенный на карту или неизвестный
0x4020 0000 <	Макс 1024K (1M) – 1048576 (0x100000)	Вспышка SPI
0x4030 0000 <		Ненанесенный на карту или неизвестный

Теперь прибывает обсуждение, которое взяло меня долгое, долгое время, чтобы постигать. Это - отношения между флэш-памятью и адресным пространством ESP8266. Если мы исследуем предыдущую таблицу, мы, кажется, видим, что Флэш-память нанесена на карту в адресное пространство ESP8266 в `0x4020 0000` адреса. Это ключевое и жизненно важное, чтобы понять. Если мы читаем от этого адреса вперед, мы на самом деле читаем от флэш-памяти.

Давайте думать о ESP-12, у которого есть 512K флэш-памяти. В ведьме это - `0x8` 0000 байтов. Это означает диапазон адресов `0x4020 0000` к `0x4027 FFFF`. Теперь давайте рассмотрим инструменты высвечивания ESP, такие как «`esptool`». Это также принимает адрес, в который можно загрузить вспышку …, но как тот адрес касается адресного пространства во время выполнения

ESP8266? Ответ - то, что, если мы пишем, чтобы обратиться `к 0x0 0000` вспышки во времени выполнения, это появится в `0x4020 0000`. Таким образом, в действительности то, что мы имеем, следующее:

Теперь мы отвечаем на еще одну загадку. Когда мы соединяем файлы объекта и создаем бинарное изображение …, часть роли компоновщика должна придумать заключительное расположение адреса, где исполняемый файл будет наконец проживать. На практике, когда мы управляем компоновщиком, мы поставляем файл контроля компоновщика, названный «`орлом app.v6.ld`». Это поставляется Espressif. Если мы смотрим в файле *по умолчанию*, мы находим следующее в начале файла:

```
ПАМЯТЬ
{

  dport0_0_seg: dram0_0_seg: iram1_0_seg: irom0_0_seg:
}
org = 0x3FF00000, len = 0x10 org = 0x3FFE8000, len = 0x14000 org =
0x40100000, len = 0x8000 org = 0x40240000, len = 0x32000
```

Теперь … смотрят на это тесно …, потому что это взяло меня навсегда, чтобы понять то, что я собираюсь сказать Вам. Посмотрите на адрес в том, где `irom0` сегмент начинается. Это начинается в 0x4024 0000 …, это - 256K **в** 512K диапазон адресов! Помещая это иначе, наши исполняемые файлы не могут использовать целый адрес с этими настройками по умолчанию. Почему Espressif устанавливал это? Ответ, как полагают, - потому что они хотят обеспечить способность

модернизации Over The Air (OTA) и хотеть позволить Вам иметь бегущую копию своего приложения **и** новую копию в полете быть загруженными. Это означает что эффективно, 256K для текущей версии и 256K для новой загружаемой версии. Если мы пытались использовать целое адресное пространство, и во время освежительного напитка что-то пошло не так, как надо, у нас нет ничего, чтобы отступить к.

Если Вы, как я, требуемый, чтобы использовать целое адресное пространство, ответ просты. Измените соответствующий вход в `«орлином app.v6.ld»` файле.

Одна последняя морщина. Считается, что последний 16K вспышки должен быть зарезервирован для хранения Espressif SDK для вещей как последний используемый SSID и пароль. Не предполагайте, что Вы можете использовать тот диапазон.

По умолчанию следующие пункты в файле ЭЛЬФА идут в местоположение вспышки 0x0 0000

- .text
- .data
- .rodata
- .iram0.text

Следующие разделы идут в местоположение вспышки 0x4 0000

- .text Видят также:

- os_memset
- os_memcpy
- os_memcmp
- os_malloc
- os_zalloc
- os_free
- Ошибка: Справочный источник, не найденный
- Ошибка: Справочный источник, не найденный
- Процесс загрузки ESP8266
- esptool.py
- esptool-ck
- gen_appbin.py

Работа с флэш-памятью

Флэш-память обеспечивает энергонезависимое хранилище

информации, которая переживает цикл власти устройства.

Данные, содержавшие во вспышке, хранятся в единицах секторов, которые составляют 4 096 байтов в размере. Чтобы написать данные, мы можем назвать `spi_flash_write`. Чтобы прочитать данные, мы называем `spi_flash_read`.

Начиная с письма вспышке выполняется в единицах 4 096 байтов, мы не можем изменить единственный байт, просто переписав его, вместо этого мы должны восстановить целый сектор, стереть сектор и затем написать сектор в ответ с измененным содержанием. Это может занять время, чтобы закончить и из-за этого, мы можем найти, что неудача, более вероятно, произойдет (например, потеря власти). Если неудача происходит после того, как мы стерли сектор или во время переписывания сектора, должно немедленно стать очевидно, что мы приведем к полной коррупции данных.

Страница 152

Данные читают и пишут, должны быть 4 байта, выровненные во вспышке.

ESP8266 должен быть проинструктирован о размере флэш-памяти, доступной ему. Попытка использовать адреса флэш-памяти, которые отличаются от ожидаемого размера доступной флэш-памяти, может привести к неожиданным результатам.

Когда использование `esptool.py`, `- flash_size` флаг может поставляться. Для esptool-ck соответствующий флаг-`bz`.

См. также:

- spi_flash_get_id
- spi_flash_erase_sector
- spi_flash_read
- spi_flash_set_read_func
- system_param_save_with_protect
- Cesanta:ESP8266, вспышка и выравнивание
- system_param_load

Модуляция ширины импульса - PWM

Идея позади модуляции ширины импульса состоит в том, что мы можем думать об обычных импульсах выходных сигналов как кодирование информации в том, сколько времени сигнал поддержан на высоком уровне. Давайте предположим, что у нас есть период 1 Гц (одна вещь в секунду). Теперь давайте предположим, что мы повышаем выходное напряжение до уровня 1 в течение ½ из секунды в начале периода. Это

дало бы нам прямоугольную волну, которая начинается высоко, длится 500 миллисекунд и затем понижается низко для следующих 500 миллисекунд. Это повторяется на в будущее. Продолжительность, что пульс высок относительно периода, позволяет нам кодировать аналоговую стоимость на цифровые сигналы. Если бы пульс 100% высотой в течение периода тогда, закодированная стоимость была бы 1.0. Если бы пульс составляет 100% низко в течение периода, то закодированная стоимость была бы 0.0. Если пульс идет для «n» миллисекунд (где n - меньше чем 1 000), то закодированная стоимость была бы n/1000.

Как правило, длина периода не секунда, но очень, намного меньшее разрешение нам произвести много отличающихся ценностей очень быстро. Отношение на сигнале к периоду называют «рабочим циклом». Этот метод кодирования называют «Модуляцией Ширины импульса» или «PWM».

Есть множество целей для PWM. Некоторые - кодирующие устройства выходных данных. Одна обычно замечаемая цель состоит в том, чтобы контролировать яркость светодиода. Если мы применяем максимальное напряжение к светодиоду, это максимально ярко. Если мы обращаемся ½ напряжение, это - приблизительно ½ яркость. Применяя быстрый период PWM сигнализируют к входу светодиода, рабочий цикл становится яркостью светодиода. Путем это работает, то, что или полное напряжение или никакое напряжение применены к светодиоду, но потому что период так короток, «среднее» напряжение со временем следует за рабочим циклом и даже при том, что светодиод мерцает на или прочь, это настолько быстро, что наши глаза не могут обнаружить его и все, что мы видим, очевидное изменение яркости.

Страница 153

Для ESP8266 период PWM может колебаться от 1 миллисекунды до 10 миллисекунд. Это - частота 1 кГц к 100 Гц. Разрешение рабочего цикла составляет до 45 наносекунд, который составляет 14 битов данных о резолюции. Устройство оказывает поддержку максимум для 8 каналов PWM, где каждый канал может быть связан со своей собственной булавкой и рабочим циклом. Период - то же для всех каналов PWM.

Чтобы начать использовать поддержку ESP8266 PWM, требование к `pwm_init ()` необходимо, который настраивает, какие булавки должны использоваться для PWM и для который каналы. Требование к этой функции также настраивает начальный период и рабочий цикл. Звонок к

`pwm_start ()` может тогда быть сделан, чтобы начать продукцию PWM. Период PWM в целом и рабочих циклов для каждого канала может быть изменен, используя `pwm_set_period ()` и `pwm_set_duty ()` функции. Функции ESP8266 PWM используют таймер аппаратных средств. Как таковой Вы можете иметь поддержку PWM или использовать таймер аппаратных средств для Вашего использования …, но не оба.

Чтобы использовать функции ESP8266 SDK PWM, Вы должны связать свое заявление с `libpwm.a`.

См. также:

- Википедия:модуляция длительности импульса
- Образец Espressif
- pwm_init
- pwm_startpwm_set_duty
- pwm_get_duty
- pwm_set_period
- pwm_get_period

Аналог к цифровому преобразованию

Аналог к цифровому преобразованию - способность прочитать уровень напряжения от булавки между 0 и некоторое максимальное значение и новообращенный что аналоговое напряжение в цифровое представление. Изменение напряжения относилось к булавке, изменит прочитанную стоимость. ESP8266 встроили аналого-цифровой преобразователь в него с разрешением 1 024 отличных ценностей. То, что это означает, - то, что 0 В произведут цифровую ценность 0, в то время как максимальное напряжение произведет цифровую ценность 1 023, и диапазоны напряжения между ними произведут соответственно чешуйчатую цифровую стоимость.

Чтобы прочитать цифровое значение аналогового напряжения, функция, названная `system_adc_read ()`, должна быть вызвана. Булавку на физическом ESP8266, из которого прочитано напряжение, называют `СПЕКУЛЯНТОМ` и не служит никакой другой цели.

Входной диапазон на булавке от 0V до 1 В. Это подразумевает, что входное напряжение к ADC не может быть максимальным напряжением, используемым, чтобы привести сам ESP8266 в действие (3.3 В). Таким образом, мы должны будем использовать схему сепаратора напряжения.

Формула, чтобы планировать их:

$$Vout = \frac{R2}{R1+R2} \cdot Vin$$

Так как мы знаем, что Vout идет в 1 В, и Вин - 3 В, и мы выбираем R2, чтобы быть 10K, мы находим:

$$R1 = \frac{R_{Vout-R2}}{2 \cdot Вин_и}$$

для наших

ценностей:

$$R1 = \frac{10000}{1.0} - 10000 * 3.3 = 23000$$

Общий 22K резистор будет работать хорошо.

Вот пример. То, что делает этот пример, напечатать стоимость, прочитанную из ADC каждую секунду.

```
os_timer_t myTimer;

пустота timerCallback (пустота *pArg) {
       uint16 adcValue =
       system_adc_read ();
       os_printf («adc = %d\n»,
       adcValue);
}//Конец timerCallback

пустота user_init (пустота) {
       uart_init (BIT_RATE_115200,
       BIT_RATE_115200); os_timer_setfn
       (&myTimer, timerCallback, ПУСТОЙ
       УКАЗАТЕЛЬ); os_timer_arm (&myTimer,
       1000, 1);
}//Конец user_init
```

Если мы пристраиваем на макете схему, которая включает легкий зависимый резистор, такой как следующее:

Тогда, когда мы изменяем количество света, падающего на резистор, мы видим, что ценности изменяются, поскольку данные написаны в регистрации продукции. Это может использоваться, чтобы вызвать действие (например), когда это становится темным.

Нерешенный вопрос: Какова частота дискретизации ADC?См. также:

- Ошибка: Справочный источник, не найденный
- Википедия:сепаратор Напряжения

Режимы ожидания

Если устройство ESP8266 постоянно включено, то оно постоянно потребляет ток. Если источник питания неограничен, то это не должно обязательно быть проблемой однако, работая от батарей или другой конечной поставки, мы, возможно, должны минимизировать потребление. Один способ достигнуть этого состоит в том, чтобы приостановить эксплуатацию устройства если не в использовании. Когда устройство приостановлено, понятие - то, что потребление будет уменьшено. Есть три определенных режима ожидания. Их называют сном модема, легким сном, глубоким сном.

Смотря на следующую таблицу мы можем получить чувство способностей в каждом из этих трех способов:

Функция	Модем	Свет	Глубоко
WiFi	прочь	прочь	прочь
Системные часы	на	прочь	прочь
Оперативные часы	на	на	на
Центральный процессор	на	ожидание	прочь
Текущее потребление	15mA	0.5mA	20µA

Сон модема может только использоваться, когда ESP8266 находится в станционном способе, связанном с точкой доступа. Применение этого способа состоит в том, когда ESP8266 должен все еще выполнить работу, но минимизирует сумму беспроводных передач.

Легкий режим ожидания совпадает со сном модема, но в этом случае часы будут приостановлены.

В способе глубокого сна устройство действительно спит. Ни центральный процессор, ни деятельность WiFi не происходят. Устройство что бы там ни было от … за одним исключением …, это может проснуться в указанном регулярном интервале.

Чтобы перейти к режиму глубокого сна, мы можем назвать `system_deep_sleep ()`. Это может поставляться временем приостановки. Устройство заснет и после того, как интервал протек, устройство проснется снова. В дополнение к наличию таймера мы можем также проснуться от глубокого сна, переключив ценность сигнала на булавке.

Мы можем управлять, в каком способе устройство находится, звоня `wifi_set_sleep_type ()`.

Охранительный таймер

ESP8266 - единственный переплетенный процессор. Это означает, что может только сделать одну вещь за один раз, поскольку нет никаких параллельных нитей, которые могут быть выполнены друг одновременно с другом. Значение этого - то, что, когда OS дает контроль к Вашему заявлению, это не возвращает контроль до тех пор, пока Вы явно оставляете его. Однако это может вызвать проблемы. ESP8266 - прежде всего, WiFi и устройство TCP/IP, которое ожидает быть в состоянии получить и передать данные, а также ответить на асинхронные события в своевременном способе. Как пример, если Ваше устройство ESP8266 связано с точкой доступа и точкой доступа, хочет утвердить это, Вы все еще связаны, это может передать пакет Вам и ожидать ответ. Вы не имеете никакого контроля, когда это произойдет. Если Ваше собственное приложение будет управлять выполнением в то время, когда запрос прибывает, тот запрос не ответят на то, пока Вы не возвратите контроль назад к OS. Между тем точка доступа может ожидать ответ в течение некоторого предопределенного периода времени и, если не получит ответ в том интервале, может

предположите, что Вы разъединили. Чтобы предотвратить такие обстоятельства, Ваш программный код должен возвратить контроль назад к OS своевременно. Рекомендуется, чтобы Ваш кодекс возвратил контроль в 50 миллисекундах получения контроля. Если Вы занимаете больше времени, Вы рискуете запросами к своему устройству, рассчитывающему.

Если Ваш собственный кодекс не возвращает контроль назад к OS, OS должен предположить, что вещи идут не так, как надо. По сути, у этого есть таймер, что мы называем «контрольную комиссию». Когда контроль дан к Вашему собственному кодексу, охранительный таймер начинает тикать. Если Вы не возвратили контроль назад к OS к тому времени, когда охранительный таймер достигает ноля, он берет дело в свои руки. Явно то, что это делает, является перезагрузкой устройство. Это может походить на довольно радикальные меры, но взгляды состоят в том, что лучше сделать это и надеяться, что то независимо от того, что было заблокировано, теперь открыто, чем, просто сидят там «мертвые».

Отчеты утверждают, что охранительный таймер может составить приблизительно 1 секунду (1 000 миллисекунд). Однако в моих тестах, я нахожу, что таймер стреляет приблизительно в 3,2 секунды (3 200 национальной безопасности).

Функция, `вызванная system_soft_wdt_stop ()`, останавливает охранительный таймер … или по крайней мере один из них. Кажется, есть **два** таймера. Каждый находится в программном обеспечении, другом в аппаратных средствах. Эта функция останавливает таймер программного обеспечения. Это может быть перезапущено с `system_soft_wdt_restart ()`
… однако, второй таймер звонил, охранительный таймер аппаратных средств будет стрелять приблизительно после 8 секунд и не кажется способным быть пойманным в ловушку. Новое требование, введенное в SDK 1.3, называют `system_soft_wdt_feed ()`. К сожалению, документация относительно этого чрезвычайно плоха. Лучшие отчеты, которые мы имеем до сих пор на том, для чего это делает, состоят в том, что, когда мы называем его, охранительное время перезагружено к его отправной точке и начинает тикать вниз снова. Я не совсем уверен в ценности этого, учитывая, что у нас уже есть API, чтобы остановить и затем перезапустить таймер. Надо надеяться, в будущем мы можем получить дополнительное знание, чтобы убрать любые тайны, которые могут скрываться в.

См. также:

- system_soft_wdt_stop
- system_soft_wdt_restart
- Ошибка: Справочный источник, не найденный

Получение контроля

Мы просто описывали понятие необходимости возвратить контроль назад к OS для него, чтобы выполнить его обязанности хранения дома. Путем мы делаем это должно просто возвратиться из отзыва, на который OS призвал нас. Если мы будем думать о том, как программа ESP8266 работает, мы будем видеть, что для нас, чтобы оставить контроль назад OS, OS, должно быть, позвонил нам во-первых. Поэтому имеет смысл для нас возвращать контроль позже. Однако, если мы возвращаем контроль, мы (очевидно), освобождаем все государство (переменные), которые были существующими, когда мы возвратились.

Теперь мы добираемся, чтобы ввести названное «получение» понятия. Идея позади получения состоит в том, что вместо нашего заявления, возвращая контроль назад к OS, что мы можем договориться сделать, возвратиться к OS, одновременно поддерживая контекст того, где мы в рамках текущего выполнения. То, когда OS заканчивает раунд домашнего хозяйства, что он может тогда сделать, «возвратиться» назад туда, где когда-либо мы были, когда мы просили урожай произойти.

Это - хитрый материал, чтобы осуществить, но к счастью Иван Горхотов достиг этой задачи для нас, и мы можем усилить то, что он уже построил.

Использовать это:

1. Включайте cont.h
2. Создайте глобальный из «`cont_t g_cont __ признак __ ((выровнял (16)))`»;
3. В user_init, названном «`cont_init (&g_cont)`»;
4. Зарегистрируйте `system_os_task ()` процессор.

Когда мы будем хотеть наметить некоторый кодекс для выполнения, отправьте задачу.

```
пустота esp_schedule ()
    {system_os_post
    (TASK_PRIORITY, 0, 0);
}

статическая пустота taskHandler
    (os_event_t *события)
    {cont_run (&g_cont,
    someFunction); если
    (cont_check (&g_cont)! = 0) {
        os_printf («Переполняют
        detected\r\n»); аварийное
        прекращение работы ();
    }
}
```

Безопасность

У ESP8266 есть способность сохранить пароль, используемый, чтобы соединиться с точкой доступа в памяти. Это означает, что, если нужно было физически поставить под угрозу устройство (т.е. украсть его) тогда они могли бы, в принципе, свалить флэш-память и восстановить Ваш пароль.

Вы не могли припрятать пароль про запас в ясном во вспышке, но вместо этого имели Ваши заявления, «расшифровывают»

закодированную версию, которая спасена во флэш-памяти …, это предотвратило бы очевидный поиск через простой захват памяти. Схема кодирования могла быть простым XOR против магического числа (или трудно закодированный или Ваш собственный MAC-адрес).

Отображение из Ардуино

Без аргумента Ардуино стал самой успешной программной окружающей средой микропроцессора до настоящего времени. Есть тонны и тонны существующих эскизов в

существование и позволило нам не забыть о богатстве библиотек. Инструменты и утилиты существуют, чтобы собрать и управлять эскизами Ардуино на ESP8266s. Что, если вместо этого мы хотели держать те эскизы Ардуино в строевой стойке к родному кодексу ESP8266? Мы можем найти отображения между API Ардуино и соответствующими API ESP8266?

Ардуино	**ESP8266**
digitalWrite (булавка, стоимость)	GPIO_OUTPUT_SET (булавка, стоимость)
digitalRead (булавка)	GPIO_INPUT_GET (булавка)
задержка (ms)	os_delay_us (ms * 1000) Примечание: ms <= 65535
delayMicroseconds (нас)	os_delay_us (нас)
millis ()	system_get_time () / 1000

С функциональной точки зрения вот некоторые сравнения между Ардуино и ESP8266:

	ESP8266	**Ардуино (Uno)**
GPIOs	17 (Меньше, как правило, подвергнутые)	14 (20 включая аналог)
Аналоговый вход	1	6
Каналы PWM	8	6
Тактовая частота	80 МГц	16 МГц
Процессор	Tensilica	Atmel
SRAM	45 кбайт	2 кбайта
Вспышка	512 КБ или более (отдельный)	32 КБ (на чипе)

Работа с напряжением	3.3 В	5 В
Ток Макса за ввод/вывод	12mA	40mA
UART (аппаратные средства)	1 ½	1
Организация сети	Встроенный	Отдельный
Документация	Бедный	Отлично
Зрелость	Рано	Зрелый

Примечание: Поскольку у Ардуино нет родной организации сети, никакие дальнейшие сравнения сетевой способности не были включены выше. Действительно помните, что, в это время, когда каждый использует ESP8266, возможности высоки, это - потому что Вам **нужен** сетевой доступ.

Файловая система магарычей при торговой сделке

Файловая система Вспышки SPI (МАГАРЫЧИ ПРИ ТОРГОВОЙ СДЕЛКЕ) является механизмом файловой системы, предназначенным для встроенных устройств. Внедрение поставляется FreeRTOS ESP8266 SDK.

Каков размер страницы вспышки ESP8266? Каков размер блока вспышки ESP8266?

Когда звонок API МАГАРЫЧЕЙ ПРИ ТОРГОВОЙ СДЕЛКЕ сделан, нулевой или положительный ответ указывает на успех, в то время как стоимость <0 указывает на ошибку. Природа ошибки может быть восстановлена через
`SPIFFS_errno ()` требование.

Внедрение МАГАРЫЧЕЙ ПРИ ТОРГОВОЙ СДЕЛКЕ непосредственно не получает доступ к флэш-памяти. Вместо этого функциональная область звонила, слой абстракции аппаратных средств («hal») предоставляет эту услугу. Интеграция МАГАРЫЧЕЙ ПРИ ТОРГОВОЙ СДЕЛКЕ требует, чтобы три функции были созданы, у которых есть следующие подписи:

```
s32_t (*spiffs_read) (u32_t addr, u32_t
размер, u8_t *dst) s32_t (*spiffs_write) (u32
_t addr, u32_t размер, u8_t *src) s32_t
```

```
(*spiffs_erase) (u32_t addr, u32_t размер)
```

Если они добиваются успеха, код возврата должен быть `SPIFFS_OK` (0).

На ESP8266 они нанесут на карту к API вспышки SPI.

Файловая система МАГАРЫЧЕЙ ПРИ ТОРГОВОЙ СДЕЛКЕ могла быть иерархической по своей природе таким образом, что она содержит и справочники и файлы, но кажется, что в действительности это не. Есть только один справочник, названный корнем. Корневой каталог «/». Чтобы определить членов справочника, мы можем открыть справочник для чтения с `SPIFFS_opendir ()` API и, когда мы закончены, закрываем операцию по чтению с `SPIFFS_closedir ()` требование API. Мы можем идти через статьи каталога с требованиями к SPIFFS_readdir ().

Например:

```
spiffs_DIR spiffsDir;
SPIFFS_opendir (&fs, «/»,
&spiffsDir); структура
spiffs_dirent spiffsDirEnt;
в то время как (SPIFFS_readdir (&spiffsDir,
  &spiffsDirEnt)! = 0) {printf («Получил статью
  каталога: %s\n», spiffsDirEnt.name);
}
SPIFFS_closedir (&spiffsDir);
```

Чтобы создать файл, мы можем использовать SPIFFS_open () API, поставляя флаг SPIFFS_CREAT.

См. также:

- GitHub:pellepl/spiffs

Партнер API TCP/IP

Если ESP8266 может действовать как один конец связи TCP/IP, что-то еще должно действовать как другой (конечно, нет ничего, чтобы препятствовать тому, чтобы два ESP8266s общались между собой). Здесь мы изучаем некоторые технологии, которые позволяют партнерам взаимодействовать с ESP8266 по протоколу TCP/IP.

Для протокола TCP/IP программный API, первоначально разработанный для платформы Unix и написанный в C, назвали «гнездами». Понятие гнезда - то, что оно логически представляет конечную точку сетевого подключения. Отправитель данных посылает данные через гнездо, и приемник данных получает данные через гнездо. Внедрение самого

«гнезда» обеспечено библиотеками, но логическое понятие гнезда остается. Вы будете работать со «случаем» гнезда, и Вы должны думать о нем как о непрозрачном типе данных, который относится к линии связи.

Гнезда остаются основным API и присутствуют в большинстве языков. Здесь мы обсуждаем некоторые варианты для некоторых более общих языков.

Гнезда TCP/IP

API гнезд - программный интерфейс для работы с организацией сети TCP/IP. Это - вероятно, самый знакомый API для сетевого программирования. Потоки сети TCP/IP прибывают в два аромата … связь, ориентированная по TCP и дейтаграмме, ориентированной по UDP. API гнезд обеспечивает отличные образцы призывов к обоим стилям.

Для TCP сервер построен:

1. Создание гнезда TCP
2. Соединение местного порта с гнездом
3. Урегулирование гнезда послушать способ
4. Принятие новой связи от клиента
5. Получите и пошлите данные
6. Закройте связь клиент-сервер
7. Возвращение к шагу 4

Для клиента TCP мы строим:

1. Создание гнезда TCP
2. Соединение с сервером TCP
3. Отправка данных о данных/получении
4. Закройте связь

Теперь давайте разобьем их в кодовые фрагменты, которые мы можем проанализировать в большей глубине. Определения заголовка для API гнезд могут быть найдены в <lwip/sockets.h>.

И для клиента и для концов сервера, задачей создания гнезда является то же. Это - звонок API на `гнездо ()` функция.

```
международный носок = гнездо (AF_INET, SOCK_STREAM, IPPROTO_TCP)
```

Возвращение из `гнезда ()` является целочисленной ручкой, которая используется, чтобы относиться к гнезду. У гнезд есть много государства, связанного с ними однако, что государство внутреннее к TCP/IP и внедрению гнезд и не должно быть выставлено сетевому программисту. По сути, нет никакой потребности выставить те данные программисту. Мы можем думать о запросе `гнезда ()` как просьба, чтобы время выполнения создало и инициализировало все данные, необходимые для сетевой коммуникации. Те данные принадлежат времени выполнения, и мы встречены «номер ссылки» или ручка, которая действует как полномочие к данным. Когда когда-либо мы хотим впоследствии выполнить работу над тем сетевым подключением, мы пасуем назад в той ручке, которая была ранее выпущена нам, и мы можем коррелировать назад к связи. Это изолирует и изолирует программиста для кишок внедрения TCP/IP и оставляет нас с полезной абстракцией.

Когда мы создаем гнездо стороны сервера, мы хотим, чтобы оно прислушалось к поступающим запросам связи. Чтобы сделать это, мы должны сказать гнездо, на какой номер порта TCP/IP это должно слушать. На данном устройстве только одно применение за один раз может использовать любой данный номер порта. Если мы хотим связать номер порта с заявлением, таким как наше заявление сервера в этом случае, мы выступаем, задача назвала «закрепление», которое связывает (или назначает), номер порта к гнезду, которое в свою очередь принадлежит применению.

```
структура sockaddr_in serverAddress;
serverAddress.sin_family = AF_INET;
serverAddress.sin_addr.s_addr = htonl
(INADDR_ANY); serverAddress.sin_port =
htons (номер порта);
свяжите (носок, (структура sockaddr *) &serverAddress, sizeof
(serverAddress));
```

С гнездом, теперь связанным с местным номером порта, мы можем просить что начало во время выполнения, прислушивающееся к поступающим связям. Мы делаем это, называя слушание () API. Перед запросом `слушают ()`, связи от клиентов были бы отклонены с признаком клиенту, что не было ничего по соответствующему целевому адресу. Как только мы звоним, `слушают ()`, сервер начнет принимать

поступающие связи клиента. API похож:

```
послушайте (гнездо, отставание)
```

Отставание - количество запросов связи, что время выполнения прислушается и примет, прежде чем они будут переданы к заявлению на обработку. Способ думать об этом, предполагают, что Вы - применение, и Вы можете только сделать одну вещь за один раз. Например, Вы можете только говорить с одним человеком за один раз по телефону. Теперь предположите, что у Вас есть секретарь, который обращается с Вашими входящими вызовами. Когда требование прибывает

и Вы не заняты, секретарь руки прочь от требования к Вам. Теперь предположите, что Вы заняты. В то время секретарь подходит к телефону и просит, чтобы посетитель ждал. Когда Вы освобождаете, она вручает Вам требование ожидания. Теперь давайте предположим, что Вы все еще заняты, когда еще один клиент звонит. Она также говорит этому посетителю ждать. Мы начинаем строить очередь посетителей. И это - то, где неудовлетворенное понятие играет роль. Отставание инструктирует время выполнения, сколько требований можно получить и попросить ждать. Если больше требований прибывает, чем наше отставание позволит, время выполнения немедленно отклоняет требование. Мало того, что это предотвращает безудержное потребление ресурса в сервере, оно также может использоваться в качестве признака посетителю, что оно может быть лучше подано, пробуя в другом месте.

Теперь с точки зрения сервера, мы о готовом, чтобы сделать некоторую работу. Применение сервера может теперь заблокировать ожидание поступающих связей клиента. Взгляды состоят в том, что цель применения сервера в жизни состоит в том, чтобы обработать запросы клиента и когда она не сделала, чтобы активный клиент просил, нет ничего для нее, чтобы сделать, но ждать просьбы прибыть. В то время как это - конечно, одна модель, это - не обязательно единственная модель или даже лучшая модель (во всех случаях). Обычно нам нравится, когда наши процессоры «используются». Используемый означает, что, в то время как у этого есть продуктивная работа, это может сделать, тогда это должно сделать это. Если единственной вещью, которую может сделать наша программа, являются сервисные требования клиента, то оригинальная модель имеет смысл. Однако есть

определенные программы, которые, если они не сделали, чтобы клиент просил немедленно обслужить, могли бы провести время, делая что-то еще, что полезно. Мы возвратимся к тому понятию позже. На данный момент мы посмотрим на `принятие ()` вызов функции. То, когда `принимают ()` называют, одна из двух вещей произойдет. Если не будет никакой связи клиента, немедленно ожидая нас, то мы заблокируем до такого времени в будущем, когда связь клиента действительно прибывает. В то время мы проснемся и будем вручены связь с клиентом. Если, с другой стороны, мы звонили, `принимают ()` и уже была связь клиента, ждущая нас, нам немедленно вручат ту связь, и мы продолжаем. В обоих случаях мы звоним, `принимают ()` и возвращены связь с клиентом. Различие между случаями - должны ли мы ждать связи, чтобы прибыть.

Требование API похоже:

```
структура sockaddr_in clientAddress;
socklen_t clientAddressLength = sizeof (clientAddress);
интервал clientSock = принимает (носок, (структура
sockaddr *) &clientAddress, &clientAddressLength);
```

Возвращение из `принимает ()`, новое гнездо, которое представляет связь между клиентом требования и сервером. Жизненно важно понять, что это отлично от гнезда сервера, которое мы создали ранее, который мы связали с нашим сервером, слушая порт. То гнездо все еще живо и здорово и существует, чтобы продолжить обслуживать дальнейшие связи клиента. Недавно возвращенное гнездо - связь для разговора, который был начат этим единственным клиентом. Как все соединения по протоколу TCP, разговор симметричен и двунаправлен.

Это означает, что нет теперь больше понятие клиент-сервера …, обе стороны могут послать и получить, как они хотели бы в любое время.

См. также:

- Википедия –гнезда Беркли
- Справочник Биджа по сетевому программированию

Ошибки из-за неправильного обращения

Большинство API гнезд возвращает международный код возврата.

Если этот кодекс <0 тогда, ошибка произошла.

Природа ошибки может быть найдена, используя названный «errno»

глобального интервала. Однако в многозадачной окружающей среде, работающей с globals, не рекомендуется. В области гнезд мы можем попросить у гнезда последней ошибки, она столкнулась с использованием следующего кодового фрагмента:

```
интервал
   espx_last_socket_errno
   (международное гнездо)
   {интервал мочит = 0;
   u32_t optlen = sizeof (мочит);
   getsockopt (гнездо, SOL_SOCKET, SO_ERROR,
   &ret, &optlen); возвращение мочит;
}
```

Значения ошибок могут быть сравнены с константами. Вот стол констант, используемых в текущем внедрении FreeRTOS:

Символ	Стоимость	Описание
EPERM	1	Операция, не разрешенная
ENOENT	2	Никакой такой файл или папка
ESRCH	3	Никакой такой процесс
EINTR	4	Прерванный системный вызов
EIO	5	Ошибка ввода/вывода
ENXIO	6	Никакое такое устройство или адрес
E2BIG	7	Список аргумента слишком долго
ENOEXEC	8	Исполнительная ошибка формата
EBADF	9	Плохой номер документа
ECHILD	10	Никакие дочерние процессы
EAGAIN	11	Попробовать еще раз
ENOMEM	12	Недостаточно памяти
EACCES	13	Разрешение отрицается
EFAULT	14	Плохой адрес
ENOTBLK	15	Блочное устройство требуется
EBUSY	16	Устройство или занятый ресурс

EEXIST	17	Файл существует
ЕКСДЕВ	18	Связь поперечного устройства
ЕНОДЕВ	19	Никакое такое устройство
ENOTDIR	20	Не справочник
EISDIR	21	Справочник
EINVAL	22	Несостоятельный довод
ENFILE	23	Переполнение таблицы файлов
EMFILE	24	Слишком много открытых файлов

ENOTTY	25	Не пишущая машинка
ETXTBSY	26	Текстовый занятый файл
EFBIG	27	Слишком большой файл
ENOSPC	28	Никакое пространство не уехало на устройстве
ESPIPE	29	Незаконный ищут
EROFS	30	Файловая система только для чтения
EMLINK	31	Слишком много связей
EPIPE	32	Сломанная труба
ЭДОМ	33	Математический аргумент из области func
ERANGE	34	Математический результат не representable
EDEADLK	35	Тупик ресурса произошел бы
ENAMETOOLONG	36	Имя файла слишком долго
ENOLCK	37	Никакие рекордные доступные замки
ENOSYS	38	Функция, не осуществленная
ENOTEMPTY	39	Справочник, не пустой
ELOOP	40	Сталкиваются со слишком многими символическими связями
EWOULDBLOCK EAGAIN	41	Операция заблокировала бы
ENOMSG	42	Никакое сообщение желаемого типа
EIDRM	43	Идентификатор удален
ECHRNG	44	Номер канала из диапазона
EL2NSYNC	45	Уровень 2, не синхронизированный
EL3HLT	46	Уровень 3 остановился
EL3RST	47	Уровень 3 перезагружен
ELNRNG	48	Число связи из диапазона
EUNATCH	49	Драйвер протокола, не приложенный
ENOCSI	50	Никакая доступная структура CSI
EL2HLT	51	Уровень 2 остановился
EBADE	52	Недействительный обмен
EBADR	53	Описатель неверного запроса

ЭКС-ПОЛНЫЙ	54	Полный обмен
ENOANO	55	Никакой анод
EBADRQC	56	Кодекс неверного запроса
EBADSLT	57	Недействительное место
EBFONT	59	Плохой формат файла шрифта
ЭНОШТРАССЕ	60	Устройство не поток
ENODATA	61	Никакие доступные данные
ETIME	62	Таймер истек

ENOSR	63	Из ресурсов потоков
ENONET	64	Машина не находится в сети
ENOPKG	65	Пакет, не установленный
EREMOTE	66	Объект отдален
ENOLINK	67	Связь была разъединена
EADV	68	Рекламируйте ошибку
ESRMNT	69	Ошибка Srmount
ECOMM	70	Ошибка связи на посылает
EPROTO	71	Ошибка протокола
EMULTIHOP	72	Мультиперелет попытался
EDOTDOT	73	RFS определенная ошибка
EBADMSG	74	Не сообщение данных
EOVERFLOW	75	Стоимость, слишком большая для определенного типа данных
ENOTUNIQ	76	Назовите не уникальными в сети
EBADFD	77	Описатель файла в плохом состоянии
EREMCHG	78	Отдаленный адрес изменился
ELIBACC	79	Не может получить доступ к необходимой общей библиотеке
ELIBBAD	80	Доступ к развращенной общей библиотеке
ELIBSCN	81	Раздел .lib в a.out испорчен
ELIBMAX	82	Попытка связаться в слишком многих общих библиотеках
ELIBEXEC	83	Не может должностное лицо общая библиотека непосредственно
EILSEQ	84	Незаконная последовательность байта
ERESTART	85	Прерванный системный вызов должен быть перезапущен
ESTRPIPE	86	Ошибка трубы потоков
EUSERS	87	Слишком много пользователей
ENOTSOCK	88	Эксплуатация гнезда на негнезде
EDESTADDRREQ	89	Адрес получателя требуется
EMSGSIZE	90	Сообщение слишком долго
EPROTOTYPE	91	Протокол неправильно печатает для гнезда

ENOPROTOOPT	92	Протокол, не доступный
EPROTONOSUPPORT	93	Протокол, не поддержанный
ESOCKTNOSUPPORT	94	Тип гнезда, не поддержанный
EOPNOTSUPP	95	Операция, не поддержанная на транспортной конечной точке
EPFNOSUPPORT	96	Семейство протоколов, не поддержанное
EAFNOSUPPORT	97	Семья адреса, не поддержанная протоколом

EADDRINUSE	98	Адрес уже в использовании
EADDRNOTAVAIL	99	Не может назначить требуемый адрес
ENETDOWN	100	Сеть снижается
ENETUNREACH	101	Сеть недостижима
ENETRESET	102	Сеть пропустила связь из-за сброса
ECONNABORTED	103	Программное обеспечение вызвало аварийное прекращение работы связи
ECONNRESET	104	Связь перезагружена ровесником
ENOBUFS	105	Никакое буферное доступное пространство
EISCONN	106	Транспортная конечная точка уже связана
ENOTCONN	107	Транспортная конечная точка не связана
ESHUTDOWN	108	Не может послать после транспортного закрытия конечной точки
ETOOMANYREFS	109	Слишком много ссылок: не может соединить
ETIMEDOUT	110	Связь, рассчитанная
ECONNREFUSED	111	В соединении отказано
EHOSTDOWN	112	Хозяин снижается
EHOSTUNREACH	113	Никакой маршрут, чтобы принять
EALREADY	114	Операция, уже происходящая
EINPROGRESS	115	Операция, теперь происходящая
ESTALE	116	Несвежий дескриптор NFS
EUCLEAN	117	Структуре нужна очистка
ENOTNAM	118	Не XENIX назвал файл типа
ENAVAIL	119	Никакие доступные семафоры XENIX
EISNAM	120	Названный файл типа
EREMOTEIO	121	Отдаленная ошибка ввода/вывода
EDQUOT	122	Квота превышена
ENOMEDIUM	123	Никакая среда не найдена
EMEDIUMTYPE	124	Неправильный средний тип

Гнезда - принимают ()

Примите поступающий запрос.

```
интервал принимает (интервал s, структура sockaddr *addr, socklen_t
*addrlen)
```

Страница 168

Здесь мы можем заблокировать ожидание поступающего запроса связи от гнезда сервера. Мы немедленно возвратимся, если будет принятие ожидания связи клиента. Адрес клиента возвращен нам наряду со своей длиной.

Если мы пытаемся принять слишком много параллельных гнезд, ESP8266 может возвратить `ENFILE`, чтобы указать, что у нас есть переполнение.

Возвращение - принятая связь гнезда с клиентом или-1 если ошибка.

Гнезда - связывают ()

Свяжите гнездо с адресом.

```
свяжите (интервал s, структура константы sockaddr *имя, socklen_t namelen)
```

Параметр `имени` - адрес гнезда, который будет связан с гнездом. `namelen` обеспечивает длину адреса. Если `sin_add.s_addr` - `htonl (INADDR_ANY)` тогда, мы - сервер, слушающий на любом поступающем IP-адресе.

Возвращение <0 на ошибке.

Вот пример того, что мы определяли нас как сервер:

```
структура sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_addr.s_addr = htonl
(INADDR_ANY); serverAddr.sin_port =
htons (80);
```

```
международное дистанционное управление = связывает (serverSocket,
(структура sockaddr *) &serverAddr, sizeof (serverAddr));
```

Гнезда - близко ()

Закройте соответствующее гнездо.

```
близко (интервал s)
```

Закройте гнездо.

Гнезда - closesocket ()

Закройте соответствующее гнездо.

```
closesocket (интервал s)
```

Закройте гнездо.

Гнезда - соединяются ()

Соединитесь с сервером.

```
соединитесь (интервал s, структура константы sockaddr *partnerAddr,
socklen_t addrlen)
```

Соедините гнездо с партнером. Адрес партнера поставляется в `partnerAddr` области. Это - клиент начатое требование и, как ожидают, соединится с сервером слушания.

Гнезда - fcntl ()

Выполните функции управления.

```
fcntl (интервал s, интервал cmd, интервал val)
```

Мы можем установить функции управления на гнездах здесь.

Команда	Стоимость	Описание
F_SETFL	O_NONBLOCK	Установите гнездо, не блокирующее.

Гнезда - freeaddrinfo ()

Гнезда - getaddrinfo ()

Гнезда - gethostbyname ()

Гнезда - getpeername ()

Восстановите адрес, связанный с партнером/ровесником, с которым связано гнездо.

```
getpeername (интервал s,
   структура
   sockaddr
   *peerAddr,
   socklen_t
   *namelen)
```

Обратите внимание, что `namelen` должен быть запущен с размером доступного буфера адреса.

Гнезда - getsockname ()

Восстановите текущий местный адрес, с которым связано гнездо.

```
getsockname (интервал s,
   структура
   sockaddr
   *имя,
   socklen_t
   *namelen)
```

Обратите внимание, что `namelen` должен быть запущен с размером

доступного буфера адреса.

Гнезда - getsockopt ()

```
getsock
   opt
   (инт
   ерва
   л s,
   межд
   унар
   одны
   й
   уров
   ень,
   инте
   рвал
   optn
   ame,
```



```
   пустота
   *optval,
   socklen_
   t
   *optlen)
```

Важный пример использования этой функции состоит в том, чтобы восстановить последнюю ошибку, связанную с гнездом. Следующий кодовый фрагмент иллюстрирует это:

```
интервал
   espx_last_socket_errno
   (международное гнездо)
   {интервал мочит = 0;
   u32_t optlen = sizeof (мочит);
   getsockopt (гнездо, SOL_SOCKET, SO_ERROR,
   &ret, &optlen); возвращение мочит;
}
```

Гнезда - htonl ()

Преобразуйте отформатированное длинное целое число хозяина в сетевой порядок байтов.

Гнезда - htons ()

Преобразуйте отформатированное короткое целое число хозяина в

сетевой порядок байтов.

Гнезда - inet_ntop ()

Гнезда - inet_pton ()

Гнезда - ioctlsocket ()

```
ioctl (интервал s, длинный cmd, пустота *argp)
```

Гнезда - слушают ()

Начните прислушиваться к поступающим связям.

```
послушайте (интервал s, международное отставание)
```

Если мы будем связаны как сервер, то мы начнем прислушиваться к поступающим запросам связей на гнезде. `Неудовлетворенный` параметр определяет, сколько гнезд мы можем держать ручкой к тому, прежде чем мы примем их.

Возвращаемое значение <0 означает ошибку.

Гнезда - читали ()

Получите данные от партнера.

```
ssize_t читал (интервал s, пустота *мадам, size_t len)
```


Подобный recv () функция.

См. также:

- Гнезда – recv ()
- Гнезда – recvfrom ()

Гнезда - recv ()

Получите данные от партнера.

```
ssize_t
   recv
   (интерва
   л s,
   пустота
   *мадам,
```

```
   size_t
   len,
   международные флаги)
```

Эта функция возвращает число байтов, на самом деле полученных. Ценность-1 указывает на ошибку. Ценность ноля указывает на партнера, закрывавшего связь.

Флаги - булева комбинация:

- MESSAGE_CMSG_CLOEXEC
- `MESSAGE_DONTWAIT` – Указывают, что мы не хотим блокировать ожидание данных. Если нет никаких данных, немедленно доступных для нас, чтобы получить, мы возвращаемся-1, чтобы указать на ошибку, и код ошибки - «EAGAIN».
- MESSAGE_ERRQUEUE
- MESSAGE_OOB
- MESSAGE_PEEK
- MESSAGE_TRUNC
- MESSAGE_WAITALL

См. также:

- Гнезда – читали ()
- Гнезда – recvfrom ()
- человек (2) - recv

Гнезда - recvfrom ()

```
recvfro
   m
   (инт
   ерва
   л s,
   пуст
   ота
   *мад
   ам,
   size
   _t
   len,
   межд
   унар
   одны
   е
   флаг
   и,
   структура
   sockaddr *от,
   socklen_t
   *fromlen)
```

См. также:

- Гнезда – читали ()

Страница 172

- Гнезда – recv ()

Гнезда - избранный ()

Проверьте на доступные данные для чтения или написания.

```
выберите
   (интервал
   maxfdp1,
   fd_set
   *readset,
   fd_set
   *writeset,
   fd_set
   *exceptset
   ,
   структура timeval *перерыв)
```

На некоторых системах Linux, чтобы использовать отобранный, можно было бы включать `<sys/select.h>`. В ESP-IDF, который не присутствует, но, кажется, не необходим.

Гнезда - посылают ()

Пошлите ряд байтов вниз гнездо партнеру.

```
ssize_t посылают (интервал s, пустота константы *dataptr, size_t размер,
международные флаги)
```

Данные, на которые указывает `dataptr` для

байтов `размера`, переданы.См. также:

- человек (2) – посылает

Гнезда - sendto ()

Пошлите данные партнеру UDP.

```
sendto (интервал s,
  пустота
  константы
  *dataptr,
  size_t
  размер,
  международные флаги,
  структура
  константы
  sockaddr *к,
  socklen_t tolen)
```

Гнезда - setsockopt ()

```
setsockopt
   (интерв
   ал s,
   междуна
   родный
   уровень
   ,
   интервал optname,
   пустота
   константы
   *optval,
   socklen_t
   optlen)
```

Варианты, которые, как ожидают, доступны:

- `TCP_NODELAY` – Отключают алгоритм Nagle.
- `SO_KEEPALIVE` – Позволяют живое свистение.

Гнезда - закрытие ()

Части закрытия гнезда.

```
закрытие (интервал s, интервал, как)
```

Закрытие все или часть гнезда.

Гнездо - закрытие на основе, как параметр, который может быть одним из:

- `SHUT_RD` – Нет далее получает, позволены.
- `SHUT_WR` – Нет далее пишет, позволены.
- `SHUT_RDWR` – Нет далее читает или пишет, позволены.

Гнезда - гнездо ()

Создайте новое гнездо для определенной области, типа и протокола.

```
гнездо (международная область, международный тип, международный протокол)
```

Область может быть одним из:

- `AF_INET` – TCP/IP
- Другие …

Тип может быть одним из:

- SOCK_STREAM
- SOCK_DGRAM
    - Протокол SOCK_RAW может быть одним из:
- IPPROTO_IP
- IPPROTO_TCP
- IPPROTO_UDP

Образец общего использования:

```
международный носок = гнездо (AF_INET, SOCK_STREAM, IPPROTO_TCP)
```

Возвращает новый описатель гнезда или стоимость <0 на ошибке.

Гнезда - пишут ()

```
напишите (интервал s, пустота константы *dataptr, size_t размер)
```


Структуры данных гнезда

Гнезда - структура sockaddr

Гнезда - структура sockaddr_in

- sin_family – AF_INET
- sin_port
- структура in_addr sin_addr – у Этой структуры есть названный s_addr участника, который является IP-адресом. У специальных ценностей есть специальные значения. Например, INADDR_ANY - любой адрес.

Явские гнезда

API гнезд - defacto стандартный API для программирования против TCP/IP. Мой предпочтительный язык программирования - Ява, и у этого есть полная поддержка гнезд. То, что это означает, - то, что я могу написать находящееся в явском применение, которое усиливает гнезда к cmomunicate с ESP*. Я могу послать и получить данные через

довольно легко.

На Яве есть два основных класса, который представляет гнезда, это - `java.net. Гнездо`, которое представляет клиентское приложение, которое сформирует связь и второй класс, является `java.net. ServerSocket`, который представляет сервер, который слушает на гнезде, ждущем связи клиента. Так как ESP* может быть или клиентом или сервером, оба из этих Явских классов сыграют роль.

Чтобы соединиться с ESP* бегущий как сервер, мы должны знать IP-адрес устройства и номера порта, на котором это слушает. Как только мы знаем их, мы можем создать случай Явского клиента с:

```
Гнездо clientSocket = новое Гнездо (ipAddress, порт);
```

Это сформирует связь с ESP*. Теперь мы можем попросить и `InputStream`, из которого можно получить данные партнера и `OutputStream`, которому мы можем написать данные.

```
InputStream =
clientSocket.getInputStream (); рот
OutputStream =
clientSocket.getOutputStream ();
```

Когда мы закончены со связью, мы должны звонить `близко ()`, чтобы закрыть Явскую сторону связи:

```
clientSocket.close ();
```

Это действительно настолько просто. Вот пример заявления:

```
пакет kolban;

импорт java.io.

OutputStream;

импорт java.net.

Гнездо;
```



```
импорт org.apache.commons.cli. CommandLine;
импорт org.apache.commons.cli.
CommandLineParser; импорт
org.apache.commons.cli. DefaultParser;
импорт org.apache.commons.cli. Варианты;

общественный  класс
   SocketClient
   {частное     имя
   хоста
   Последовательно
   сти;     частный
```

```
   международный
   порт;

   общественное статическое
      недействительное основное
      (Последовательность [] args)
      {варианты Вариантов = новые
      Варианты (); options.addOption
      («h», истинное, «имя хоста»);
      options.addOption («p», верный,
      «порт»);
      Анализатор CommandLineParser = новый
      DefaultParser (); попробуйте {
         CommandLine cmd = parser.parse (варианты, args);

         Клиент SocketClient = новый
         SocketClient (); client.hostname
         = cmd.getOptionValue («h»);
         client.port = Integer.parseInt (cmd.getOptionValue
         («p»)); client.run ();
      } выгода
         (Исключение
         e)
         {e.printStack
         Trace ();
      }
   }

   общественн
      ый
      недейств
      ительный
      пробег
      ()
      {попытка
      {
         международный РАЗМЕР = 65000;
         данные о байте [] =
         новый байт [РАЗМЕР];
         для (интервал i = 0; я
         <РАЗМЕР; я ++) {
            данные [я] = 'X';
         }
         Гнездо s1 = новое Гнездо (имя
         хоста, порт); рот OutputStream
         =    s1.getOutputStream    ();
         os.write (данные);
         s1.close ();
         System.out.println («Данные
         послан!»);
      } выгода
         (Исключение
         e)
         {e.printStack
         Trace ();
      }
   }
}//Конец
класса//К
онец
файла
```

Чтобы настроить JAVA-приложение как, сервер гнезда так же легок. На этот раз мы создаем случай использования класса `SocketServer`:

```
SocketServer serverSocket = новый SocketServer (порт)
```

Снабженный порт является номером порта на машине, на которой JVM бежит, который будет конечной точкой удаленных запросов связи клиента. Как только у нас есть а

Случай `ServerSocket`, мы должны ждать поступающей связи клиента. Мы делаем это использование названного метода API блокирования `принимает ()`.

```
Гнездо partnerSocket = serverSocket.accept ();
```

Это требование блоки, пока клиент не соединяется, прибывает. Возвращенный `partnerSocket` - подключенное гнездо партнеру, который может используемый тем же способом, как мы ранее обсудили для связей клиента. Это означает, что мы можем просить `InputStream`, и `OutputStream` возражает, чтобы читать и написать и от партнера. Так как Ява - многопоточный язык, как только мы просыпаемся от, `принимают ()`, мы можем выдать полученное гнездо партнера к новой ветви дискуссии и повторить, `что принятие ()` призывает к другим параллельным связям. Не забудьте `закрывать ()` любые связи гнезда партнера, которые Вы получаете, когда Вы сделаны с ними.

До сих пор мы говорили об ориентированных связях TCP, где, как только связь открыта, это остается открытым, пока не закрыто за это время или конец может послать или получить независимо от другого. Теперь мы смотрим на дейтаграммы, которые используют протокол UDP.

Основной класс позади этого называют `DatagramSocket`. В отличие от TCP, класс `DatagramSocket` используется оба для клиент-серверов.

Во-первых, давайте посмотрим на клиента. Если мы хотим написать Явскому клиенту UDP, то мы создадим случай использования `DatagramSocket`:

```
DatagramSocket clientSocket = новый DatagramSocket ();
```

Затем мы «соединимся» с отдаленным партнером UDP. Мы должны будем знать IP-адрес и порт, на который слушает партнер. Хотя API называют, «соединяются», мы должны понять, что никакая связь не сформирована. Дейтаграммы без установления соединения поэтому, что мы на самом деле делаем, связывает наше гнездо клиента с

гнездом партнера на другом конце так, чтобы**, когда** мы на самом деле хотим послать данные, мы знали, куда послать его в.

```
clientSocket.connect (ipAddress, порт);
```

Теперь мы готовы послать дейтаграмму, используя посылание () метод:

```
Данные DatagramPacket = новый DatagramPacket (новый
байт [100], 100); clientSocket.send (данные);
```

Чтобы написать слушателю UDP, который прислушивается к поступающим дейтаграммам, мы можем использовать следующее:

```
DatagramSocket serverSocket = новый DatagramSocket (порт);
```

Порт здесь - номер порта на той же машине как JVM, который будет использоваться, чтобы прислушаться к поступающим связям UDP.

Чтобы ждать поступающей дейтаграммы, требование `получает ()`.

```
Данные DatagramPacket = новый DatagramPacket (новый
байт [100], 100); clientSocket.receive (данные);
```


Если Вы собираетесь использовать Явские API Гнезда, читайте, JavaDoc полностью для этих классов есть много особенностей и вариантов, которые не были перечислены здесь.

См. также:

- Явская обучающая программа: Все О Гнездах
- JDK 8 JavaDoc

WebSockets

WebSockets - и API и протокол, введенный в HTML5. Проще говоря, если мы воображаем заседание сервера HTTP, ждущее поступающих запросов HTTP, мы можем преобразовать текущий запрос в связь гнезда между сервером и браузером, таким образом, что любой конец может послать данные, которые будут получены его партнером.

Здесь мы видим сырую просьбу модернизировать связь HTTP со связью WebSocket:

```
ДОБЕРИТЕС
Ь /
Хозяин
HTTP/1.1:
192.168.1
.10
Связь:
Модерниза
ция
```

```
Pragma:
без
тайников
Контроль
тайника:
Модернизация
без тайников:
Происхождение
websocket:file:
// Sec-
WebSocket-
Version: 13
Пользователь-агент: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, как Геккон) Хромовое/46.0.2490.86 Сафари/537.36
Принятый закодированный: gzip,
выкачайте, sdch Принимать-язык:
en-США, en; q=0.8 Sec-WebSocket-
Key: saim6TzFH+zVb4qY2nrh0Q ==
Sec-WebSocket-Extensions: permessage-выкачайте; client_max_window_bits
```

См. также:

- html5rocks –Представление WebSockets: Обеспечение Гнезд к сети
- Протокол WebSocket – RFC6455
- WebSocket API

Приложение браузера WebSocket

Самая высокая вероятность - то, что Вы будете управлять своим ESP8266 как сервером WebSocket. Это подразумевало бы, что Вы собираетесь иметь ведшие заявления браузера, которые будут соединяться с сервером WebSocket как клиенты и оттуда, Вы будете, вероятно, писать некоторый код клиента WebSocket … если ничто иное тогда для модульного тестирования Ваш сервер. Из-за этого мы теперь проведем некоторое время, говоря о том, что включено в письменной форме клиентское приложение WebSocket.

Давайте предположим, что мы будем писать JavaScript, ведший в браузере. Мы начинаем, создавая случай объекта WebSocket, проходящего в URL к серверу WebSocket:

```
вар ws = новый WebSocket («ws://<somehost> [: <someport>]»);
```

WebSocket API главным образом управляем событиями и есть событие числа типы интереса для нас:

- `открытый` – Призванный, когда связь с сервером WebSocket была установленный и мы теперь готовы послать или получить

данные. Мы можем определить это с «onopen» собственностью WebSocket как ссылка функции.

- сообщение – Получает сообщение от сервера. Мы можем определить это с собственность «onmessage» WebSocket как ссылка функции.
- ошибка – Получает признак, что ошибка была обнаружена. Мы можем определить это с «onerror» собственностью WebSocket как ссылка функции.
- близко – Получают признак, что просьба закрыть связь была обнаруженный. Мы можем определить это с «onclose» собственностью WebSocket как ссылка функции.

Обработчики событий могут быть зарегистрированы или в «на <Событии>» механизм или с addEventListener () требование.

Есть два метода, определенные на объекте WebSocket. Это:

- пошлите – Посылают данные в сервер WebSocket
- близко – Близко связь с сервером WebSocket. Завершение () метод берет два параметра:
  - близкий кодекс – целое число закрывает кодекс, описывающий причину завершения.
    - 1000 – CLOSE_NORMAL
    - 1001 – CLOSE_GOING_AWAY
  - сообщение о состоянии – последовательность, описывающая близкую причину.

Наконец, есть несколько признаков:

- readyState – состояние связи WebSocket. Ценности включают:
  - WebSocket. СОЕДИНЕНИЕ
  - WebSocket. ОТКРЫТЫЙ
  - WebSocket. ЗАКРЫТИЕ
  - WebSocket. ЗАКРЫТЫЙ

Страница 179

- bufferedAmount – Объем данных, который буферизован надвигающаяся передача к

Сервер WebSocket

- `протокол` – сервер WebSocket выбрал используемый протокол.

FreeRTOS WebSocket

FreeRTOS SDK распределяет внедрение noPoll общедоступного проекта. Чтобы использовать noPoll в Вашем приложении, Вы должны включать «`nopoll.h`».

Контекст очень важен и должен быть создан в начале и избавлен в конце. Например:

```
noPollCtx *ctx = nopoll_ctx_new ();
//Сделайте
что-то ...
nopoll_ctx
_unref ();
```

Как только контекст был создан, мы можем тогда зарегистрироваться как сервер:

```
noPollCon *слушатель = nopoll_listener_new (ctx,
«0.0.0.0», «1234»); nopoll_ctx_set_on_msg (ctx,
listener_on_message_handler, ПУСТОЙ УКАЗАТЕЛЬ);
nopoll_loop_wait (ctx, 0);
```

Когда сообщение будет получено, зарегистрированная функция (listener_on_message_handler) будет призвана:

```
пустота
  listener_on_message_handler
  (noPollCtx *ctx,
  noPollConn
  *ведут,
  noPollMsg
  *сообщение,
  noPollPtr
  *userData)
  {//Делают
  что-то
  nopoll_conn_send_text (ведут, «Спасибо», 5);
}
```

Пример применение WebSocket поставляется Espressif в FreeRTOS SDK для ESP8266. Образец может быть найден в/examples/websocket_demo.

См. также:

- noPoll: набор инструментов OpenSource WebSocket
- Протокол WebSocket – RFC6455
- WebSocket API

Мангуста WebSocket

Пользуясь библиотеками Мангусты Сесэнты, мы можем установка сервер WebSocket. После подготовки закрепления для поступающей сети просит, чтобы мы могли назвать `mg_set_protocol_http_websocket ()`. Это приложит обработчик событий к сетевому уровню протокола, чтобы обращаться с событиями, связанными с WebSockets. А именно, это следующие события, которыми мы интересуемся:

- MG_EV_WEBSOCKET_HANDSHAKE_REQUEST
- MG_EV_WEBSOCKET_HANDSHAKE_DONE
- MG_EV_WEBSOCKET_FRAME

Когда `MG_EV_WEBSOCKET_HANDSHAKE_REQUEST` получен, данные содержат разобранный запрос HTTP как `структуру http_message`.
Структура http_message содержит:

- сообщение
- метод
- uri
- первичный
- resp_code
- resp_status_msg
- query_string
- header_names
- header_values
- тело

Когда `MG_EV_WEBSOCKET_FRAME` получен, данные содержат ссылку на `структуру websocket_message`. `Структура websocket_message` содержит:

- `данные` – данные прошли от партнера.
- `размер` – размер переданных данных.
- `флаги` – (неизвестные) Флаги.

Веб-серверы

Веб-сервер - компонент программного обеспечения, который прислушивается к поступающим запросам HTTP от веб-браузеров. Есть

много внедрений веб-серверов, которые могут бежать в окружающей среде ESP8266.

Мангуста

Мангуста предоставляет API, который можно использовать, чтобы построить богатый и мощный сервер HTTP. На высоком уровне мы называем `mg_mgr_init ()`, чтобы инициализировать окружающую среду. Затем мы связываем его с укладчиком, использующим `mg_bind ()`. Наконец, мы получаем голоса сервера для работы, используя `mg_mgr_poll ()`.


```
пустота
  setupMongoo
  se ()
  {структура
  mg_mgr
  менеджер;
  printf («Стартовая
  мангуста setup\n»);
  mg_mgr_init (&mgr, ПУСТОЙ
  УКАЗАТЕЛЬ); printf
  («Succesfully inited\n»);
  mg_bind (&mgr, «80»,
  evHandler); printf
  («Succesfully bound\n»);
  в то время как (1) {
    mg_mgr_poll (&mgr, 1000);
  }
}
```

Когда запрос прибывает от браузера, мы полагаем, что событие, которое передано к обработчику событий. Подпись обработчика событий:

```
пустота eventHandler (
  структура
  mg_connection
  *nc, интервал
  ev,
  пустота *evData)
```

Где:

- `nc` – связь, которая получила событие.
- `ev` – тип события, которое вызвало отзыв.
- `evData` – Данные связались с событием.

К настоящему времени мы получим отзывы для связей гнезда. Если

мы желаем, мы можем теперь зарегистрировать это, мы хотим разобрать поступающие данные как запрос HTTP. Мы делаем это, звоня `mg_set_protocol_http_websocket ()`.

Когда установка как сервер HTTP, поступающий запрос браузера появится с типом событий MG_EV_HTTP_REQUEST. `evData` прошел в, будет случай `структуры http_message`, от которого может быть определена природа запроса.

См. также:

- GitHub:cesanta/mongoose
- Центр разработчика мангусты

Программирование Затмения использования

Затмение - популярная общедоступная структура, прежде всего, используемая для хостинга инструментов разработки приложений. Хотя, прежде всего, приспособлено для строительства JAVA-приложений, у этого также есть первый класс C и C ++ поддержка.

Проект для строительства приложений ESP8266, используя Затмение может быть найден здесь:

Страница 182

- http://www.esp8266.com/viewtopic.php?f=9&t=820

Не включайте места ни в одну из частей пути, указывающих на рабочее пространство. Здесь некоторые примечания по установке этого проекта … однако, всегда читают документацию, сопровождающую проект.

Загрузите `Espressif ESP8266 DevKit vxxx x86`. Это - большая загрузка приблизительно 125 мегабайтов.

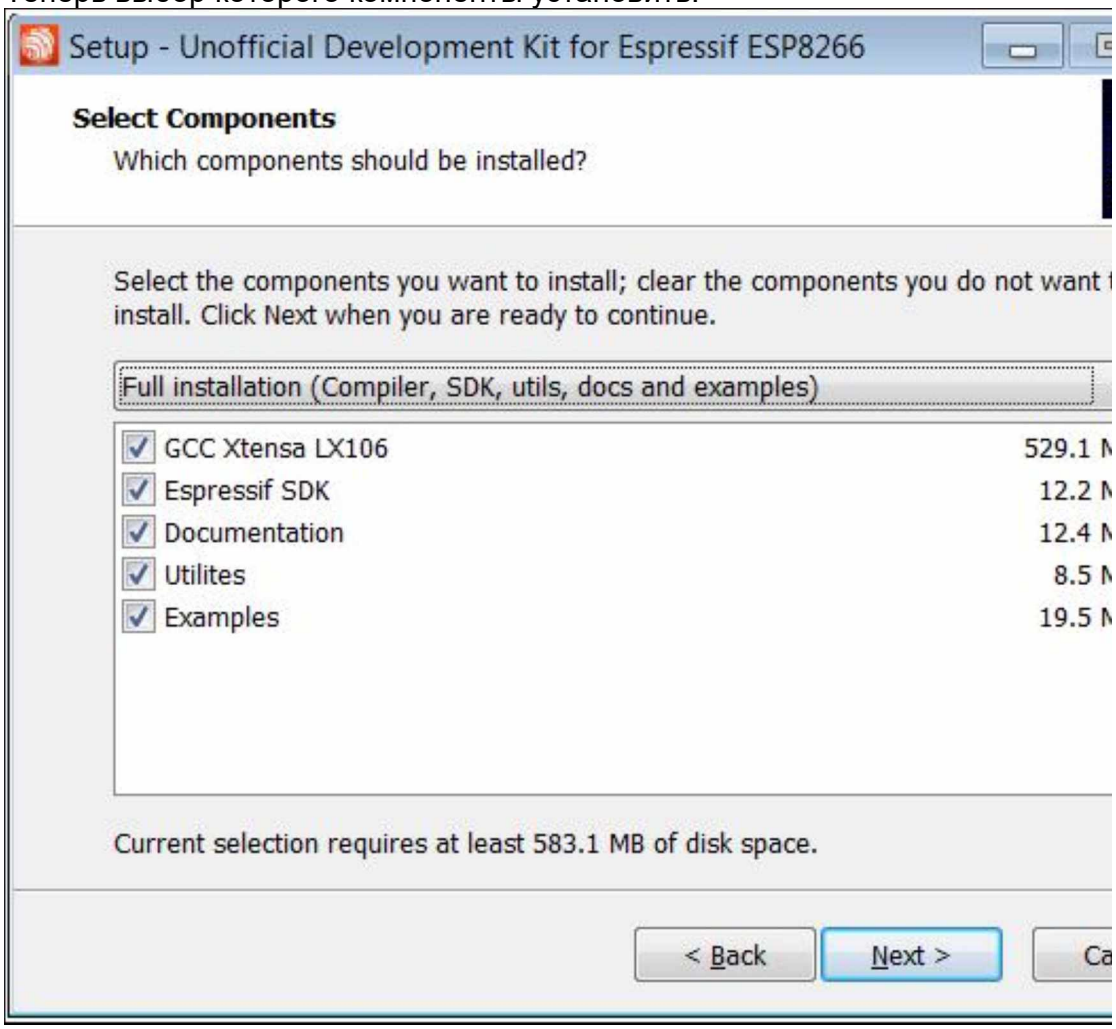
Управляйте инсталлятором. Это попросит у Вас Вашего выбора инсталляционного языка.

Затем прибывает заставка:

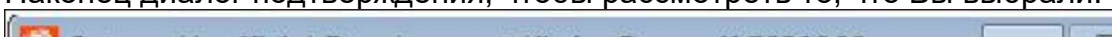
Затем прибывает лицензионное соглашение:

Страница 183

Теперь выбор которого компоненты установить:



Наконец диалог подтверждения, чтобы рассмотреть то, что Вы выбрали.



Результатом этого будет новая структура каталогов в `C:\Espressif\`.

Есть другие зависимости, в которых Вы будете нуждаться, которые перечислены в связи выше. Они включают:

- Явская окружающая среда времени выполнения. Я использую последнюю Яву 8 от Oracle.
- Окружающая среда затмения с C/C ++ инструменты разработчика. Я использую последний выпуск «Марса».

- `MinGW` – инструменты Unix и утилиты, которые выполняют на Windows.
- `Инсталляционный помощник MinGW` – тайник и список пакетов MinGW, которые должны быть установлены для правильной операции.

Мэкефайлы, снабженные пакетом, ключевые. Они были обработаны, чтобы обеспечить, самое легкое собирает. Цели, содержавшие в Мэкефайлах, включают:

- `все` – Собирают весь кодекс, но не вспыхивают.
- `чистый` – Убирают, любой предыдущий строит.
- `вспышка` – Собирает кодекс в случае необходимости и затем вспыхивает.
- flashboot
- flashinit

- flashonefile

Есть некоторые флаги, которые используются с Makefile, который Вы можете отредактировать. Они включают:

- `VERBOSE=1` – Позволяют многословие, которое включает информацию об отладке. Конкретно
  команды компиляции

показывают.См. также:

- Eclipse.org
- Затмите C/C ++ развитие, Оснащающее (CDT)
- Основная нить форума

Установка Затмения Последовательный терминал

Хотя есть много превосходных последовательных терминалов, доступных как автономные Приложения Windows, альтернатива - Терминал Затмения, у которого также есть последовательная поддержка. Это позволяет последовательному терминалу появляться как представление в Затмении IDE. Это не прибывает установленное по умолчанию, но шаги, чтобы добавить не сложны.

Сначала начните Затмение (я

использую выпуск Марса).

Пойдите в `Помощь>`,

Устанавливают новое

программное обеспечение.

Выберите хранилище загрузки

затмения.

Выберите Мобильный и разработка Устройств> Терминал ТМ.

Страница 186

Шаг через следующие разделы и когда он побужден перезапустить, примите да. Мы не готовы использовать его все же, мы должны добавить поддержку последовательного порта в Затмение. Вернитесь к `Помощи>, Устанавливают новое программное обеспечение` и добавляют новое хранилище

Страница 187

URL хранилища:

- http://archive.eclipse.org/tm/updates/rxtx/

Теперь мы можем выбрать библиотеку поддержки времени выполнения Последовательного порта:

Страница 188

Выполните дальнейшие навигационные экраны и перезапустите Затмение, когда Вы побуждены.

Нам теперь установили предельную поддержку и готовы использовать ее. Из `Windows> Выставочное Представление> Другой` мы найдем новую категорию под названием «Терминал».

Страница 189

Открытие этого добавляет Предельное представление о нашей перспективе. Есть кнопка, которая позволит нам открывать новый предельный случай, который показывают по следующему изображению:

Нажатие на это поднимает диалог, прося у нас тип терминала и свойств. В наших целях мы хотим выбрать последовательный терминал. Не забывайте также устанавливать порт и скорость передачи в бодах соответствовать тому, что использует Ваш ESP8266.

После нажимания OK после нескольких секунд мы будем видеть, что связаны, и появляется новый символ разъединения:

И теперь терминал активен. В моих целях я соединяю этот терминал с UART1 ESP8266 для отладки, оставляя UART0 для высвечивания новых копий моего заявления. Вот пример того, на что похоже мое типичное окно:

Вы можете инвертировать цвета, чтобы произвести белое на черной визуализации, которую предпочитают многие пользователи.

Веб-разработка используя Затмение

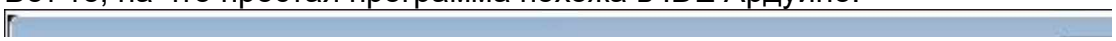
Затмение также обеспечивает окружающую среду веб-разработки первого класса для написания и тестирования веб-приложений включая страницы HTML. Предложено, чтобы Инструменты веб-разработчика Затмения были установлены.

Программирование использования IDE Ардуино

Значительно прежде было ESP8266, был Ардуино. Жизненно существенный вклад в общедоступное сообщество аппаратных средств и точку входа для большинства людей, увлеченных своим хобби, в мир домашних построенных схем и процессоров.

Одна из ключевых достопримечательностей об Ардуино - своя относительная низкая сложность, разрешающая всем способность построить что-то быстро и легко. Integrated Development Environment (IDE) для Ардуино всегда была бесплатной для загрузки с Интернета. Если бы профессиональный программист должен был сесть с ним, то они были бы потрясены в его очевидных ограниченных возможностях. Однако подмножество функции, которую это обеспечивает по сравнению с «полнофункциональной» IDE, оказывается, покрывает 90% того, чего каждый хочет достигнуть. Объедините это с интуитивным интерфейсом, и IDE Ардуино - сила, с которой будут считаться.

Вот то, на что простая программа похожа в IDE Ардуино:



В языке Ардуино заявление называют «эскизом». Лично, я не поклонник той фразы, но я уверен, что исследование было проведено, чтобы узнать, что это - наименее пугающее имя

Страница 193

поскольку, что иначе назвали бы программой языка C и это запугает наименьшее количество числа людей.

IDE назвали кнопку, «Проверяют», который, когда по нему щелкают, собирает программу. Конечно, у этого также будет побочный эффект, что он проверит, что программа собирает чисто …, но компиляция - то, что он делает. Вторую кнопку называют «Закачкой», которая, когда по ней щелкают, что она делает, является, развертывают применение к Ардуино.

В дополнение к обеспечению редактора языка C плюс инструменты, чтобы собрать и развернуться, IDE Ардуино обеспечивает предварительно снабженные библиотеки установленного порядка C, который «скрывает» сложные детали внедрения, которые могли бы иначе быть необходимы, программируя советам Ардуино. Например, программирование UART должно было бы, несомненно, установить регистры, перерывы ручки и т.д. Вместо того, чтобы делать бедных пользователей должны изучить эти технические API. люди Ардуино обеспечили библиотеки высокого уровня, которые можно было назвать из эскизов с более чистыми интерфейсами, которые скрываются, механические «жадно едят», который происходит под покрытиями. Это понятие - ключевой …, поскольку эти библиотеки, так же как что-либо еще, предоставляют окружающую среду программистам Ардуино.

Интересный, поскольку эта история может быть, Вы можете спрашивать, как это касается нашей истории ESP8266? Ну, группа талантливых людей пристроила Общедоступный проект на GitHub, который обеспечивает «программное расширение», или «расширение» к инструменту IDE Ардуино (помните, что IDE Ардуино самостоятельно свободен). То, что делает это расширение, позволяют писать эскизы в

IDE Ардуино, которые усиливают интерфейсы библиотеки Ардуино, которые, в собирают и время развертывания, генерируют код, который будет работать на ESP8266. То, что это эффективно означает, - то, что мы можем использовать IDE Ардуино и создать приложения ESP8266 с минимумом суеты.

Последствия Ардуино поддержка IDE

Способность рассматривать ESP8266, как будто это было похоже на Ардуино, является понятием, что я не был в состоянии полностью поглотить все же. ESP8266 - центральный процессор Tensilica в отличие от Ардуино, который является центральным процессором ATmega. Espressif создали посвященный и спроектированный API в форме своего SDK для непосредственно выставленных API ESP8266. Библиотеки Ардуино для ESP8266, кажется, наносят на карту свое намерение к этим выставленным API. По этим причинам и подобный, можно было бы утверждать, что поддержка Ардуино - ненужный фасад сверх совершенно приятной атмосферы и налагая «иностранную» технологическую модель сверх родных функций ESP8266, мы маскируем доступ, чтобы понизить уровни знаний и функцию. Далее, размышление о ESP8266, как будто это был Ардуино, может привести к проблемам проектирования. Например, ESP8266 нужен регулярный контроль, чтобы обращаться с WiFi и другими внутренними действиями. Это находится в противоречии с моделью Ардуино, где программист может сделать то, что он хочет в функции петли столько, сколько он хочет.

Оборот - то, что кривая обучения, чтобы получить что-то работающее на Ардуино, как показывали, была чрезвычайно низкой. Не занимает много времени вообще получать blinky свет, идущий на макет. С тем ходом мыслей, почему пользователи ESP8266 должны быть оштрафованы за то, что они имели необходимость установить и узнать больше, сложные цепи инструмента и синтаксис, чтобы достигнуть того же результата с большим количеством ESP8266 ориентировали инструменты и методы? Название игры должно быть должно позволить людям переделывать центральные процессоры и датчики, не имея необходимость иметь университетские дипломы в области вычислительной науки или электротехники и если цена, которую каждый платит, чтобы добраться,

там должна вставить «простой использовать» иллюзию тогда почему нет? Если я строю бумажный самолетик и бросаю его мое окно …, я могу получить удовольствие от этого. Астроном НАСА не должен насмехаться над моими действиями, или отсутствие знаний аэродинамики … согнутая бумага сделало свою работу, и я достиг своей цели. Однако, если моя работа состояла в том, чтобы поместить человека на луну, способность визуализировать факты технологии на «реалистическом» уровне становится чрезвычайно важной.

Установка IDE Ардуино с поддержкой ESP8266

Чтобы собрать эту окружающую среду, нужно загрузить текущую версию IDE Ардуино. Это будет приблизительно 140 мегабайтов.

Я загружаю версию zip-файла и затем извлекаю ее содержание.

Затем, мы начинаем IDE Ардуино и открываем `Предпочтительный` диалог:

В Дополнительном менеджере Советов область URL входит в URL для пакета ESP8266, который является:
http://arduino.esp8266.com/stable/package_esp8266com_index.json

Страница 196

Выберите менеджера Советов из меню `Tools> Board`:

Установите поддержку ESP8266:

Это свяжется с Интернетом и загрузит артефакты, необходимые для поддержки ESP8266.
После того, как законченный, в выборах Совета IDE Ардуино, Вы найдете «`Универсальный Модуль ESP8266`»:

Теперь мы готовы начать строить, собирая и бегущие эскизы. Простой и типовой эскиз, который я рекомендую для тестирования:

```
недействительная
  установка ()
  {Serial1.begin
  (115200);
}

недействительная петля () {
```

```
  Serial1.println («Привет! - millis () =» + Последовательность (millis
  ()));
}
```

Когда управляется, петля сообщений появится на продукции UART1, говорящей привет и количестве миллисекунд начиная с последнего ботинка. Так же как что-либо, это утвердит это

окружающая среда была установкой правильно, Вы можете собрать программу, и то развертывание к ESP8266 успешно.

См. также:

- GitHub:esp8266/Arduino
- Ардуино IDE

Советы для работы в окружающей среде Ардуино

Помните, что окружающая среда Ардуино - две вещи. Во-первых, фактическое приложение, которое Вы устанавливаете на своей машине, обеспечивающей IDE Ардуино. Во-вторых, ряд библиотек, которые моделируют доступных фактическому устройству Ардуино, которые нанесены на карту к ESP* возможности. Имея это в виду, вот некоторые полезные советы, которые я нахожу полезными, сочиняя эскизы Ардуино для ESP* окружающая среда.

Инициализируйте глобальные классы в установке ()

В рамках эскиза Ардуино у нас есть предварительно поставляемая функция, `вызванная установка ()`, который называют только однажды во время начальной загрузки ESP8266. В этой функции Вы выполняете функции инициализации времени. В C ++, у нас есть способность создать случаи класса глобально. Например:

```
MyClass myClass (123);

недействительная установка () {
       //Некоторый кодекс здесь
}
```

вместо этого используйте следующее:

```
MyClass *myClass;

недействительная установка () {
       myClass = новый
       MyClass
       (123);//Некоторый
       кодекс здесь
```

```
}
```

Это, конечно, изменяет тип данных Вашей переменной. Это пошло от того, чтобы быть случаем `MyClass` к тому, чтобы быть указателем на случай `MyClass`, что означает, что Вам, возможно, придется изменить другие аспекты Вашей программы …, но причина этого состоит в том, что в первом случае, конструктор для Вашего случая `MyClass` бежал за пределами `установки ()`, и мы не можем сказать, что указывает, что окружающая среда, возможно, была в в том пункте. В рамках `установки ()` кодекс, у нас есть разумное ожидание контекста окружающей среды.

Призыв API Espressif SDK из эскиза

Нет ничего, чтобы препятствовать тому, чтобы Вы призвали API Espressif SDK из своего эскиза. Вы должны включать, любой включает файлы, которые необходимы. Вот пример включения «user_interface.h».

```
экстерн «C» {
   #include «user_interface.h»
}
```

Заметьте заключение в скобки с C ++ конструкция, которая заставляет содержание появляться, как будто это определялось в программе C.

Обработка исключений

Когда исключение обнаружено в кодексе, кодовых остановках. Как правило, мы видим следующее, зарегистрированное к последовательному порту, когда это происходит:

```
 ets 8 января 2013, rst cause:2,
 загружают способ: (1,7) ets 8
 января 2013, rst cause:4,
 загружают способ: (1,7)
WDT перезагрузило
```

К сожалению, это абсолютно ничего не говорит нам о местоположении или причине проблемы.

Файловая система МАГАРЫЧЕЙ ПРИ ТОРГОВОЙ СДЕЛКЕ

Команда mkspiffs

Инструмент был сделан доступным, который строит набор из двух предметов файловой системы «магарычей при торговой сделке» из структуры каталогов, найденной на диске. Команду называют «mkspiffs» и имеет его собственный проект GitHub.

Полный синтаксис команды:

```
mkspiffs {-c <pack_dir> |-l |-i} [-b <число>] [-p <число>] [-s
      <число>] [-] [-версия] [-h] <image_file>
```

Где параметры:

- -c <pack_dir> или - создают <pack_dir> – Создают файл магарычей при торговой сделке изображения из экспертиза справочника, который будет упакован в изображение магарычей при торговой сделке.
- -l или - список – Список содержание существующего файла изображения.
- -i или - визуализируют – Визуализируют изображение магарычей при торговой сделке.
- - b <число> или - блок <число> – размер Файловой системы блокирует размер в байтах.
- - <число> или - p страница <число> – Размер страницы Файловой системы в байтах.
- - <число> или - s размер <число> – Размер Файловой системы изображения в байтах.
- -или - ignore_rest – Игнорируют остающиеся аргументы.

- -версия – Показ информация о версии.
- -h или - помощь – информация об использовании/помощи Показа
- <image_file> – Файл, чтобы содержать (или уже содержит), изображение магарычей при торговой сделке.

См. также:

- GitHub:igrr/mkspiffs

Архитектура поддержки IDE Ардуино

IDE Ардуино для ESP8266 использует понятие о «менеджере правления». Взгляды позади этого состояли в том, который с растущим числом Ардуино связал доски там, все с различными возможностями и

тонкостью, акт добавляющей поддержки нового типа устройства (правление) должен быть сделан универсальным и легче. С этой целью поддержка была добавлена в 1.6.4 и вне для менеджера правления файл JSON. Этот файл описывает содержание нового правления и где «получить» части, необходимые для создавания приложений.

Для ESP8266 Arduino IDE правление файл JSON может быть найден в:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

Если мы исследуем содержание этого файла вместе со спецификацией формата файла пакета IDE Ардуино, мы изучаем много интересных вещей.

Вот файл с 2015-08-03 …

```
{
  «пакеты»:
    [{«Имя»:
    «esp8266»,
    «автогрейдер»: «Сообщество ESP8266»,
    «websiteURL»:
    «https://github.com/esp8266/Arduino»,
    «электронная почта»: «ivan@esp8266.com»,
    «помощь»: {«онлайн»:
      «http://arduino.esp8266.com/versions/1.6.5-947 -
g39819f0/doc/reference.html»
    },

    «платформы»: [{«Имя»:
      «esp8266»,
      «архитектура»:
      «esp8266», «версия»:
      «1.6.5-947-g39819f0»,
      «категория»: «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», «контрольная сумма»:
      «SHA-
256:79a395801a94c77f4855f3921b9cc127d679d961ec207e7fb89f90754123d66a»,
      «размер»: «2295584»,
      «помощь»: {«онлайн»:
        «http://arduino.esp8266.com/versions/1.6.5-94
        7 -
g39819f0/doc/reference.html»
```



```
      },
      «правления»: [
        {
          «имя»: «Универсальный модуль ESP8266»
        },
        {
```

```
        «имя»: «Olimex MOD-WIFI-ESP8266 (-DEV)»
      },
      {
        «имя»: «NodeMCU 0.9 (модуль ESP-12)»
      },
      {
        «имя»: «NodeMCU 1.0 (Модуль ESP-12E)»
      },
      {
        «имя»: «Adafruit YPA ESP8266 (ESP-12)»
      },
      {
        «имя»: «SweetPea ESP-210»
      }
    ],
    «toolsDependen
    cies»: [{
      «поставщик программного блока»: «esp8266»,
      «имя»: «esptool»,
      «версия»: «0.4.5»
    },
    {
      «поставщик
      программного блока»:
      «esp8266», «имя»:
      «эльф xtensa lx106
      gcc», «версия»:
      «1.20.0-26-gb404fb9»
    ''''},


  «инструменты
    »:
    [{«Имя»:
    «esptool»,
    «версия»:
    «0.4.5»,
    «системы»:
    [
      {
          «хозяин»: «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»,
          «контрольная сумма»: «SHA-

256:1b0a7d254e74942d820a09281aa5dc2af1c8314ae5ee1a5abb0653d0580e531b»,
          «размер»: «17408»
      },
      {
          «хозяин»: «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»,
          «контрольная сумма»: «SHA-

256:924d31c64f4bb9f748e70806dafbabb15e5eb80afcdde33715f3ec884be1652d»,
          «размер»: «11359»
      },
```

```
        {
            «хозяин»: «i386-apple-darwin», «URL»:
            «http://arduino.esp8266.com/esptool-0.4.5-1-gfaa5794-
            osx.tar.gz», «archiveFileName»: «esptool 0.4.5 1 gfaa5794
            osx.tar.gz», «контрольная сумма»: «SHA-

256:722142071f6cf4d8c02dea42497a747e06abf583d86137a6a256b7db71dc61f6»,
            «размер»: «20751»
        },
        {
            «хозяин»: «гну x86_64 pc-linux», «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»,
            «контрольная сумма»: «SHA-

256:4ce799e13fbd89f8a8f08a08db77dc3b1362c4486306fe1b3801dee80cfa3203»,
            «размер»: «12789»
        },
        {
            «хозяин»: «гну i686 pc-linux», «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»,
            «контрольная сумма»: «SHA-

256:4aa81b97a470641771cf371e5d470ac92d3b177adbe8263c4aae66e607b67755»,
            «размер»: «12044»
        }
      ]
    },
    {
      «имя»:  «эльф  xtensa
      lx106 gcc», «версия»:
      «1.20.0-26-gb404fb9»,
      «системы»: [
        {
           «хозяин»: «i686-mingw32», «URL»:
           «http://arduino.esp8266.com/win32-xtensa-lx106-elf-
           gb404fb9.tar.gz», «archiveFileName»: «эльф win32 xtensa lx106
           gb404fb9.tar.gz», «контрольная сумма»: «SHA-

56:1561ec85cc58cab35cc48bfdb0d0087809f89c043112a2c36b54251a13bf781f»,
           «размер»: «153807368»
        },
        {
           «хозяин»: «x86_64-apple-darwin», «URL»:
           «http://arduino.esp8266.com/osx-xtensa-lx106-elf-
           gb404fb9-2.tar.gz», «archiveFileName»: «эльф osx xtensa lx106
           gb404fb9 2.tar.gz», «контрольная сумма»: «SHA-

256:0cf150193997bd1355e0f49d3d49711730035257bc1aee1eaaad619e56b9e4e6»,
           «размер»: «35385382»
        },
        {
```

```
            «хозяин»: «i386-apple-darwin», «URL»:
            «http://arduino.esp8266.com/osx-xtensa-lx106-elf-
            gb404fb9-2.tar.gz», «archiveFileName»: «эльф osx xtensa lx106
            gb404fb9 2.tar.gz», «контрольная сумма»: «SHA-

256:0cf150193997bd1355e0f49d3d49711730035257bc1aee1eaaad619e56b9e4e6»,
            «размер»: «35385382»
```



```
         },
         {
            «хозяин»: «гну x86_64 pc-linux», «URL»:
            «http://arduino.esp8266.com/linux64-xtensa-lx106-elf -
gb404fb9.tar.gz», «archiveFileName»: «эльф linux64 xtensa lx106
            gb404fb9.tar.gz», «контрольная сумма»: «SHA-

256:46f057fbd8b320889a26167daf325038912096d09940b2a95489db92431473b7»,
            «размер»: «30262903»
         },
         {
            «хозяин»: «гну i686 pc-linux», «URL»:
            «http://arduino.esp8266.com/linux32-xtensa-lx106-
            elf.tar.gz», «archiveFileName»: «linux32 xtensa lx106
            elf.tar.gz», «контрольная сумма»: «SHA-
256:b24817819f0078fb05895a640e806e0aca9aa96b47b80d2390ac8e2d9ddc955a»,
            «размер»: «32734156»
         }
      ]
      ''''

}
```

Ломая это, у нас есть один пакет в этом файле, у которого есть следующие разделы:

- `имя` – esp8266 – название самого пакета
- `автогрейдер` – Сообщество ESP8266 – Кто поддерживает пакет
- `websiteURL` – Куда пойти, чтобы найти больше об этом пакете
- `электронная почта` – Кто послать по электронной почте, чтобы узнать больше
- `помощь` – Куда пойти для помощи онлайн
- `платформы` – набор платформ, на которых бежит это правление
- `инструменты` – детали необходимых инструментов

Для платформ мы описываем детали каждой платформы … в настоящее время есть только один:

- `имя` – esp8266
- `архитектура` – esp8266

- `версия` – ID вариантов этого пакета/платформы
- `категория` – ESP8266
- `URL` – Где загрузить эту платформу
- `archiveFileName` – название файла

- `контрольная сумма` – мешанина, которая может использоваться против файла, чтобы видеть, вмешалось ли это
  с
- `размер` – размер в байтах файла
- `помощь` – Где прочитать докторов для этой платформы
- `правления` – список правлений, которые связаны с платформой.
- `toolDependencies` – названия дополнительных инструментов/компонентов, которые требуются

Для инструментов это - список инструментов, необходимых для пакета. У каждого инструмента есть следующее:

- `имя` – логическое название инструмента
- `версия` – версия инструмента
- `системы` – список записей, которые определяют, где загрузить инструмент для разнообразия
  платформы включая Windows, Linux и OSx.

С этой информацией и копией файла, Вам необходимо видеть, как некоторые части совмещаются.

Когда пакет установлен, он создан в справочнике:

```
C:\Users\<User>\AppData\Roaming\Arduino15\packages
```

Для нашей истории ESP8266 пакет - esp8266, и следовательно все файлы могут быть найдены в:

```
C:\Users\<User>\AppData\Roaming\Arduino15\packages
```

мы назовем это корнем.

Ниже корня мы найдем два справочника:

- оборудование

- инструменты

Справочник `инструментов` содержит корень наших инструментов, необходимых для выполнения …, это компилятор C и инструмент закачки.

Папка `аппаратных средств` содержит

остальную часть нашей информации.

Конкретно следующие папки:

- `загрузчик операционной системы` – тайна...
- `ядра` – Основной заголовок и исходные файлы, предоставляющие код всегда, связывались с нашим
  эскизы. Это - основной набор оберток для библиотек Ардуино.
- `инструменты` – Espressif SDK

Страница 207

- `библиотеки` – библиотеки по умолчанию для нашего пакета
- `варианты` – Заголовочные файлы, которые отличаются вариантом отобранного правления

И следующие файлы:

- `правления`
- `платформа`
- `программисты`

В IDE Ардуино мы можем включить многословные параметры настройки, который приводит к дополнительным деталям, зарегистрированным во время компиляции или закачки. От них мы можем узнать больше о том, что происходит.

Если мы исследуем типичное заявление компиляции, мы находим следующее.

```
эльф
xtensa
lx106 gcc-
D __ ets
__-
DICACHE_FL
ASH-U __
STRICT_ANS
I __-
Itools/sdk
//включает
```

-c
- g
- x
assembler-
with-cpp-
MMD-DF_CPU=
80000000L-
DARDUINO=
10605
- DARDUINO_ESP8266_ESP01
-
DARDUINO_ARC
H_ESP8266-
DESP8266-
Icores
\esp8266
- Ivariants\generic\cores
\esp8266\cont. S-o
продолжение следует S.o
эльф
xtensa
lx106 gcc-
D __ ets
__-
DICACHE_FL
ASH-U __
STRICT_ANS
I __-
Itools/sdk
//включает
-c
-
P
o
т
-
g
- Wpointer-арифметика-
Wno-implicit-function-
declaration-Wl, - EL

- fno-
inline-
functions-
nostdlib-
mlongcalls

- mtext-
section-
literals-
falign-
functions=4-
MMD
- std=gnu99

Страница 208

- DF_CPU=80000000L

```
- DARDUINO=
10605-
darduino_esp8266
_esp01-
darduino_arch_es
p8266-desp8266-
icores\esp8266-
ivariants
\generic cores
\esp8266\cont_
util.c-o
cont_util.c.o
```

Содержание основного справочника - артефакты, которые связаны с Вашими эскизами Ардуино.

магарычи при торговой сделке	
abi.cpp	
Arduino.h	Предварительные выборы включают файл для заявлений.
binary.h	Двойные определения для диапазона 0-255 до 8 битов.
cbuf.h	Круглый буфер.
Client.h	
cont.h	
продолжение следует. S	
cont_util.c	
core_esp8266 _eboot_command.c	
core_esp8266_flash_utils.c	
core_esp8266_i2s.c	
core_esp8266_main.cpp	Главная точка входа в применение 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	

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

Когда мы смотрим на то, как применение загружено, мы видим команду, подобную следующему:

```
esptool.exe-vv - CD ck-cb 115200 - CP COM11 - приблизительно 0x00000-cf
ESP_I2CScanner.cpp.bin
```

Строительство ESP приложения Ардуино, используя Затмение IDE

Теперь наши головы действительно собираются повредить … нет никакого легкого способа пройти через этот …, но история важна, и результаты большие.

До сих пор мы видели, что можем построить программы ESP, используя компилятор C и Espressif SDK. Мы также видели, что можем построить эти программы в окружающей среде Затмения … также против Espressif SDK. Мы только что исследовали понятие технических заданий на строительство, используя IDE Ардуино, который предоставляет отображения многим библиотекам Ардуино, осуществленным для ESP. Теперь мы собираемся возвратиться к использованию Затмения, но на этот раз как альтернатива IDE Ардуино, но все еще пользованию библиотеками Ардуино, чтобы построить «Ардуино приправленные» программы ESP.

Ключ к этой истории - превосходный Общедоступный набор инструментов Затмения для здания Ардуино, найденного здесь:

http://eclipse.baeyens.it/index. shtml

Этот набор программных расширений к структуре Затмения усиливает существующую окружающую среду Ардуино, такую как та, которую мы только что построили, который включает поддержку ESP. Программные расширения опрашивают установку IDE Ардуино и обеспечивают строение и инструменты редактирования, используемые там.

Прежде, чем идти дальше, жизненно важно, чтобы Вы получили ESP Ардуино IDE, работающий, следуя всем инструкциям, необходимым, чтобы построить решения, используя ту окружающую среду. Это - предпосылка для получения работы окружающей среды Затмения.

У нас есть два варианта для получения работы окружающей среды Затмения. Первое должно загрузить полностью подготовленную окружающую среду Затмения, которая включает средства разработки C **и** штепсель-ins для поддержки Ардуино. Это - самый легкий … однако, также вероятно, что у Вас есть существующая структура Затмения, уже установил это, Вы можете хотеть снова использовать или простираться. Затмение предназначается, чтобы быть расширяемой окружающей средой, которая

служит основой, в которую дополнительные программные расширения могут быть добавлены по мере необходимости. В том образце мы можем загрузить последнюю структуру Затмения (Марс) и затем добавить в соответствующих программных расширениях.

Вот пример готовящихся, используя предварительно построенную загрузку Затмения.

Сначала загрузите ток, в течение ночи строят … и извлекают его содержание. Обратите внимание, что автогрейдер распределяет файлы в формате tar.gz. Я использую7 - Почтовый индекс, чтобы развернуть. Размер загрузки составляет приблизительно 170 мегабайтов, так удостоверьтесь, что Вы загружаете раньше, чем позже. Удостоверьтесь, что Вы загружаете правильную версию окружающей среды Затмения, которая соответствует версии Явы, которую Вы установили. Например, если Вам установили 32-битную Яву, не загружайте 64-битную версию Затмения. Если Вы сделаете и попытаетесь начать Затмение, то Вы получите ошибки, в которые Вы должны будете вырыть только, чтобы найти похороненным в регистрациях, что есть несовместимость. Если Вы действительно совершаете ошибку, сообщение, которое Вы получаете, могло бы быть похожим:

Страница 211

Как Вы видите, не с готовностью очевидно, что пошло не так, как надо.

Когда Вы будете готовы начать, запустите программу, названную «eclipseArduinoIDE».

Если все подходило, мы будем видеть следующее:

Страница 212

Теперь пора настроить окружающую среду Затмения для узнать о нашей среде Ардуино. Рецепты, которые следуют, используются, чтобы преодолеть некоторые ошибки, так может изменяться со временем …

Во-первых, мы должны сказать Затмение, где оно может найти нашу среду Ардуино.

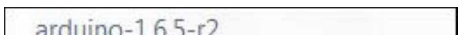
Откройте `Предпочтения` и выберите `Ардуино`:

Страница 213

Мы должны изменить некоторые настройки.

Для пути IDE Ардуино укажите на корневой каталог, где Ваш IDE Ардуино установлен. Это должно быть справочником, который содержит следующее:

Страница 214



Следующая часть немного более хитра. Мы должны поставлять ценности и для «Частного Пути к библиотеке» и для «Частного пути аппаратных средств». Эти справочники - справочники для пакета ESP Ардуино а **НЕ** родного Ардуино. Вы найдете их в следующих справочниках:

```
C:\Users\<Your
Userid> \AppData\Roaming\Arduino15\packages\esp8266\hardware\esp8266 \<Ваш
Версия> \libraries
```

```
C:\Users\<Your Userid>AppData\Roaming\Arduino15\packages\esp8266\hardware
```

Я также рекомендую, чтобы Вы изменились Ваш, «Строят перед закачкой», чтобы быть «Да» в этом пункте. После входа Ваш экран мог бы быть похожим.

Применяя изменения Вы сделали здесь, Вы можете видеть соблюдающее предупреждение несколько раз:

Нажмите «Yes», чтобы проигнорировать предупреждения и идти дальше.

Затем, мы должны добавить «*.ino» как новый C ++ тип файла …, мы делаем это снова в предпочтениях:

Страница 216

В получающемся диалоге добавьте следующее:

Мы рядом. Теперь у нас есть несколько заключительных одноразовых изменений, чтобы сделать. Найдите файл под названием «`platform.txt`», который расположен в:

```
C:\Users\<Your
Userid> \AppData\Roaming\Arduino15\packages\esp8266\hardware\esp8266
\<Версия> \platform
.txt
```

Отредактируйте этот файл со своим редактором текста и найдите линию, которая читает:

```
tools.esptool.upload.pattern = «{путь} / {cmd}» {upload.verbose} - CD
{upload.resetmethod}-cb {upload.speed} - CP «{serial.port}» -
приблизительно 0x00000-cf «{build.path} / {строят project_name} .bin»
```

и измените его, чтобы быть:

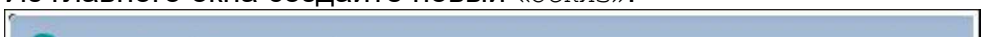
```
tools.esptool.upload.pattern = «{путь} / {cmd}»-vv - CD
{upload.resetmethod}-cb {upload.speed} - CP «{serial.port}» -
приблизительно 0x00000-cf «{build.path} / {строят
project_name} .bin»
```

(Это, мы изменяемся «`{upload.verbose}`» на «`-vv`»),

Сохраните файл.

Мы теперь готовы построить использование наша среда.

Из главного окна создайте новый «Эскиз».



Обратите внимание, что мы можем также сделать это через стандартное Затмение> Новый Проект

Первая страница волшебника создания проекта - имя, которое мы хотим дать нашему проекту:

Затем мы поставляем некоторые основные параметры настройки. Не торопитесь по ним. Тем, который будет, вероятно, отличаться для Вас, является «Порт»: то, которое является последовательным портом, раньше высвечивало Ваше устройство:

При завершении волшебника (принимающий Вас взял остальную часть

вариантов в качестве дефолтов) … у Вас будет проект, который похож:

Страница 219

Отредактируйте `Test1.ino C` ++ исходный файл и добавьте свой кодекс.

Здесь Вы будете видеть свою выплату. Вы теперь редактируете свой источник в профессиональном C/C ++ редактор, который является частью Затмения. Это включает вход, помогают, проверка синтаксиса и выдвижение на первый план (и больше).

Прежде чем Вы сможете собрать свой проект, Вы должны изменить проект определенные параметры настройки, чтобы сказать проект, где найти Ваш `делать` программу. В моей среде я использую, `«mingw32-`

делают». Вы видите, где внести изменения в следующем скрин-шоте. Примечание: Должен быть лучший путь, чем этот …, но я хотел вывести этот рецепт, а не поддержать всех, в то время как я переделал этот маленький самородок:

Страница 220

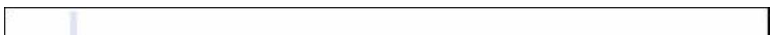
И наконец, мы можем собрать нашу программу.

Нажмите символ «Verify»:

Теперь компиляция произойдет. Поскольку Ваши первые строят из этого проекта, **весь** исходный код всех библиотек будет построен. Будущее строит, просто соберет то, что изменилось. На моей машине строение заняло 51 секунду:

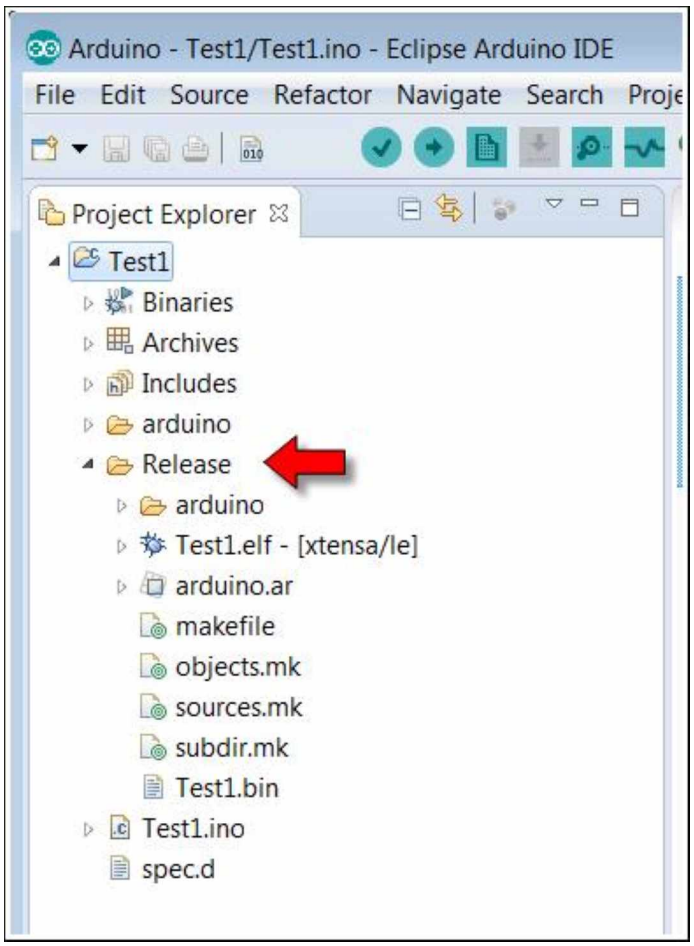
Страница 221

Однако, когда я теперь редактирую свой `файл (Test1.ino)` проекта и повторно собираю, восстанавливание только занимает 4 секунды только в качестве файлов, которые изменили потребность, которая будет повторно собрана.



После строения может быть найден новый справочник под названием `«Выпуск»`, который содержит все артефакты, которые были собраны. Если Вы хотите вызвать восстанавливание всех, просто удалите

«Выпуск».


Arduino - Test1/Test1.ino - Eclipse Arduino IDE
File Edit Source Refactor Navigate Search Proje
Project Explorer
Test1
Binaries
Archives
Includes
arduino
Release
arduino
Test1.elf - [xtensa/le]
arduino.ar
makefile
objects.mk
sources.mk
subdir.mk
Test1.bin
Test1.ino
spec.d

Теперь, когда Вы имеете, строят программу, Вы можете загрузить ее с символом закачки:

Это загрузит исполняемый файл. К сожалению, нет очень, чтобы видеть, в то время как закачка происходит, так расслабьтесь и быть терпеливыми. Если у Вас будет USB → UART соединитель, приложенный к UART1 ESP8266, то Вы будете видеть, что закачка прогрессирует …, но это не важно (хотя я рекомендую это).

И …, который является этим. Вы теперь создаете приложения ESP8266, пользующиеся библиотеками Ардуино в окружающей среде Затмения.

Если мы хотим добавить одну из снабженных библиотек, мы можем выбрать библиотеку, чтобы включать с:

```
Ардуино> Добавляет библиотеку к отобранному проекту
```

Оттуда, мы можем выбрать одну из библиотек:

Мы можем добавить внешние библиотеки, которые существуют исходные файлы в файловой системе. Из меню Ардуино выберите, «Добавьте исходная папка к отобранному проекту».

Диалог будет представлен, где можно выбрать справочник, который будет

включен в строение.

Страница 224

'

После того, как добавленный это становится связанным справочником в проекте, и содержание справочника будет собрано и связано.

Примечание: Предыдущий рецепт был основан на 1.6.5-r2 IDE Ардуино, стабильная версия Ардуино для ESP8266 с 2015-07-23 и эти 2015-08-05 строит из Затмения Ардуино.

См. также:

- GitHub:esp8266/Arduino
- Ардуино – Затмение

Как ранее упомянуто, нет никакого вопроса, что IDE Ардуино намного более дружественный и потребляемый что профессиональная окружающая среда Затмения для людей, плохо знакомых с областью. Кажется, нет ничего, что нельзя построить использование IDE Ардуино, который означал бы, что нужно было бы переключиться на Затмение. Итак, почему тогда можно было бы когда-либо рассматривать использование Затмения?

Страница 225

Есть компромисс между простотой использования и богатством функции. Например, Затмение построило в помощи синтаксиса, проверке ошибок, кодовых перекрестных ссылках, рефакторинге и многое другое. Ни одна из этих вещей не «важна», но любой из них, как могут полагать, делает программную работу легче если и, когда необходимый. Если я должен переименовать переменную в Ардуино IDE, я должен вручную найти и заменить каждое возникновение. В Затмении я могу ре - учитывать переменную, используя встроенного волшебника, и IDE делает работу для меня. Как другой пример, если бы я не могу помнить синтаксис за метод в Ардуино IDE, я пошел бы в сеть и искал бы его, в то время как в Затмении я мог ввести имя метода и толпиться, моя мышь по нему и набор инструментов покажут мне возможные варианты для параметров.

Примечания по использованию Затмения пакет Ардуино

- **Не** создавайте проекты Затмения, у которых есть места на их имена. Это путает компилятор.
- Если Вы собираете на многоядерной машине, Вы можете заставить компиляции исходных файлов прогрессировать в параллельном использовании, добавив «`-рабочие места <цифровой>`» параметр к Вашему `делать` команду. Это вызовет, делают, чтобы выполнить некоторое количество рабочих мест параллельно. Например, если у Вас есть 4 основных машины, устанавливание `цифры` быть 4 могло бы быть хорошим началом. Этот флаг работы с GNU `делает` инструмент.

• Время от времени мы хотим написать эскизы, которые работают и над реальным Ардуино и над ESP8266, но кодекс должен немного отличаться. Мы можем включать кодекс для обоих

Страница 226

архитектура при помощи `#defines` под названием `ARDUINO_ARCH_ESP8266` и/или `ESP8266`. Используя их мы можем `#if/ #endif` разделы на основе архитектуры, против которой мы собираем.

• С 2015-08-08, пытаясь использовать родную функцию SDK в окружающей среде Затмения Ардуино не работает**, если** Ваши исходные файлы - «`*.ino`». Кажется, что «`user_include.h`» включен автоматически с C ++ имя функции, корежащее в действительности.

• Я рекомендую переименовать любые «`*.ino`» файлы в Вашем проекте к «`*.cpp`».

Библиотеки ESP Ардуино

Библиотека WiFi

У Ардуино есть библиотека WiFi для использования с ее щитом WiFi. Библиотека с подобным интерфейсом была снабжена для окружающей среды Ардуино для ESP8266.

Чтобы использовать библиотеку ESP8266 WiFi, Вы должны включать ее удар головой:

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

Чтобы быть станцией и соединиться с точкой доступа, выполните требование к `WiFi.begin (ssid, пароль)`. Теперь мы должны, чтобы

получить голоса `WiFi.status ()`. Когда это возвращает WL_CONNECTED, тогда мы связаны с сетью.

Чтобы настроить точку доступа, мы назвали бы `WiFi.softAP ()` поставкой информация о пароле и ssid.

Вот пример того, что мы соединялись как станция:

```
WiFi.mode
(WIFI_STA);
WiFi.begin
(SSID, ПАРОЛЬ);
если (WiFi.waitForConnectResult ()! =
       WL_CONNECTED) {(«Неудавшийся»)
       Serial1.println;
       вернуть;
}
WiFi.printDiag(Serial1);
//Мы теперь связаны как станция
```

См. также:

- WiFiClient
- WiFiServer
- Библиотека Arduino WiFi

WiFi.begin

Начните связь WiFi как станцию.

```
интервал начинается (
   случайная работа константы *ssid,
```



```
   случайная работа
   константы
   *passPhrase=NULL,
   int32_t channel=0,
   uint8_t bssid[6]=NULL)

интерв
   ал
   нач
   ина
   етс
   я
   (сл
   уча
   йна
   я
   раб
   ота
   *ss
   id,
   случайная
   работа
   *passPhrase=N
```

```
ULL, int32_t
channel=0,
uint8_t
bssid[6]=NULL
)
```

Начните связь WiFi как станция. `ssid` параметр обязателен, но другие можно оставить как дефолт. Возвращаемое значение - наш текущий статус связи.

WiFi.beingSmartC
onfig bool
beginSmartConfig
()

WiFi.beginWPSC
onfig bool
beginWPSConfig
()

WiFi. BSSID

Восстановите текущий BSSID.

```
uint8_t BSSID ()
uint8_t *BSSID (uint8_t networkItem)
```

Восстановите текущий BSSID.

WiFi. BSSIDstr

Восстановите текущий BSSID как представление последовательности.

```
Натяните BSSIDstr ()
Натяните BSSIDstr (uint8_t networkItem)
```

Восстановите текущий BSSID как представление последовательности.

Канал WiFi

Восстановите текущий канал.

```
канал int32_t ()
канал int32_t (uint8_t networkItem)
```

Восстановите текущий канал.

WiFi.config

Установите конфигурацию связи WiFi.

```
недействительная конфигурация (IPAddress local_ip, ворота IPAddress,
подсеть IPAddress)
недействительная конфигурация (IPAddress local_ip, ворота IPAddress, подсеть
IPAddress, IPAddress dns)
```

Установите конфигурацию WiFi, используя статические параметры. Это отключает DHCP.

WiFi.disconnect

Разъедините от точки доступа.

```
международное разъединение (bool wifiOff = ложный)
```

Разъедините от текущей точки доступа.

WiFi.encryptionType

Возвратите тип шифрования просмотренной точки доступа WiFi.

```
uint8_t encryptionType (uint8_t networkItem)
```

Возвратите тип шифрования просмотренной точки

доступа WiFi. Ценности - один из:

- ENC_TYPE_NONE
- ENC_TYPE_WEP
- ENC_TYPE_TKIP
- ENC_TYPE_CCMP
- ENC_TYPE_AUTO

WiFi.gatewayIP

Получите IP-адрес станционных ворот.

```
IPAddress gatewayIP ()
```

Восстановите IP-адрес станционных ворот.

WiFi.getNetworkInfo

Восстановите все подробности указанного просмотрели networkItem.

```
bool getNetworkInfo (uint8_t
```

```
networkItem,
Последовательность &ssid,
uint8_t
&encryptionType,
int32_t &RSSI,
uint8_t *&BSSID,
```



```
uint32_t
&channel,
bool
&isHidden
)
```

Восстановите все подробности указанного просмотрели networkItem.

WiFi.hostByName

Поиск хозяин именем.

```
интервал hostByName (случайная работа константы *имя хоста, IPAddress
&result)
```

Ищите хозяина по имени и получите его IP-адрес. Эта функция возвращается 1 на успехе и 0 на неудаче.

WiFi.hostname

Восстановите и установите имя хоста, используемое этой станцией.

```
Натяните имя хоста ()
имя хоста bool (случайная работа *имя хоста)
имя хоста bool (случайная

работа константы *имя хоста)

bool имя хоста (Натягивают

имя хоста),
```

WiFi.isHidden

Определите, сигнализируется ли просмотренный сетевой пункт, как скрытый.

```
bool isHiddem (uint8_t networkItem)
```

Определите, сигнализируется ли просмотренный сетевой пункт, как скрытый.

WiFi.localIP

Получите станционный IP-адрес.

```
IPAddress localIP ()
```

Получите IP-адрес для станции. Есть отдельный IP-адрес, если ESP - точка доступа.

См. также:

- WiFi.softAPIP

WiFi.macAddress

Доберитесь станция соединяют MAC-адрес.

```
uint_t *macAddress
(uint8_t *Mac)
Последовательность
macAddress ()
```

Доберитесь станция соединяют MAC-адрес.

WiFi.mode

Установите рабочий режим.

```
недействительный способ (способ WiFiMode)
```

Установите рабочий режим WiFi. Это - один из:

- `ВИФИ_ОФФ` – выключает WiFi
- `WIFI_STA` – Быть станцией WiFi
- `WIFI_AP` – Быть точкой доступа WiFi
- `WIFI_AP_STA` – Быть и станцией WiFi и точкой доступа WiFi

См. также:

- Определение рабочего режима

WiFi.printDiag

Зарегистрируйте состояние связи WiFi.

```
пустота printDiag (Печать &dest)
```

Зарегистрируйте состояние связи WiFi. Мы можем пройти или в Последовательном или в Serial1 как аргумент, чтобы зарегистрировать данные к Последовательному порту. Пример продукции как показано следующий:

```
Спос
```

```
об:
СТАН
ЦИЯ
спос
об
PHY:
N
Кана
л: 7
иден
тифи
като
ров
AP:
0
Стат
усов
: 5
Автомобиль
соединяется:
0 SSID (7):
yourSSID
Пароль (8):
yourPassword BSSID
установил: 0
```

Обратите внимание, что стоимость статуса - результат `wifi_station_get_connect_status ()` требование.

WiFi. RSSI

Восстановите RSSI (Полученный Индикатор Силы Сигнала) ценность просмотренного сетевого пункта.

```
int32_t RSSI (uint8_t networkItem)
```

Восстановите ценность RSSI просмотренного сетевого пункта.

WiFi.scanComplete

Определите статус предыдущего запроса просмотра.


```
int8_t scanComplete ()
```

Если результат> = 0 тогда, это - количество найденных точек доступа WiFi. Иначе стоимость - меньше чем 0, и кодексы:

- `SCAN_RUNNING` – просмотр в настоящее время происходит.
- `SCAN_FAILD` – просмотр потерпел неудачу.

См. также:

- WiFi.scanNetworks

- WiFi.scanDelete

WiFi.scanDelete

Удалите результаты предыдущего просмотра.

```
пустота scanDelete ()
```

Удалите результаты предыдущего просмотра. Просьба просмотреть сеть приводит к распределению памяти. Это требование выпускает ту память.

См. также:

- WiFi.scanComplete
- WiFi.scanNetworks

WiFi.scanNetworks

Просмотрите точки доступа в окружающей среде.

```
int8_t scanNetworks (bool async = ложный)
```

Просмотрите точки доступа в окружающей среде. Мы можем или выполнить это синхронное или асинхронное. На синхронном требовании результат - количество найденных точек доступа.

См. также:

- WiFi.scanComplete
- WiFi.scanDelete

WiFi.smartConfi gDone bool smartConfigDone

()

WiFi.softAP

Установка точка доступа.

```
      пустота softAP
    (случайная работа
     константы *ssid)
      пустота softAP
    (случайная работа
     константы *ssid,
     случайная работа
   константы *пароль,
      интервал channel=1,
      интервал ssid_hidden=0)
```

`ssid` используется, чтобы рекламировать нашу сеть. `Пароль` - пароль, который должна поставлять станция, чтобы быть уполномоченной получить доступ.

WiFi.softAPConfig

пустота softAPConfig (IPAddress local_ip, ворота IPAddress, подсеть IPAddress)

WiFi.softAPdisconnect

интервал softAPdisconnect (bool wifiOff=false)

WiFi.softAPmacAddress

Получите MAC-адрес интерфейса точки доступа.

```
uint8_t *softAPmacAddress (uint8_t *Mac)
```

Получите MAC-адрес интерфейса точки доступа.

WiFi.softAPIP

Получите IP-адрес интерфейса точки доступа.

```
IPAddress softAPIP ()
```

Возвратите IP-адрес интерфейса точки доступа. Есть отдельный IP для станции.

См. также:

- WiFi.localIP

WiFi. SSID

Восстановите SSID.

```
случайная работа *SSID ()
случайная работа константы *SSID (uint8_t networkItem)
```

Здесь мы восстанавливаем SSID текущей станции или SSID отсканированного сетевого id.

WiFi.status

Восстановите текущий статус WiFi.

```
статус wl_status_t ()
```

Статус возвратился, будет один из:

- WL_IDLE_STATUS (0)

- WL_NO_SSID_AVAIL (1)
- WL_SCAN_COMPLETED (2)
- WL_CONNECTED (3)
- WL_CONNECT_FAILED (4)
- WL_CONNECTION_LOST (5)
- WL_DISCONNECTED (6)

Пустота
WiFi.stopSmar
tConfig
stopSmartCo
nfig ()

WiFi.subnetMask
IPAddress subnetMask ()

WiFi.waitForConnectResult
Ждите, пока связь WiFi не была сформирована или подведена.

```
uint8_t waitForConnectResult ()
```

Если мы - станция, то блокируем ожидание нас, чтобы стать связанными или неудавшимися. Код возврата - статус. А именно, эта функция наблюдает статус, чтобы видеть, когда это становится чем-то другим, чем `WL_DISCONNECTED`. Возможно, более положительная форма этой функции была бы:

```
в то время как (WiFi.status
       ()! = WL_CONNECTED)
       {задержка (500);
       Serial.print (». «);
}
```

WiFiClient

Эта библиотека предоставляет соединения по протоколу TCP партнеру. Отдельный класс обеспечивает коммуникации UDP. Чтобы пользоваться этой библиотекой, Вы должны включать

«ESP8266WiFi.h».

Мы создаем случай этого класса и затем соединяемся с партнером, использующим `соединение ()` метод.

WiFiClient

WiFiClient.available

Возвратите объем данных, доступный, чтобы быть прочитанными
l15summe


```
доступный интервал ()
```

Возвратите объем данных, доступный, чтобы быть прочитанными.

WiFiClient.connect

Соединитесь с данным хозяином в данном порту, используя TCP.

```
интервал соединяется (случайная
работа константы* хозяин, uint16_t
порт), интервал соединяется (IP
IPAddress, uint16_t порт)
```

Соединитесь с данным хозяином в данном порту, используя TCP. Эта функция возвращается `0` на неудаче.

WiFiClient.connected

Определите, связаны ли мы с партнером.

```
uint8_t соединился ()
```

Возвратитесь верный, если связано и ложный иначе.

Пото
к
пусто
ты
WiFiC
lient.fl
ush ()

WiFiClient.g
etNoDelay
bool
getNoDelay
()

Быст
рый
взгля
д
интер
вала
WiFiC
lient.p
eek ()

WiFiClient.read
Прочитайте данные от партнера.

```
интервал читал ()
интервал читал (uint8_t *buf, size_t размер)
```

Прочитайте данные от партнера. Эти функции читают или единственный байт или последовательность байтов от партнера.

WiFiClient.remoteIP
Восстановите отдаленный IP-адрес связи.

```
IPAddress remoteIP ()
```


Восстановите отдаленный IP-адрес связи.

WiFiClient.remotePort
Возвратите отдаленный порт, используемый в существующей связи.

```
uint16_t remotePort ()
```

Возвратите отдаленный порт, используемый в существующей связи.

WiFiClient.setLocalPortStart

Установите начальный порт для распределения местных портов для связей.

```
пустота setLocalPortStart (uint16_t порт)
```

Установите начальный порт для распределения местных портов для связей.

WiFiClient.setNoDelay

пустота setNoDelay (bool nodelay)

Статус WiFiClient.status uint8_t ()

WiFiClient.stop

Разъедините клиента.

```
недействительная остановка ()
```

Разъедините клиента.

WiFiClient.stopAll

Остановите все связи, сформированные этим клиентом WiFi.

```
пустота stopAll ()
```

WiFiClient.write

Напишите данные партнеру.

```
size_t пишут (uint8_t b)
size_t пишут (константа uint8_t *buf, size_t размер) size_t пишут
(источник T&, size_t unitSize);
```


Напишите данные партнеру. Первая функция пишет один байт, в то время как вторая функция пишет множество знаков.

WiFiServer

WiFiServer

Создайте случай Сервера, слушающего на снабженном порте.

```
WiFiServer (uint16_t порт)
```

Создайте случай Сервера, слушающего на снабженном порте. Интересный, кажется, что, как только мы упаковываем случай сервера в ESP8266, нет никакого способа остановить его управление.

WiFiServer.available

Восстановите объект WiFiClient, который может использоваться для коммуникаций.

```
Доступный WiFiClient (байт* статус)
```

Восстановите

соответствующий

WiFiClient.См. также:

- WiFiClient

WiFiServer.begin

Начните прислушиваться к поступающим связям.

```
пустота начинается ()
```

Начните прислушиваться к поступающим связям. Пока этот метод не называют, ESP8266 не принимает поступающие связи. Интересно, когда-то названный, нет никакого очевидного способа прекратить слушать. Порт, используемый для поступающих связей, является тем, поставляемым, когда объект WiFiServer был построен.

WiFiServer.getNoDelay

WiFiServer.hasClient

Возвратитесь верный, если

нам соединили клиента. bool

hasClient ()

WiFiServer.setNoDelay

WiFiServer.status

WiFiServer.write

ПРЕДУПРЕЖДЕНИЕ!! Этот метод не осуществлен.

```
size_t пишут (uint8_t b)
size_t пишут (константа uint8_t *буфер, size_t размер)
```

Хотя существующий в интерфейсе, этот метод еще не осуществлен.

IPAddress

Представление IPAddress. У этого класса есть некоторый

оператор, отвергает: [я] – Получаю ith байт адреса. Я должен

быть 0-3.

ESP8266WebServer

Класс ESP8266WebServer обеспечивает основное внедрение сервера HTTP. Это - программное обеспечение, которое отвечает на запросы браузера. Чтобы использовать этот класс, мы создаем случай объекта ESP8266WebServer, определяющего номер порта TCP, на котором он послушает. Порт по умолчанию для браузеров - порт 80, таким образом, это - хороший выбор.

Как только объект был создан, мы определяем одну или несколько функций обратного вызова, которые будут призваны, когда связь браузера будет получена. Функция, `к которой обращаются ()`, используется, чтобы зарегистрировать их. Эти функции обратного вызова включены на пути URL, который требует браузер. Например, если наш ESP8266 бежит в IP-адресе 192.168.1.2, и URL браузера: `http://192.168.1.2/index.html`

Путем URL будет «`/index.html`».

Если бы мы хотим послать ответ на запрос в том URL, мы зарегистрировали бы функцию обратного вызова, используя тот путь в качестве ключа.

Например:

```
myServer.on («/index.html», myFunction);
```

где `myFunction` - функция C с подписью:

```
пустота myFunction ()
```

Функция обратного вызова, когда ее называют, может использовать

объект ESP8266WebServer выполнить `посылание ()` требование метода послать ответ.

Страница 238

Если запрос браузера прибывает для пути URL, который явно не обработан, требование к функции обратного вызова, зарегистрированной в `onNotFound ()`, метод призван. Это может служить вместилищем для обработки.

Когда URL содержит свойства вопроса формы «x=y», число, имена и ценности этих свойств доступны в `args ()`, `argName ()` и `аргумент ()` функции. Обратите внимание, что кодирование URL в настоящее время не поддержано так, данные еще не могут содержать характеры инвалида URL.

Когда запрос от браузера получен, каждый хочет передать обратно ответ и способ достигнуть, который является посредством просьбы `посылания ()` методом. Это берет код ответа к браузеру (200 для хорошо), тип кодирования ПАНТОМИМЫ и полезный груз данных как параметры.

Когда Вы отправляете запрос от браузера до веб-сервера, ожидаете, что дополнительный HTTP ПОЛУЧАЕТ запрос, который хочет восстановить файл, названный «/favicon.ico», который используется, чтобы определить символ, который представляет получаемый доступ веб-сайт. Чтобы обращаться с этим, мы могли бы хотеть добавить функцию укладчика, которая смотрит следующим образом:

```
webServer.on (»/favicon.ico», [] ()
       {webServer.send (404,
       «текст/равнина», «»);
});
```

Это регистрирует укладчика для файла символа и передает 404 обратно (не найденный) ответ на браузер. Заметьте использование действующей анонимной функции в C ++. Вашим выбором использовать этот стиль кодирования является Ваше собственное. Лично, я предпочитаю явно объявленные функции.

Вот пример приложения веб-сервера:

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

```
<WiFiClient.h>
#include
<ESP8266WebServer.
h>
случайная работа константы *ssid = «mySsid»;
случайная работа константы *пароль = «myPassword +»;

ESP8266WebServer webServer (80);

пустота logDetails ();

пустота testHandler ()
  {Serial1.println
  («testHandler»);
  logDetails ();
  webServer.send (200, «текст/равнина», «Вот наш ответ»: +
  Последовательность (millis ()));
}

пустота notFoundHandler ()
  {Serial1.println («Не
  Найденный Укладчик»);
  logDetails ();
}
```



```
Натяните methodToString (метод
  HTTPMethod) {выключатель (метод) {
              случай
           HTTP_GET:
         возвратитесь
       «ДОБИРАЮТСЯ»;
              случай
          HTTP_POST:
      возвратите
    «ПОЧТУ»; случай
    HTTP_PUT:
      возвратитесь
    «ПОМЕЩЕННЫЙ»;
    случай
    HTTP_PATCH:
            возвратите
     «УЧАСТОК»; случай
          HTTP_DELETE:
          возвратитесь
            «УДАЛЯЮТ»;
  }
  возвратитесь «Неизвестный»;
}

пустота logDetails () {
  Serial1.println («URL»: + webServer.uri ());
  Serial1.println («Метод HTTP по запросу был»: +
methodToString (webServer.method ()));
```

```
//     Печать, сколько свойств мы получили и затем печатаем их имена
//     и ценности.
  Serial1.println («Количество свойств вопроса»: + Последовательность
  (webServer.args ())); интервал i;
  для (i=0; я <webServer.args (); я ++) {
    Serial1.println (» - «+ webServer.argName (i) +» = «+ webServer.arg (i));
  }
}

недействительная установка ()
  {Serial1.begin (115200);
  Serial1.println
  («Начинающийся...»); WiFi.begin
  (ssid, пароль);
  //Ждите связи
  в то время как (WiFi.status ()! =
    WL_CONNECTED) {задержка (500);
    Serial1.print (». «);
  }
  Serial1.println (»»); Serial1.print
  («Связанный с»);
  Serial1.println(ssid); Serial1.print
  («IP-адрес: «); Serial1.println
  (WiFi.localIP ()); webServer.on (» /
  проверяют», testHandler);
  webServer.on (»/favicon.ico», [] () {
    webServer.send (404, «текст/равнина»,
  «»);});
  webServer.onNotFound (notFoundHandler);
  webServer.begin ();
  Serial1.println («Мы начали веб-сервер»);
}
```



```
недействительная
  петля ()
  {webServer.handle
  Client ();
}
```

См. также:

- Википедия:Favicon
- ESP8266WebServer.on
- ESP8266WebServer.send

ESP8266WebServer

Постройте случай объекта WebServer.

```
ESP8266WebServer:: ESP8266WebServer (международный порт)
```

Постройте случай класса. Поставляемый `номер порта` является портом, на который послушают для поступающих запросов браузера.

ESP8266WebServer.arg

Восстановите ценность аргумента.

```
Аргумент последовательности (международный индекс)
```

Поскольку собственность передала последовательность вопроса, здесь мы можем восстановить соответствующую стоимость.

ESP8266WebServer.argName

Восстановите название имени аргумента.

```
Натяните argName (международный индекс)
```

Поскольку собственность передала последовательность вопроса, здесь мы можем восстановить соответствующее имя.

ESP8266WebServer.args

Восстановите количество свойств передало последовательность вопроса, поставляемую вопросом браузера.

```
интервал args ()
```

ESP8266WebServer.begin

Начните прислушиваться к поступающим связям браузера.

```
пустота начинается ()
```

ESP8266WebServer.client

Клиент WiFiClient ()

ESP8266WebServer.handleClient

Обращайтесь с клиентом (браузер) запрос.

```
пустота handleClient ()
```

Это - функция, которая должна периодически вызываться, чтобы обработать поступающие запросы браузера. Например, это - функция, которая помещена в тело петли ().

ESP8266WebServer.hasArg

Возвратитесь верный, если названная собственность поставлялась.

```
bool hasArg (случайная работа константы* имя)
```

Параметр имени - название собственности, которая, возможно, поставлялась как собственность в последовательности вопроса.

ESP8266WebServer.method

Восстановите метод, поставляемый оригинальным запросом браузера.

```
Метод HTTPMethod ()
```

`HTTPMethod` может быть одним из:

- HTTP_GET
- HTTP_POST
- HTTP_PUT
- HTTP_PATCH
- HTTP_DELETE

ESP8266WebServer.on

Отзывы регистра, чтобы обработать запросы браузера.

```
пустота на (случайная работа константы *uri,
       ESP8266WebServer:: укладчик
       THandlerFunction)
недействительный ESP8266WebServer:: на
       (случайная работа константы *uri,
       метод HTTPMethod, ESP8266WebServer::
       укладчик THandlerFunction)
```

Зарегистрируйте функцию обратного вызова для URI и метода. Первый вариант функции будет обращаться с соответствующим URI для всех методов, в то время как второе позволяет нам обращаться с отзывами для той же части URI, но различных методов HTTP.

У функции укладчика есть подпись:

```
пустота (укладчик *) ()
```


ESP8266WebServer.onFileUpload

ESP8266WebServer.onNotFound

Зарегистрируйте отзыв, когда никакой определенный укладчик иначе не будет существовать.

```
пустота onNotFound (THandlerFunction fn)
```

Если никакой отзыв не был явно зарегистрирован для поступающего запроса URL, эта функция обратного вызова будет призвана как вместилище.

ESP8266WebServer.send

Пошлите ответ на браузер.

```
пустота посылает (международный кодекс, случайная работа константы
*contentType, Последовательность константы &content)
```

Пошлите данные в браузер. Это - основная точка входа ответа.

`Кодовый` параметр - код ответа HTTP. Кодовое обозначение 200 средств хорошо. `contentType` параметр - содержание ПАНТОМИМЫ полезного груза. Параметр `содержания` - фактическое содержание, чтобы послать.

ESP8266WebServer.sendContent

Пошлите последовательность в браузер. Это ниже находится на одном уровне интерфейс и использование посылания (), метод - правильный способ послать данные о приложении.

```
пустота sendContent (Последовательность константы &content)
```

Пошлите последовательность в браузер. Последовательность, переданная в содержании, используется, чтобы передать данные.

ESP8266WebServer.sendHeader

Пошлите заголовок HTTP.

```
пустота sendHeader (константа имя String&, константа стоимость String&,
bool сначала)
```

Добавьте заголовок HTTP к потоку продукции, посланному в браузер. Параметр имени - название заголовка, в то время как стоимость - ценность заголовка. Первый параметр говорит, будет ли заголовок добавлен впереди списка заголовков или в конце.

ESP8266WebServer.setContentLength

Установите длину содержания быть посланной.

```
пустота setContentLength (size_t contentLength)
```

ESP8266WebServer.streamFile

Теките содержание файла.

size_t streamFile (T &file, Последовательность константы &contentType)

ESP8266WebServer.upload

Закачка HTTPUpload& ()

ESP8266WebServer.uri

Восстановите текущий URI, который поставлялся браузером.

```
Uri последовательности ()
```

Это имеет основную ценность в укладчике отзыва запроса, где мы можем определить относительный путь URI.

Библиотека ESP8266mDNS

Рекламируйте нас через Передачу DNS. Библиотека под названием «ESP8266mDNS» должна быть добавлена к проекту, и включение названного «ESP8266mDNS.h» должно быть включено.

Исследуя эту библиотеку, это, кажется, **не** использует функции ESP8266 SDK для mDNS. Это кажется странным.

См. также:

- Системы доменных имен передачи

MDNS.addService

пустота addService (случайная работа *обслуживание, случайная работа *первичный, uint16_t порт);

пустота addService (случайная работа константы *обслуживание, случайная работа константы *первичный, uint16_t порт) пустота addService (Обслуживание последовательности, первичная Последовательность, uint16_t порт)

MDNS.begin

Начните отвечать на запросы mDNS.

bool начинаются (случайная работа константы* имя хоста);

bool начинаются (случайная работа константы* имя хоста, IP IPAddress,

uint32_t ttl=120)

Обратите внимание, что версия функции с больше, чем имя хоста осуществлена, игнорируя другие параметры. Прибыль функции, верная на успехе.

Обн
овл
ени
е
пуст
оты
MDN
S.up
date
();

I2C - Провод

Класс `Прослушки` оказывает поддержку I2C. Чтобы использовать этот класс, импортируйте `«Wire.h»` в свой эскиз. Когда мы используем этот класс, глобальный случай под названием `«Провод»` сделан доступным для нас. Один провод называют SCL, который обеспечивает часы, в то время как другой провод называют SDA и является шиной данных. На Ардуино, при этом поддержки библиотеки - или владелец или раб однако в текущем внедрении, только будучи владельцем, поддержан.

Чтобы использовать этот класс, сначала мы определяем, какие булавки должны использоваться и затем начать обслуживание.

```
Wire.begin (SDApin, SCLpin);
```

Чтобы послать данные, мы начинаем передачу, используя `beginTransmission ()`:

```
Wire.beginTransmission(deviceAddress);
```

теперь мы можем написать некоторые данные …

```
Wire.write (стоимость);
```

и наконец закончите передачу:

```
Wire.endTransmission ();
```

если мы хотим получить данные от раба, мы можем назвать `requestFrom ()`:

```
Wire.requestFrom (deviceAdress, размер, верный);
```

и данные могут быть прочитаны, используя `доступное ()` и `прочитаны ()` функции.

См. также:

- Работа с I2C
- Ардуино –Проводная библиотека

Wire.available

Определите число байтов, доступных, чтобы читать.

```
интервал, доступный (пустота)
```

Определите число байтов,

доступных, чтобы читать.См. также:

- Wire.read
- Wire.requestFrom

Wire.begin

Инициализируйте проводную библиотеку.

```
пустота начинается
(международный SDApin,
международный SCLpin),
пустота начинается ()
пустота
начинается (uint8
_t адрес),
пустота
начинается
(международный
адрес)
```

Страница 245

Инициализируйте проводную библиотеку. Когда адрес поставляется, мы - раб иначе, мы - владелец. Мы можем также определить булавки, которые будут использоваться для SDA и SCL. Если мы будем владельцем, и никакие булавки не поставляются, то мы будем использовать булавки по умолчанию.

ПРЕДУПРЕЖДЕНИЕ!! – Кажется, что нет **НИКАКОЙ** поддержки того, чтобы на самом деле быть рабом, и только владелец поддержан в это

время.

ПРЕДУПРЕЖДЕНИЕ!! – В текущем кодексе проигнорирован параметр `адреса`!!См. также:

- Wire.pins

Wire.beginTransmission

Прошение передачи блокирует рабу.

```
пустота beginTransmission
(uint8_t адрес) пустота
beginTransmission
(международный адрес)
```

Начните понятие отправки передачи к рабскому устройству с поставляемым адресом. Дальнейшие требования `написать ()` будут стоять в очереди данные, которые будут переданы, который наконец выполнен с а звоните в `endTransmission ()`.

Wire.endTransmission

Закончите заключение в скобки передачи.

```
uint8_t endTransmission (пустота)//Дефолты к
sendStop = истинный uint8_t endTransmission (uint8
_t sendStop)
```

Закончите заключение в скобки передачи и выполните фактическую передачу. Коды возврата:

- `0` – Переданный правильно
- `2` – Полученный NACK на передаче адреса
- `3` – Полученный NACK на передаче данных
- `4` – занятая линия

Wire.flush

Откажитесь от любых непрочитанных или ненаписанных данных.

```
недействительный TwoWire:: поток (пустота)
```

Откажитесь от любых непрочитанных или ненаписанных данных. Требование к `доступному ()` возвратится 0, и требование к `endTransmission ()` не передаст данных.

Wire.onReceive

Отзыв, когда мы находимся в роли раба и получаем передачу от владельца.

```
пустота onReceive (пустота (*function) (интервал numBytes))
```

Отзыв, когда раб получает передачу от

владельца. **ПРЕДУПРЕЖДЕНИЕ!!** – Эта функция

не осуществлена.

Wire.onReceiveService

Не осуществленный.

```
пустота onReceiveService (uint8_t* inBytes, интервал numBytes)
```

Не осуществленный.

ПРЕДУПРЕЖДЕНИЕ!! – Эта функция не осуществлена.

Wire.onRequest

Отзыв, призванный, когда мы находимся в роли раба и владельца, запрашивает данные от нас.

```
пустота onRequest (пустота (*function) (пустота))
```

Отзыв, призванный, когда мы находимся в роли раба и владельца, запрашивает данные от нас.

ПРЕДУПРЕЖДЕНИЕ!! – Эта функция не осуществлена.

Wire.onRequestService

Не осуществленный.

```
пустота onRequestService (пустота)
```

Не осуществленный.

ПРЕДУПРЕЖДЕНИЕ!! – Эта функция не осуществлена.

Wire.peek

Быстрый взгляд на следующий байт.

```
международный быстрый взгляд (пустота)
```

Быстрый взгляд на следующий байт, если Вы доступны. Возвращение-1, если нет никакого доступного байта.

Wire.pins

ПРЕДУПРЕЖДЕНИЕ!! - Эта функция была осуждена в пользу, `начинаются (sda, scl).`

Страница 247

Определите булавки по умолчанию для SDA и SCL.

```
недействительные булавки (интервал sda, интервал scl)
```

Определите булавки по

умолчанию для SDA и

SCL.См. также:

- Wire.begin

Wire.read

Прочитайте единственный байт.

```
интервал читал (пустота)
```

Прочитайте единственный байт от автобуса. Ценность `-1` возвращена,

если нет никакого доступного байта.См. также:

- Wire.available
- Wire.requestFrom

Wire.requestFrom

Запросите данные от раба.

```
size_t requestFrom (uint8_t адрес, size_t размер, bool sendStop)
uint8_t requestFrom (uint8_t адрес, uint8_t количество, uint8
_t sendStop) uint8_t requestFrom (uint8_t адрес, uint8_t
количество)
uint8_t requestFrom (международный адрес, международное количество)
uint8_t requestFrom (международный адрес, международное количество,
интервал sendStop)
```

Запросите данные от раба. Этот метод нужно назвать, когда мы играем роль владельца. Параметр `адреса` определяет рабский адрес для устройства, которое должно ответить. Если `sendStop` верен, сообщение остановки передано, выпустив автобус I2C. Если `sendStop` ложный, сообщение перезапуска передано, препятствуя тому, чтобы другой мастер шины брал на себя управление.

Параметр `количества` указывает, сколько байтов мы

хотим получить. Возвращаемое значение - число

байтов, которые были получены.

См. также:

- Wire.read
- Wire.available

Wire.setClock

Установите частоту часов.

```
пустота setClock (uint32_t частота)
```

Установите частоту часов. Всегда называйте `setClock ()` ПОСЛЕ требования `начаться ()`.

Wire.write

Напишите один или несколько байтов рабу.

```
size_t пишут (uint8_t данные)
size_t пишут (константа uint8_t *данные, size_t количество)
```

Напишите один или несколько байтов рабу.

Библиотека тикера

Эта библиотека настраивает функции обратного вызова, которые называют после промежутка времени. Чтобы пользоваться этой библиотекой, Вы должны включать «`Ticker.h`». Например:

```
#include <Ticker.h>

пустота timerCB ()
       {Serial1.println
       («Тиканье...»);
}

недействительная установка ()
{
       Serial1.begin (115200);
       ticker.attach (5, timerCB);
       Serial1.println («Тикер был
       свойственен»);
}
```

Тикер

Случай объекта Тикера. Обычно это создано как глобальное такое как:

```
Тикер myTicker;
```

быть свойственным

Приложите функцию обратного вызова к тикеру.

```
           пустота
       свойственна
(секунды плавания,
callback_t отзыв),
           пустота
       свойственна
(секунды плавания,
      пустота
      (*callback)
      (TArg), аргумент
      TArg)
```

Приложите функцию обратного вызова к тикеру, таким образом, что отзыв призван каждый период секунд. Обратите внимание, что секунды - плавание, таким образом, мы можем определить ценности такой как 0,1, чтобы указать на отзыв каждая 1/10-я из секунды (100 миллисекунд).

callback_t - определенный как:

```
пустота (*callback_t) (пустота)
```


attach_ms

Приложите функцию обратного вызова к тикеру.

```
пустота attach_ms (uint32_t
      миллисекунды,
      callback_t отзыв)
пустота attach_ms (uint32_t
      миллисекунды, пустота
      (*callback) (TArg),
      аргумент TArg)
```

Приложите функцию обратного вызова к тикеру, таким образом, что отзыв призван каждый период миллисекунд. Только одно приложение может быть сделано к таймеру.

отделить

Отделите тикер от таймера.

```
пустота отделяет ()
```

Отделите функцию обратного вызова от таймера. Никакие дальнейшие отзывы не произойдут.

однажды

Приложите функцию обратного вызова к таймеру для одноразового увольнения.

```
пустота однажды
       (пускают в
       ход секунды,
       callback_t
       отзыв),
пустота однажды (пускают в ход секунды,
       пустота
       (*callback)
       (TArg),
       аргумент TArg)
```

Приложите функцию обратного вызова к таймеру для одноразового увольнения. Обратите внимание, что секунды - плавание, таким образом, мы можем определить ценности такой как 0,1, чтобы указать на отзыв каждая 1/10-я из секунды (100 миллисекунд).

once_ms

Приложите функцию обратного вызова к таймеру для одноразового увольнения.

```
пустота once_ms (uint32_t
       миллисекунды,
       callback_t отзыв)
пустота once_ms (uint32_t
       миллисекунды, пустота
       (*callback) (TArg),
       Аргумент TArg)
```

Приложите функцию обратного вызова к таймеру для одноразового увольнения.

Библиотека EEPROM

Эта библиотека позволяет нам хранить и восстанавливать данные из хранения, которое сохраняется через перезапуск устройства. Объект единичного предмета под названием EEPROM предварительно поставляется для использования.

EEPROM.begin

Начните процесс написания или чтения от EEPROM. Размер -

объем хранения, с которым мы хотим работать.

```
пустота начинается (size_t размер)
```

EEPROM.commit

Изменения данных посвящают себя EEPROM. Возвращение `истинных` указывает на успех, в то время как возвращение ложных указывает на неудачу.

```
bool передают ()
```

EEPROM.end

Передает изменения данных и затем выпускает любое местное хранение. Нет далее читает или пишет, должен быть предпринят, пока следующие `не начинают ()` требование.

```
недействительный конец ()
```

EEPROM.get

Прочитайте структуру данных от хранения.

```
T &get (международный адрес, T &t)
```

EEPROM.getDataPtr

Восстановите указатель на хранение, которое мы собираемся прочитать или написать.

```
uint8_t *getDataPtr ()
```

EEPROM.put

Поместите структуру данных в хранение.

```
константа T &put (международный адрес, константа T &t)
```

EEPROM.read

Прочитайте байт от хранения.

```
uint8_t читал (международный адрес)
```

EEPROM.write

Напишите байт хранению.

```
пустота пишет (международный адрес, uint8_t стоимость)
```

МАГАРЫЧИ ПРИ ТОРГОВОЙ СДЕЛКЕ

FS - библиотека Файловой системы, которая обеспечивает способность прочитать и написать файлы из окружающей среды ESP Ардуино. Но ждите прочитанный … и напишите файлы туда, где? На ESP8266 нет никаких «двигателей». Данные для файлов прочитаны и написаны области флэш-памяти и так как вспышка относительно маленькая в размере (приблизительно 4 мегабайта макс.) тогда, это - верхняя граница максимального размера совокупных файлов … однако, это все еще более чем достаточно для многих образцов использования, таких как экономия государства, регистраций или информации о конфигурации.

SPIFFS.begin

Начните работать с файловой системой МАГАРЫЧЕЙ ПРИ ТОРГОВОЙ СДЕЛКЕ.

```
bool начинаются ()
```

Прибыль, верная об успехе и ложная иначе.

SPIFFS.open

Откройте названный файл.

```
Открытый файл (случайная работа
константы *путь, случайная работа
константы *способ) открытый Файл
(Последовательность константы &path,
случайная работа константы *способ)
```

Способ определяет, как мы хотим получить доступ к файлу. Варианты:

- `r` – Прочитанный файл. Файл должен существовать.
- `w` – Пишут файлу. Усеките файл, если он существует.
- – Прилагают к файлу.
- `r +` – Прочитанный и пишут файл.
- `w +` – Прочитанный и пишут файл.
- `+` – Прочитанный и пишут файл.

См. также:

- File.close

SPIFFS.openDir

Откройте справочник.

```
Директор openDir (случайная работа константы *путь)
Директор openDir (Последовательность константы &path)
```

SPIFFS.remove

Удалите/удалите файл из файловой системы.

```
bool удаляют
(случайная работа
константы *путь) bool
удаляют
(Последовательность
константы &path)
```


SPIFFS.rename

Переименуйте файл.

```
bool переименовывают (случайная работа
константы *pathFrom, случайная работа
константы *pathTo) bool переименовывают
(Последовательность константы &pathFrom,
Последовательность константы &pathTo)
```

File.available

Возвратите число байтов, которые доступны в рамках файла от текущего положения файла до его максимального размера.

```
доступный интервал ()
```

File.close

Закройте ранее открытый файл.

```
пустота близко ()
```

Никакие дополнительные материалы для чтения, ни письмо не должны быть предприняты, чтобы быть выполненными.

File.flush

Смойте файл.

```
недействительный поток ()
```

File.name

Восстановите название файла.

```
случайная работа константы *имя ()
```

File.peek

Быстрый взгляд на следующий байт данных в файле, не потребляя его.

```
международный быстрый взгляд ()
```

File.position

Восстановите текущее положение указателя файла.

```
положение size_t ()
```

File.read

Прочитайте данные из файла.


```
интервал читал ()
size_t читал (uint8_t *buf, size_t размер)
```

Прочитайте или единственный байт данных или буфер данных из файла.

File.seek

Смените текущее положение указателя файла.

```
bool ищут (uint32_t pos, способ SeekMode)
```

Способ может быть одним из:

- `SeekSet` – Меняют положение указателя файла к абсолютной величине.
- `SeekCur` – Меняют положение указателя файла, чтобы быть относительно текущего положения.
- `SeekEnd` – Меняют положение указателя файла, чтобы быть относительно конца файла.

File.size

Восстановите максимальный размер файла.

```
размер size_t ()
```

File.write

Напишите данные файлу.

```
size_t пишут (uint8_t c)
size_t пишут (uint8_t *buf, size_t размер)
```

Напишите или единственный байт или буфер байтов в файл в текущем положении указателя файла.

Dir.fileName

Восстановите название файла.

Имя файла последовательности ()

Dir.

next

bool

зат

ем

()

Dir.open

```
Открытый файл (случайная работа константы *способ)
Открытый файл (Натягивают &path, случайная работа константы *способ),
```

Dir.openDir

```
Директор openDir (случайная работа константы *путь)
Директор openDir (Последовательность &path)
```


Dir.remove

Dir.rename

Библиотека ESP

Классу предоставили названный ESP, который предоставляет окружающей среде ESP8266 определенные функции. Вы должны понять, что использование этих функций приведет к Вашим заявлениям, не являющимся портативным в Ардуино (если это будет желанием).

ESP.deepSleep

пустота deepSleep (uint32_t time_us, способ WakeMode)

ESP.erase

Config
bool
eraseConf
ig ()

ESP.getBootM
ode uint8_t
getBootMode
()

ESP.getBootVers
ion uint8_t
getBootVersion
()

ESP.getChi
pId uint32
_t
getChipId ()

ESP.getCpuFreqM
Hz uint8_t
getCpuFreqMHz ()

ESP.getCycleCou
nt uint32_t
getCycleCount ()

ESP.getFlashChip
Id uint32_t
getFlashChipId ()

ESP.getFlashChipMode
FlashMode_t getFlashChipMode ()

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
Натяните getResetInfo ()

ESP.getResetInfoPtr
структура rst_info * getResetInfoPtr ()

ESP.getSdkVersion
Восстановите представление последовательности SDK того, чтобы быть используемым.

```
случайная работа константы *getSdkVersion ()
```

ESP.getSketchSize uint32_t getSketchSize ()

ESP.getVcc uint16_t getVcc ()

ESP.reset
недействительный EspClass:: сброс ()

ESP.restart
недействительный EspClass:: перезапуск ()

ESP.updateSketch
bool updateSketch (Stream& в, uint32_t размер,
bool restartOnFail, bool restartOnSuccess)

Пустота ESP.wdtDisable wdtDisable ()

ESP.wdtEnable
пустота wdtEnable (uint32_t timeout_ms)

Пустота ESP.wdtFeed wdtFeed ()

Библиотека последовательности

Хотя считается, что эта библиотека может быть идентична библиотеке Последовательности Ардуино, я полагаю, что столь важно понять, что я

собираюсь перечислить методы снова.

Последовательность

Конструктор

```
Последовательность
(случайная работа
константы *cstr =»»);
Последовательность
(Последовательность
константы &str)
Последовательность
(случайная работа c)
Последовательность (неподписанная
случайная работа, неподписанная
случайная работа базируется = 10),
Последовательность (интервал,
неподписанная основа случайной работы =
10) Последовательность (долгая,
неподписанная основа случайной работы =
10) Последовательность (неподписанная
долгая, неподписанная основа случайной
работы = 10) Последовательность
(плавание, неподписанная случайная
работа decimalPlaces = 2)
Последовательность (двойная,
неподписанная случайная работа
decimalPlaces = 2)
```

Создайте случай класса Последовательности, отобранного с различными инициализаторами типа данных.

Последовательность c_str

Восстановите представление струны до.

```
случайная работа константы *c_str ()
```

String.reserve

неподписанный запас случайной работы (неподписанный международный размер)

String.length

Возвратите длину последовательности.

```
неподписанная международная длина ()
```

Возвратите длину последовательности.

String.concat

```
неподписанная случайная работа
concat (Последовательность
константы &str) неподписанная
случайная работа concat
(случайная работа константы
*cstr) неподписанная случайная
работа concat (случайная работа
c)
неподписанная случайная работа
concat (неподписанная случайная
```

```
работа c) неподписанная
случайная работа concat
(международная цифра)
неподписанная случайная работа
concat (неподписанная
международная цифра)
неподписанная случайная работа
concat (длинная цифра)
неподписанная случайная работа
concat (неподписанная длинная
цифра) неподписанная случайная
работа concat (пускают в ход
цифру), неподписанная случайная
работа concat (удваивают цифру),
```

String.equalsIgnoreCase

```
неподписанная случайная работа equalsIgnoreCase (Последовательность
константы &s) константа;
```

String.startsWith

Определите, начинается ли эта последовательность с другой последовательности.


```
неподписанная случайная работа startsWith (Последовательность константы
&prefix)
неподписанная случайная работа startsWith (Последовательность константы
&prefix, неподписанное международное погашение)
```

String.endsWith

```
неподписанная случайная работа endsWith (Последовательность константы
&suffix)
```

String.charAt

```
случайная работа charAt (неподписанный международный индекс)
```

String.setCharAt

```
пустота setCharAt (неподписанный международный индекс, случайная работа c)
```

String.getBytes

```
пустота getBytes (неподписанная случайная работа *buf, неподписанный
интервал bufsize, неподписанный международный индекс = 0)
```

Последовательность toCharArray

```
пустота toCharArray (случайная работа *buf, неподписанный интервал bufsize,
неподписанный международный индекс = 0)
```

String.indexOf

Найдите положение последовательности или характера в текущей последовательности.

```
интервал indexOf (случайная работа ch)
интервал indexOf (случайная работа ch,
неподписанный интервал fromIndex)
интервал indexOf (Последовательность
константы &str)
```

```
интервал indexOf (Последовательность константы &str, неподписанный интервал
fromIndex)
```

Найдите положение последовательности или характера в текущей последовательности. Если матч не найден,`-1` возвращен иначе, положение начала матча возвращено.

String.lastIndexOf

```
интервал lastIndexOf (случайная работа ch)
интервал lastIndexOf (случайная работа
ch, неподписанный интервал fromIndex)
интервал lastIndexOf (Последовательность
константы &str)
интервал lastIndexOf (Последовательность константы &str, неподписанный
интервал fromIndex)
```

String.substring

Восстановите подстроку из текущей последовательности.

```
Подстрока последовательности (неподписанный интервал beginIndex)
Подстрока последовательности (неподписанный интервал beginIndex,
неподписанный интервал endIndex)
```

Восстановите подстроку из текущей последовательности.

String.replace

```
пустота заменяет (случайная работа находят, случайная работа заменяют),
пустота заменяет (константа, которую String& находят, константа, которую
String& заменяют),
```

String.remove

```
пустота удаляет (неподписанный международный индекс)
пустота удаляет (неподписанный международный индекс, неподписанное
международное количество)
```

Пустота String.toLowerCase (пустота)

Пустота String.toUpperCase (пустота)

String.trim

```
недействительная отделка (пустота)
```

String.toInt

```
длинный toInt (пустота)
```

String.toFloat

```
пустите в ход toFloat (пустота)
```

Программирование с JavaScript

JavaScript - интерпретируемый язык высокого уровня. Некоторые ее основные конструкции - свободная печать, объектно-ориентированная, поддержка функций лямбды, поддержка закрытий и, самое главное, стала языком сети. Если Вы пишете, что браузер принял применение, то это - уверенность, что это будет написано в JavaScript. Но что из окружающей среды небраузера? Некоторое время теперь JavaScript разъедал серверный код через проекты, такие как Node.js. Как язык для бегущего кода сервера, у этого есть значительный набор особенностей, чтобы понять эту способность. А именно, это поддерживает управляемую событиями парадигму архитектуры. В JavaScript мы можем зарегистрировать функции, которые будут призваны обратно на обнаруживаемые события. Эти отзывы могут быть определены как простые действующие функции на том, что сделать. В них составил примеры, мы иллюстрируем это:

```
httpServer.on (»/path1»,
       функция ()
       {//Делает что-то
       для (ответа)/path1
       httpServer.send;
});
```

или

```
socket.accept (порт, функция
       (newSocket) {newSocket.on
       («получают», функция (данные) {
              печать («Мы получили новые данные»:
              + данные); newSocket.send («Мы
              получили данные», функция () {
                     newSocket.close ();
              });
       });
});
```

И если мы можем осуществить хорошую модель JavaScript, она наносит на карту превосходно к модели ESP8266 мира, который самостоятельно управляем событиями через отзывы. Это отображение не будет легким …, но планы надвигаются.

Еспруино - общедоступный проект обеспечить пробег JavaScript, разовый для встроенных устройств. Это было осуществлено для процессоров ARM Cortex M3/M4 и других.

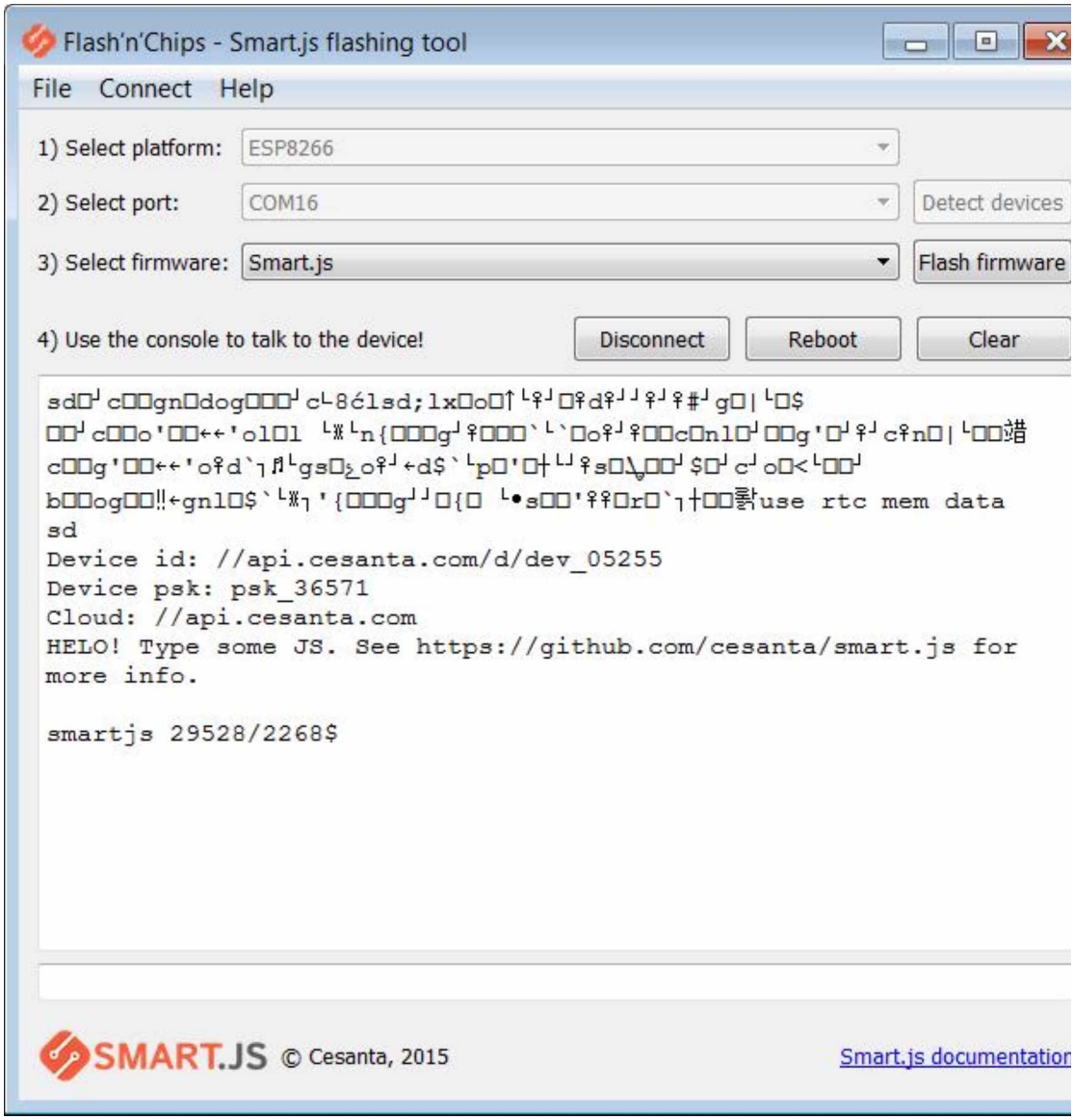
Вопрос теперь - …, он может использоваться для ESP8266? Активный проект пытается сделать просто это.

См. также:

- Еспруино

Smart.js

Smart.js - внедрение JavaScript для множества вложенных платформ. Это оказывает родную поддержку для большинства общих функций ввода/вывода включая GPIO, SPI, I2C, PWM. У этого есть сетевая поддержка через WiFi и сервер HTTP. Это поддерживает файловую систему.



Страница 261

Настроить WiFi,

пробег: wifi.setup

(ssid, пароль),

Чтобы

перечислить

файлы …

управляют File.list

('. ')

Чтобы определить который версия, управляйте версией

См. также:

- GitHub:cesanta/smart.js
- Домашняя страница Smart.js
- Центр разработчика Smart.js

Smart.js GPIO

У Smart.js есть некоторые сложные возможности GPIO. Прежде, чем читать или написать от булавки, мы должны сначала объявить способ доступа. Мы можем сделать это с `GPIO.setmode (булавка, способ, усилие)`. Параметр булавки - булавка, которую мы определяем, способ - способ ввода/вывода, который закодирован как:

- 0 – Вход и выход

- 1 – Вход

- 2 – Продукция
- 3 – Перерывы

Параметр усилия определяет любые особенности усилия:

- 0 – плавание
- 1 – подтягивание
- 2 – со спуском

Как только способ был установлен, мы можем читать от булавки через `GPIO.read (булавка)` или написать булавке с `GPIO.write (булавка, стоимость)`.

Если мы хотим использовать перерывы, мы можем зарегистрировать функцию, которую назовут, когда ценность булавки изменяется. Это установлено с `GPIO.setisr (булавка, напечатайте, функция)`. Параметр типа - кодирование типа изменения сигнала, которое вызовет перерыв.

- 0 – Перерывы отключены
- 1 – Положительный край
- 2 – Отрицательный край
- 3 – Любой край
- 4 – Низко
- 5 – Высоко
- 6 – Кнопка

У функции, которая вызвана, есть `функция` подписи `(булавка, уровень)`.

Связанный с GPIO понятие PWM. Мы можем определить PWM, использующий `PWM.set (булавка, точка, обязанность)`. И период и обязанность поставляются в микросекундах.

Подготовка сервера HTTP

Smart.js обеспечивает внедрение сервера Http. Предварительно поставляемый класс под названием «Http» доступен. Сначала мы создаем случай сервера Http, используя `createServer ()` метод.

```
вар httpServ = Http.createServer (функция (req, req) {});
```

Создание сервера немедленно не начинает его слушание, вместо этого `слушание ()`, метод на сервере Http нужно назвать.

```
httpServ.listen (80);
```

Когда браузер соединится с сервером, функция будет вызвана с параметрами и запроса и объекта ответа.
Объект запроса содержит следующие свойства:

- `URL` – относительная часть URL запроса

Объект ответа содержит следующие методы:

- writeHead (статус, заголовки)
- напишите (данные)
- конец ()

Отладка

Управляя Smart.js программы JavaScript, мы можем хотеть сделать некоторую отладку. Мы можем выполнить регистрацию, используя `печать ()` или `Debug.print ()`.
От памяти и другой точки зрения ресурсов, команда `GC.stat ()` возвращает объект, который содержит детали об имеющихся ресурсах. Это включает:

- owned_max
- находящийся в собственности
- astsize
- funcncell
- funcnfree
- propncell
- propnfree
- objncell
- objfree
- struse
- strres
- jsfree
- jssize
- sysfree
-

Еспруино

Еспруино - общедоступное внедрение JavaScript, который работает на множестве платформ включая ESP8266. Веб-страница для поддержки ESP8266 может быть найдена здесь:

http://www.espruino.com/EspruinoESP8266

Строение из Еспруино для ESP8266 доступно, поскольку часть основного набора строит доступный здесь:

http://www.espruino.com/Download

Вот рецепт для загрузки и установки копии Еспруино на ESP8266, используя Linux в качестве основы.

```
$ mkdir
espruin
o CD $
espruin
o
$ wget $
http://www.espruino.com/files/espruino_
1v87.zip расстегивают молнию на espruino_
1v87.zip
$ cd espruino_1v87_esp8266
$      esptool.py - порт/dev/ttyUSB0 - бод 115 200 write_flash \-flash_freq
80 м - flash_mode qio - flash_size 32 м \0x0000 «boot_v1.4 (b1) .bin» \
       0x1000 espruino_esp8266_user1.bin \0x37E000 blank.bin
```

Когда я управлял этим, результат был следующим:

```
esptool.py v1.2-dev
Соединение...
Управление окурком маяка мигалки Cesanta...
Высветите набор params к 0x004f
Написание 4096 0x0... 4096 (100%)
Написал 4 096 байтов в 0x0 за 0,4 секунды (88,7 кбит/с)...
Написание 466944 0x1000... 466944 (100%)
Написал 466 944 байта в 0x1000 за 41,0 секунды (91,0 кбита/с)...
Написание 4096 0x37e000... 4096 (100%)
Написал 4 096 байтов в 0x37e000 за 0,4 секунды (88,6 кбит/с)...
Отъезд...
```

Если `esptool.py` не установлен, он может быть добавлен с:

```
$ sudo склонный -
добираются,
устанавливают зернышко
$ зернышка питона,
```

```
устанавливают esptool
```

Редактирование и развертывание кодекса

У Еспруино есть изящный редактор кода и инструмент развертывания, который основан на Хроме. Это позволяет нам писать наш JavaScript и загружать его в разовый пробег Еспруино. Мы можем также использовать `свалку ()` метод, чтобы спросить Еспруино, что это установило как программа. Это может быть скопировано назад редактору.

Мало того, что веб-редактор Еспруино доступен для выполнения в Хроме, это доступно как приложение, которое может быть в местном масштабе установлено. У принятия Вас есть местная копия установленного Node.js, мы можем бежать:

```
$ sudo npm устанавливают-g
espruino-web-ide $ espruino-
web-ide
```

Это начнет местную копию средств разработки.

Работа с переменными

Объект назвал «`глобальным`», может использоваться, чтобы перечислить глобальные данные в объеме.

Загрузка Еспруино

То, когда Еспруино загрузит, он продолжится от того, где это было в кодексе, когда «`экономят ()`», было выполнено. Эффект управления «экономит ()», должен заставить государство программы быть написанным, чтобы вспыхнуть и затем загруженный на следующем ботинке. Если мы должны выполнить инициализацию на ботинок, мы используем `E.ON () init` обработчик событий. Например:

```
E.ON («init»,
   функция ()
   {//Кодекс
   здесь...
});
```

Метод, названный «`сброс ()`», перезагрузит штат Эспруино, не выполняя сброс аппаратных средств.

Доступ WiFi

Модуль под названием «Wi-Fi» содержит большинство функций доступа WiFi. Чтобы использовать это, мы вводим модуль с:

```
Wi-Fi вара = требует («Wi-Fi»);
```

Чтобы начать слушать как точка доступа, мы можем использовать `startAP ()` метод. Например:

```
Wi-Fi вара =
требует («Wi-Fi»);
wifi.startAP
(«ESP8266», {
  «authMode»:
«открытый»},
функция
(допускает
ошибку) {
  console.log («AP теперь
начался»: + допускают
ошибку);});
```

Мы можем опросить статус Точки доступа, используя `getAPDetails ()`. Например:

```
wifi.getAPDetails
   (функция (данные)
   {//обрабатывают данные
});
```


Структура возвратилась, мог бы быть похожим:

```
{
  «статус»:
  «позволенный
  »,
  «authMode»:
  «открытый»,
  «скрытый»: ложный,
  «maxConn»: 4, «ssid»:
  «ESP8266», «пароль»:
  «»,
  «savedSsid
  »: пустой
  указатель,
  «станции»:
  [
    {
      «IP»:
      «192.168.4.2»,
      «Mac»:
      «00:36:76:21:97:a3
      »
```

```
      }
    ]
  }
```

Обратите внимание, что «станции» - множество станций, которые связаны с точкой доступа.

Если мы хотим знать IP-адрес точки доступа, мы можем назвать `wifi.getAPIP ()`. Например:

```
wifi.getAPIP
   (функция (данные)
   {console.log
   (данные);
});
```

Пример того, что могло бы быть возвращено, будет:

```
{
  «IP»:
  «192.168.4.1»,
  «netmask»:
  «255.255.255.0»,
  «gw»:
  «192.168.4.1»,
  «Mac»:
  «1a:fe:34:f5:2e:ec
  »
}
```

Альтернатива тому, чтобы быть точкой доступа является станцией. Если мы выбираем тот выбор тогда, мы добираемся, чтобы соединиться с существующей сетью WiFi как станционный участник. Мы делаем это, звоня `wifi.connect ()`. Это берет идентичность сети, чтобы соединиться с, варианты и функция обратного вызова. Например:

```
wifi.connect
   («mySssid»,
   {пароль:
   «myPassword»,
}, функция (допускает ошибку) {
   console.log («Соединяют законченный ..., допускает ошибку,»: + допускают
   ошибку);
});
```

Обратите внимание, что варианты и функции обратного вызова дополнительные.

Написание сетевых приложений гнезда, используя Еспруино

К сетевой библиотеке, обеспеченной окружающей средой Еспруино, можно получить доступ, используя:

```
вар, чистый =, требует («чистый»)
```

Это возвращает сетевой объект, который выставляет операции, необходимые, чтобы взаимодействовать с сетью. Методы, выставленные этим объектом, включают:

- `createServer` – Создают случай сервера
- `соединитесь` – Соединяются с партнером

```
net.connect («http://1.2.3.4:80», функция (связь)
       {//Связь - объект связи партнеру.
});
```

Объект связи (который называет «Гнездом» Еспруино) всегда является ссылкой на определенную связь между ESP8266 и партнером по сети. У объекта связи есть следующие методы:

- `доступный` – число байтов, доступных, чтобы читать. Это будет всегда возвращаться 0 если а
  слушатель данных был зарегистрирован.
- `читайте` – Прочитанный некоторое число байтов.
- `напишите` – Пишут некоторое число байтов.
- `конец` – Заканчивает связь.
- `на («данных»)` – Регистр отзыв, который будет призван, когда данные становятся
  доступный. Сеть Writing приложения HTTP, используя Еспруино

Написание клиента ОТДЫХА, использующего Еспруино

Клиент ОТДЫХА - тот, который отправляет запросы ОТДЫХА, которые являются в основном запросами HTTP. На высоком уровне следующий пример иллюстрирует, как достигнуть этого:

```
  вар http = требует
     («http»); вар
getRequest = http.get
       ({хозяин:
    «192.168.1.9»,
     порт: 1817,
     путь: «/sendmessage?
  message=1»}, функция
  (ответ) {
     //Ответ ручки здесь
  }
);
```

Есть два основных параметра к `получению ()` метод. Первыми являются варианты объект JavaScript, которые настраивают детали остальных запрос. Этот объект включает:

- `хозяин` – IP-адрес цели остальных запрос

- `порт` – номер порта, к которому будет отправлен запрос
- `путь` – относительная часть URL
- `метод` – Неизвестный

Страница 268

- `заголовки` – объект JavaScript, представляющий заголовки, чтобы послать.

Второй параметр - функция обратного вызова, которая будет призвана, когда ответ остальным запрос будет получен. У функции обратного вызова есть подпись:

```
отзыв (ответ)
```

где `ответ` - случай объекта `HTTPCRS` Еспруино. У этого объекта есть следующие методы:

- `доступный` – Возвращение число доступных байтов для чтения.
- `событие близко` – Названный, когда связь закрыта.
- `данные событий` – Названный, когда есть доступные данные.
- `труба` – Делает что-то с трубой.
- `читайте` – Прочитанные доступные данные.

Кроме того, у объекта ответа есть собственность, названная заголовками, которые являются заголовками ответа HTTP, возвращенными из сервера. Они будут включать обычные свойства HTTP, которые могут включать:

- Тип контента
- Дата
- Связь
- Довольная длина

Возвращение из `http.get ()` вызов функции является объектом типа `HTTPCRQ`.

Написание веб-сервера, используя Еспруино

Мы можем также настроить нашу среду, чтобы быть веб-сервером, который может обработать поступающие запросы клиента HTTP. Чтобы сделать это, мы должны получить доступ `к http` использованию класса, `требуют («http»)`. У этого класса есть метод на нем, чтобы создать

случай веб-сервера. У метода есть подпись:

```
createServer (handlerFunction)
```

Функция обратного вызова укладчика призвана, когда каждый новый запрос клиента прибывает. В большинстве случаев клиент будет браузером, но это могло так же, как легко быть другое применение, обращающееся с просьбой ОТДЫХА. Укладчик отзыва берет два параметра в качестве входа. Первым является объект, представляющий запрос `(httpSRq)` и второе объект, представляющий ответ `(httpSRs)`

`createServer` метод возвращает случай объекта сервера HTTP `(httpSrv)`. Мы можем тогда начать это сервер HTTP, слушающий с требованием `послушать ()`, который берет номер порта, на который должен послушать сервер.

Например:

```
вар http = требует («http»);
вар httpServer = http.createServer (функция (запрос, ответ)
{...}); httpServer.listen (80);
```

Как только сервер начинает слушать, поступающие запросы HTTP заставят функцию укладчика быть призванной, проходя в объекте, представляющем запрос и объект, который может использоваться, чтобы сформулировать ответ.

У объекта запроса, переданного к функции обратного вызова, есть следующие свойства:

- `URL` – местный путь поступающего запроса.
- `метод` – метод HTTP поступающего запроса.
- `заголовки` – заголовки содержатся в данных.

Так как параметр URL может быть составлен из вариантов вопроса, могло бы быть хорошо быть в состоянии извлечь их. Вот фрагмент, который сделает просто что:

```
вар partsOfUrl =
request.url.split (»?»); если
(partsOfUrl.length> 1) {
  варианты вара =
  partsOfUrl[1].split ('& ');
  вар optionsObj = {};
  для (вар i=0; я <options.length; я ++) {
    вар splitEquals = варианты
```

```
    [я] .split (' = '); optionsObj
    [splitEquals[0]] = splitEquals[1];
  }
  печать («Финал obj»: + JSON.stringify (optionsObj));
}
```

Если поступающий запрос - ПОЧТА HTTP или ПОМЕЩЕННАЯ команда, нам можно было послать дополнительные данные о полезном грузе как часть запроса от клиента. Мы получаем доступ к тем данным, регистрируя отзыв данных событий на объекте запроса:

```
request.on («данные», функция
   (receivedData)
   {//Обрабатывает данные
});
```

Позвольте нам пауза здесь на мгновение, чтобы рассмотреть то, что происходит. Данные прибывают от посетителя как поток гнездо TCP. Это означает, что данные прибудут в заказ, но не обязательно все однажды. Если клиент хочет послать в 100 байтах данных о полезном грузе в запросе, мы можем получить все 100 байтов как единственный кусок, или мы могли бы столь же легко получить 100 отзывов 1 байта каждый. По сути, это - наша обязанность гарантировать, что мы получили все данные, в которых мы нуждаемся перед обработкой. Если мы будем следовать за этим понятием, то мы также поймем, что `request.available ()` и `request.read ()` методы должен быть понят правильно. Запрос `request.available ()` говорит нам, какие данные были получены

до сих пор … и не указывает, сколько фактических данных может быть в конечном счете получено. Точно так же `request.read ()` данные о прибыли метода, которые были получены, но не блокируют ожидание новых/дополнительных данных, чтобы прибыть.

Событие сделано доступным, чтобы определить, когда клиент послал все необходимые данные. Это - через `request.on («близко», функция () …)` механизм. Мы можем зарегистрировать близкого укладчика отзыва, чтобы быть информированными, когда клиент закрыл связь. На данном этапе мы можем прочитать все доступные данные () и читать (), потому что мы знаем, что не будет никакого дальнейшего прибытия данных.

Чтобы возвратить ответ, мы можем призвать `writeHead (код состояния, заголовки)` метод, чтобы установить код состояния и заголовки для возвращения. Чтобы написать содержание в ответе, мы можем использовать:

```
response.write (данные);
```

Чтобы проверить, мы должны отправить запрос HTTP. Принятие нас не проверяет с браузером, мы можем использовать:

```
wget http://<IP-адрес> - тихий - документ продукции =-
```

- тихий флаг выключает болтовню приложения, в то время как - документ продукции пишет полученные данные stdout.

Чтобы отправить запрос, содержащий данные, мы можем использовать:

```
wget http://<IP-адрес> - постданные» <некоторые данные> - тихий - документ продукции =-
```

Работа с GPIO

Еспруино оказывает поддержку GPIO через многие глобальные методы. Они включают «pinMode ()», чтобы установить способ булавки, «getPinMode ()» заставлять способ булавки, «digitalWrite ()» устанавливать логический уровень, «digitalRead ()» заставлять логический уровень, «digitalPulse ()» пульсировать логический уровень.

Работа с I2C и JavaScript

Механизм программного обеспечения I2C доступен от класса I2C, который является встроенным. Чтобы использовать, мы можем усилить случай объекта I2C. Еспруино предварительно создает тот по имени I2C1. Оттуда, мы можем использовать установку () метод на него. Установка () метод берет объект в качестве входа, у которого есть свойства:

- scl – булавка, которая будет использоваться для часов (дефолт равняется 14),
- sda – булавка, которая будет использоваться для SDA (дефолт равняется 2),
- битрейт – битрейт коммуникации (дефолт 50000),

Чтобы написать устройству I2C, мы можем призвать writeTo () метод. У этого есть подпись:


```
writeTo (адрес, данные, ...)
```

Где адрес - адрес рабского устройства I2C, и данные - данные, чтобы передать. Чтобы прочитать данные, мы можем использовать readFrom () метод. У этого есть подпись:

```
readFrom (адрес, количество)
```

Где `адрес` - адрес рабского устройства I2C, и `количество` - число байтов, чтобы читать. Результат - множество байтов.

Константы удобства доступны названный

«ВЫСОКО» и «НИЗКО».См. также:

- Работа с I2C

Отладка JavaScript

Есть много способов, которыми мы можем отладить код JavaScript.

Первым является через свалку () заявление. Это зарегистрирует текущее состояние переводчика.

След () заявление может использоваться, чтобы свалить переменные включая их типы. Это может взять имя переменной.

Глобальная переменная - определитель объема. Используя глобальный [» \xFF»] получит доступ к Еспруино «скрытые» переменные.

Редактирование JavaScript

Лично, я предпочитаю использовать программную окружающую среду Затмения для всей своей работы. Мы можем установить средства разработки JavaScript, чтобы предоставить нам хорошего редактора JavaScript. Это может быть установлено через нормальные инсталляционные механизмы программного расширения Затмения. Просто поиск JavaScript в устанавливаемых компонентах. После того, как установленный и редактирование сценария, Вы добираетесь, все виды языковой поддержки JavaScript включая вход помогают и схема программы:

Библиотеки Espruino ESP8266

Есть определенная библиотека ESP8266, чем можно получить доступ из потребовать заявления, которое похоже:

```
вар esp8266 = требует («ESP8266»);
```

Есть много методов, выставленных на возвращенном объекте включая:

- `crc32` – Создают 32-битный CRC.
- `deepSleep` – Заставляют ESP8266 перейти к режиму «глубокого сна». Эффективно рассчитанный
  перезагрузка.
- `dumpSocketInfo` – информация о гнезде Отладки, в письме к это регистрации.
- `getFreeFlash` – Возвращение сумма свободной флеш-памяти. Осуждаемый.
- `getResetInfo` – Восстанавливают причину последнего сброса/перезагрузки.
- `getState` – Восстанавливают государство устройства
  - `sdkVersion` – Версия SDK
  - `cpuFrequency` – MHZ скорости центрального процессора
  - `freeHeap` – Объем бесплатного хранения кучи
  - `maxCon` – Максимальное количество связей

- `flashMap` – Как вспышка нанесена на карту
- `flashKB` – Настроенный размер вспышки
- `flashChip` – Производитель чипа вспышки

вот пример продукции:

```
{
  «sdkVersion»: «1.5.0»,
  «cpuFrequency»: 160, «freeHeap»: 10096,
  «maxCon»: 10, «flashMap»: «4MB:512/512»,
  «flashKB»: 4096,
  «flashChip»: «0xe0
  0x4016»
 }
```

- `logDebug` – Позволяют или отключают регистрацию отладки.
- `neopixelWrite` – Пишут ряду NeoPixels.
- `звон` – Звон данный IP-адрес.
- `printLog` – Печать журнал отладки к пульту.
- readLog
- `перезагрузка` – Перезагружает устройство.
- `setCPUFreq` – Набор частота центрального процессора. Осуждаемый.
- `setLog` – Набор способ регистрации
  - 0 – прочь
  - 1 – в памяти
  - 2 – в памяти и UART0
  - 3 – в памяти и UART1

Основные возможности JavaScript

Язык JavaScript, обеспеченный Espruino, покрыт подробно документацией Espruino. Однако вот некоторые основные самородки, которые я нахожу чрезвычайно полезными.

См. также:

- Ссылка программного обеспечения Espruino

Управление кодексом с промежутками

Мы можем определить таймер, который уволит любого однажды (`setTimeout ()`) или периодически (`setInterval ()`) и вызывать функцию.

Синтаксис для этого:

```
setTimeout (функция, задержка, [args,
…]) setInterval (функционируют, точка
[args, …]),
```


Где задержка и период - времена, измеряемые в миллисекундах. Дополнительные аргументы могут также поставляться, которые переданы к функции. Обе этих функции возвращают идентификационную стоимость, которая может использоваться, чтобы отменить запрос, прежде чем это произойдет. Функция, чтобы сделать это называют `clearInterval ()`.

```
clearInterval (id)
```

Мы можем также изменить интервал на периодическом отзыве с `changeInterval ()` функция.

```
changeInterval (id, newPeriod)
```

Работа с GPIO

Мы можем определить объект Булавки типа представлять булавку GPIO и затем или установить ее значение или прочитать ее стоимость. Например, вот простой blinky:

```
вар ledOn = ложный;
вар ledPin =
новая Булавка
(4);
setInterval
(функция () {
       digitalWrite
       (ledPin, ledOn);
       ledOn =! ledOn;
}, 1000);
```

SPI

SPI - проводной протокол, используемый, чтобы вести соответствующие интерфейсные компоненты SPI. У Еспруино есть модуль под названием SPI, который предоставляет нам доступ к тем возможностям. Сначала

мы создаем использование порта SPI:

```
вар mySPI = новый SPI ();
```

Затем мы можем настроить этот порт, используя `установку ()` функция. Параметр к установке - объект, который содержит:

- `sck` – Булавка, чтобы использовать для часов.
- `мисо` – булавка, чтобы использовать для владельца в.
- `mosi` – булавка, чтобы использовать для владельца/раб в.
- (дополнительный) `бод` – Дефолты к 100 000.
- (дополнительный) `способ` – Дефолты к 0.
- (дополнительный) `заказ` – Дефолты к «msb».

Наконец, мы можем назвать `писание ()` функция, чтобы написать данные. Кроме того, мы можем звонить, `посылают ()`.

Вот пример. MAX7219 - сильный небольшой IC, который в состоянии двигаться 8x8 матрица светодиодов. Будучи устройством SPI, это использует три сигнала SPI:

- `CS` – Низко, чтобы выбрать MAX7219 для коммуникации SPI.
- `MOSI` – линия данных, по которой будут течь последовательные данные.
- `CLK` – прием управления линии часов новых частей данных.

Если мы смотрим на ESP8266, мы могли бы нанести на карту их к следующим булавкам ESP8266:

Функция	Булавка	NodeMCU	Цвет
CS	GPIO 12	D6	
MOSI	GPIO 13	D7	
CLK	GPIO 14	D5	

Основные отличия из JavaScript

Хотя Еспруино - превосходное внедрение, у него действительно есть некоторые различия от стандарты ECMAScript. Некоторые вещи тонкие и маловероятные споткнуться через то, в то время как другие более

крупные. Вот некоторые примеры:

- Вызывание функций перед их декларацией не поддержано.
- Объявление переменной как константа не делает его так.

Строительство Еспруино

Построить Еспруино из исходного дерева:

```
Мерзавец $ клонирует
https://github.com/espruino/Espruino.git
CD $ Еспруино
$ экспортируют
экспорт $
ESP8266_BOARD=
1 FLASH_4MB=1
Экспортные $ ESP8266_SDK_ROOT
=/esp8266/sdk/ESP8266_NONOS_SDK $
экспортируют $PATH:/pot/xtensa-lx106-elf/bin
PATH=
$ экспортируют ESPHOSTNAME=espruino:88
```

Создайте директора 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. почтовый индекс
$ mkdir src
$ расстегивают молнию на esp8266_nonos_sdk_v2.0.0_16_08_10.zip-d sdk
$ расстегивают молнию на esp8266_nonos_sdk_v2.0.0_patch_16_08_09.zip-d
sdk/ESP8266_NONOS_SDK/lib
```


Программирование с Lua

Lua - сильный язык сценариев, который доступен на окружающей среде ESP8266. Самое популярное внедрение Lua для ESP8266 известно как встроенное микропрограммное обеспечение NodeMCU Lua и доступно в его хранилище GitHub. Некоторые люди обмениваются фразой NodeMCU для самого встроенного микропрограммного обеспечения Lua, так заботьтесь, чтобы удостовериться, что Вы понимаете включенный контекст.

Строит из встроенного микропрограммного обеспечения, может быть загружен непосредственно, как может источник.

Как только у Вас есть копия встроенного микропрограммного обеспечения, Вы можете высветить его, используя Ваши любимые инструменты вспышки.

Мы не будем описывать сам язык Lua в этой книге. Есть превосходные книги, уже написанные на Lua, и также ссылки и обучающие программы могут быть найдены в Интернете. Скорее давайте посмотрим на специфические особенности управления Lua на ESP8266.

Принятие Вас высветило ESP8266 с Lua, Вы должны будете соединить последовательный терминал с ним, чтобы взаимодействовать с ним. Последовательная скорость передачи в бодах должна быть 9600.

См. также:

- GitHub:nodemcu/nodemcu-firmware
- Встроенное микропрограммное обеспечение NodeMCU Wiki
- Справочное Lua 5.1 руководство
- lua.org
- Форум NodeMCU ua по ESP8266.com
- nodemcu-unofficial-faq

ESPlorer IDE

IDE ESPlorer - среда разработки для того, чтобы создать приложения Lua для ESP8266.

См. также:

- Домашняя страница Esplorer
- электронная книга:Начало работы с IDE ESPlorer
- GitHub:4refr0nt/ESPlorer

GPIO с Lua

У Lua есть понятие 13 логических булавок, определенных 0-12. Эти булавки наносят на карту к булавкам GPIO ESP8266 следующим образом:

Булавка Lua число	Булавка ESP8266	NodeMCU devKit	Булавка Lua число	Булавка ESP8266	NodeMCU devKit
0	GPIO16	D0	7	GPIO13	D7
1	GPIO5	D1	8	GPIO15	D8
2	GPIO4	D2	9	GPIO3	D9
3	GPIO0	D3	10	GPIO1	D10
4	GPIO2	D4	11	GPIO9	SD2

5	GPIO14	D5	12	GPIO10	SD3
6	GPIO12	D6			

Булавки GPIO, известные как GPIO6, GPIO7 и GPIO8 на ESP8266, не выставлены.

Прежде, чем читать или написать от булавки GPIO, мы должны сообщить Lua способа той булавки. Наш выбор введен, произведен или прерывает. Для входа мы можем также объявить, является ли вход подтягиванием или плаванием.

Синтаксис заявления:

```
gpio.mode (булавка, способ, усилие)
```

Как только способ был установлен, если он введен, мы можем назвать `gpio.read ()`, чтобы прочитать стоимость от булавки и `gpio.write ()`, чтобы написать стоимость булавке.

Если мы хотим быть вызванными перерывом на булавке (вход), мы можем использовать `gpio.trig ()`.

WiFi с Lua

Организация сети с Lua

Программирование с основным

См. также:

- Основной ESP8266

Интеграция с веб-приложениями

REST Services

Понятие распределенных вычислений датируется много десятилетий. Идея, что один компьютер мог выполнить обслуживание от имени другого, является классическим понятием. Взгляды состоят в том, что работа могла быть распределена через системы, данные могли быть централизованы, или специальные системы могли выполнить специализированные роли. За эти годы много форм распределенных вычислений попробовали. Они включают серверы гнезда, удаленные вызовы процедуры (RPC), Systems Network Architecture (SNA), Distributed Computing Environment (DCE), веб-сервисы и других.

Сегодня (2015), нынешнее должностное лицо распределенных вычислительных протоколов и технологии ОТДЫХ. ОТДЫХ - простой протокол, который усиливает существующий Hyper Text Transport Protocol (HTTP), используемый в качестве транспорта между браузерами и веб-серверами. Этот протокол был, строят, чтобы позволить браузеру запрашивать данные от отдаленной файловой системы, принятой веб-сервером. Это предоставляет HTTP «команды», которые включают, ПОЛУЧАЮТ, ОТПРАВЛЯЮТ, ПОМЕЩАЮТ и другие.

Понятие позади ОТДЫХА - больше несчастного случая, чем дизайн. Перецели ОТДЫХА HTTP как коммуникационный трубопровод от клиента к серверу, где клиент обращается с просьбой ОТДЫХА и сервером, предлагают услугу ОТДЫХА. От сети

перспектива, это «смотрит» точно так же, как взаимодействие браузера/Веб-сервера, но оба конца принимают решение договориться о формировании и интерпретации коммуникации.

Когда мы добавляем ESP8266 в соединение, наше желание двойное. Мы хотим, чтобы ESP8266 был в состоянии быть клиентом внешним поставщикам услуг ОТДЫХА, и мы хотим, чтобы ESP8266 был целью клиентов, обращающихся с просьбами ОТДЫХА. С точки зрения партнера это должно не знать, что взаимодействует с ESP8266 по сравнению с любым другим вычислительным устройством.

Протокол ОТДЫХА

ОСТАЛЬНОЕ протокол построено сверху HTTP.

См. также:

- RFC7230 –HTTP/1.1 – Синтаксис сообщения и Направление
- HTTP: протокол каждый веб-разработчик должен знать – часть 1

ESP8266 как клиент ОТДЫХА

Чтобы ESP8266 был клиентом ОТДЫХА, он должен построить и передать запросы HTTP поставщику услуг. Это будет включать строительство заголовки HTTP, передачу данных в форме, ожидаемой поставщиком (например, JSON, XML или другое текстовое представление) и обработка ответа от поставщика, который может включать интерпретацию полученного полезного груза.

Передать запрос ОТДЫХА состоит из двух частей. Сначала это открывает соединение по протоколу TCP для партнера и затем передает соответствующие данные HTTP вниз та связь. Первая часть легка, вторая часть - больше проблемы. Мы могли прочитать и понять спекуляцию HTTP и построить часть запроса частью, но это должно будет быть сделано для каждого проекта, который хочет использовать технологию клиента ОТДЫХА. То, что было бы лучше, - то, если у нас была библиотека, которая «знает», как обратиться с хорошо сформированными просьбами ОТДЫХА, и мы просто усилили его существующие функции.

Создание запроса ОТДЫХА, используя Мангусту

Используя API Мангусты, мы можем довольно легко отправить запрос ОТДЫХА и работать с ответом. История высокого уровня должна инициализировать Мангусту с `mg_mgr_init ()`, просить связь с остальными поставщик услуг с `mg_connect ()`, связать связь, как являющуюся ориентированным HTTP, и затем начать обрабатывать события. Первым событием, которое возвратится, будет событие `MG_EV_CONNECT`, указывающее, что мы - теперь связанная сеть. Оттуда мы можем использовать mg_printf (), чтобы отправить остальным запрос. То, когда остальные сотрудничают, отвечает, мы получим событие `MG_EV_HTTP_REPLY`, и мы закончили наше соединение запроса/ответа.

ESP8266 как поставщик услуг ОТДЫХА

Чтобы ESP8266 был поставщиком услуг ОТДЫХА, в основном средства, что это должно играть роль веб-сервера и ответить на запросы веб-сервера. Однако в отличие от простого веб-сервера, который просто восстанавливает и посылает содержание файла как функцию пути на URL, вероятно, что остальные, поставщик услуг выполнит некоторое вычисление, когда запрос клиента HTTP прибудет. Например, если бы мы приложили датчик температуры к GPIOs ESP8266, когда запрос ОТДЫХА прибывает, ESP8266 мог читать, текущая температура оценивают и передают закодированный результат обратно как ответ на запрос.

Быть веб-сервером в основном означает слушать на порте TCP и когда связи прибывают, интерпретируя данные, полученные как протокол HTTP. Это было бы большой работой над проектом основанием проекта, но к счастью есть много пред - письменные библиотеки, которые выполняют эту задачу для нас и всего, чем мы должны коснуться нас, экспертиза любых параметров, переданных с запросом и выполнением логики, которой мы желаем выполняемый, когда когда-либо новый запрос получен.

Библиотека, такая как ESP8266WebServer идеально

подошла бы для этой задачи.См. также:

- ESP8266WebServer

Tasker

Tasker - применение Android, которое автоматизирует и задачи сценариев, которые будут выполнены на устройстве на базе Android. Используя Tasker мы можем создать задачу, которая определена как последовательность команд и действий, которые будут выполнены. Затем мы можем создать Профиль, который наносит на карту событие, которое, когда оно обнаружено, выполняет задачу. Хотя это полезно, как это касается ESP8266? Предположите, что событием, которое происходит, является ESP8266, посылая сообщение в Ваш телефон. С тем понятием ESP8266 может, эффективно, вызвать что-либо, что можно было бы быть в состоянии сделать с таким телефоном. Например, это могло бы сделать телефонный звонок, послать SMS-сообщение или захватить фотографию.

См. также:

- Домашняя страница Tasker
- YouTube:Tasker 101 обучающая программа

AutoRemote

Следуя за нашим обсуждением Tasker выше, у нас теперь есть допуск. Кажется, что у Tasker нет способности прислушаться к базирующимся событиям и сообщениям поступающего TCP/IP. Однако, потому что Tasker расширяем, и разработчики могут

напишите программные расширения для него, Tasker может быть увеличен. Одно такое увеличение - плагин AutoRemote.
Используя тот плагин, сообщение TCP/IP может тогда послать и получить AutoRemote, который может тогда действовать как источник событий для Tasker.

С AutoRemote, настроенным как плагин Tasker, мы можем настроить его, чтобы прислушаться к запросам HTTP. Это заставляет AutoRemote слушать на номере порта TCP `1817`. Данные, к которым это прислушивается, являются запросом HTTP. Например:

```
http://<звонят IP>:1817/sendmessage? message=1
```

И с Tasker и с AutoRemote установил, он все еще не будет прислушиваться к поступающим сообщениям WiFi по местной окружающей среде WiFi, если мы не будем связанным Интернетом. Мы должны управлять Задачей Tasker, названной «AutoRemote WiFi».

Например, в Tasker:

1. Создайте новый профиль, вызванный Событием → Система → Ботинок Устройства
2. Создайте Новую Задачу, связанную с профилем
3. Добавьте действие от Вставного Wi-Fi AutoRemote
4. В конфигурации для действия проверьте «Обслуживание Wi-Fi»

То, что это сделает, начать Обслуживание Wi-Fi каждый раз, когда устройство (Android) загружает.

К сожалению, у AutoRemote есть серьезный недостаток. Это не позволяет Tasker передавать ответ обратно в оригинальном запросе ОТДЫХА, который мог бы содержать данные, которые могли использоваться. Например, если мы хотим использовать AutoRemote, чтобы отправить запрос, который возвратил текущее местоположение GPS, которое просто не возможно.

Когда запрос AutoRemote прибывает, он устанавливает много переменных в окружающей среде Tasker, которая может использоваться в качестве параметров к задачам Tasker. Они включают:

- %armessage
- %arpar ()
- %arcomm
- %artime
- %arfiles
- %arsenderbtmac
- %arsenderid
- %arsenderlocalip
- %arsendername
- %arsenderpublicip

- %arsendertype
- %arvia

- Wi-Fi Видит так же:
  - Домашняя страница AutoRemote

DuckDNS

Я ожидаю, что в большинстве народных зданий есть точка доступа WiFi, которую или непосредственно или через модем, соединяет с Интернетом. Так как точка доступа WiFi предлагает местную сеть, к которой может присоединиться ESP8266, мы теперь видим, что ESP8266 может достигнуть внешнего мира через точку доступа. Однако что относительно перемены? Что, если мы хотим, чтобы клиент в Интернете достиг нашего ESP8266. Как мы могли достигнуть этого?

Если мы смотрим на вышеупомянутую диаграмму (весь составленный IP-адрес), мы видим, что ESP8266 знает свой собственный IP-адрес как 192.168.1.2. Однако это не может быть «разделено» с Интернетом, поскольку это - местный адрес и не глобальный IP-адрес. То, что должно было бы быть разделено, является IP-адресом точки доступа, как замечено в Интернете.

Один способ достигнуть, который является с помощью поставщика услуг, такого как DuckDNS. Это бесплатное обслуживание позволяет Вам регистрировать имя. Ваше устройство (обычно PC) периодически отправляет запрос к веб-сайту DuckDNS, говорящему, «Привет … я

здесь!». Обратный адрес того запроса всегда - IP-адрес Вашей точки доступа, связанной с Интернетом, и следовательно DuckDNS изучает Ваш внешний адрес. Позже, кто-то (возможно, третье лицо) может спросить, «Что является IP-адресом» имени, которое Вы зарегистрировали, и тот адрес сделан доступным. По существу DuckDNS действует как брокер в реальном времени логических имен к IP-адресам.

Если Вы обеспокоены, что «некоторый страшный человек» может узнать, что IP-адрес Вашей точки доступа … тогда не использует DuckDNS. Однако для большинства нас, наш модем/маршрутизатор/точка доступа препятствует тому, чтобы поступающий трафик достиг нас и по существу блокирует что-либо, что мы не хотим. Но ждите …, это также не заблокирует запросы к ESP8266? Ответ, «да он будет», который является, почему Вы должны определить перенаправление портов. Перенаправление портов функция Вашего модема/маршрутизатора/точки доступа, который говорит, что, когда запрос прибывает для данного местоположения порта, автоматически отправьте его IP-адресу в Вашей местной сети …, например, сетевой адрес Вашего ESP8266.

https://www.duckdns.org/update? domains=XXX&token=XXX&ip =

Мобильные приложения

Blynk

См. также:

- Домашняя страница Blynk

Типовые отрывки

Есть времена, когда все, в чем мы нуждаемся, является отрывком кодекса, который мы можем скопировать, чтобы достигнуть задачи. Здесь мы представляем ряд таких отрывков в том мае использования просто, копируя и приклеивая их.

Формирование соединения по протоколу TCP

Здесь мы видим отрывок кода, который может использоваться, чтобы установить связь TCP/IP.

```
#define REMOTE_PORT 80
#define REMOTE_IP
«216.58.218.206»
```

```
структура espconn conn1;
esp_tcp tcp1;

пустота connectCB (пустота
   *аргумент) {os_printf
   («У нас есть connected
   \n»);
}

пустота errorCB (пустота *аргумент,
   sint8 допускают ошибку)
   {os_printf («У нас есть ошибка:
   %d\n», допускают ошибку);
}

пустота
   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 («Мы попросили связь!»);
}
```


Примеры приложения

Чтение и рассматривание заявлений приложения являются хорошей практикой. Это позволяет Вам изучать то, что другие написали и видят, можете ли Вы понять каждое из заявлений и процесса выполнения программы в целом.

Образец - Освещает светодиод на основе прибытия дейтаграммы UDP

В этом образце у нас будет ESP8266, становятся станцией WiFi и соединяются. Это начнет прислушиваться к поступающим дейтаграммам и если первый байт полученных данных будет характером «1», то это осветит светодиод. Если характер будет «0», то он погасит светодиод.

Вот полный код приложения со следующим комментария:

```
#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

МЕСТНАЯ структура espconn conn1;
МЕСТНЫЙ esp_udp udp1;

МЕСТНАЯ пустота recvCB (пустота *аргумент, случайная работа *pData,
короткое целое без знака len);
МЕСТНАЯ пустота eventCB (System_Event_t *событие);
МЕСТНАЯ пустота setupUDP ();
МЕСТНАЯ пустота initDone ();

МЕСТНАЯ пустота recvCB (пустота *аргумент, случайная
   работа *pData, короткое целое без знака len)
   {структура espconn *pEspConn = (структура espconn
   *) аргумент; os_printf («Полученные данные!! -
   длина = %d\n», len);
   если      (len == 0 || (pData[0]! = '0' &&
      pData[0]! = '1')) {возвращение;
   }
   интервал v =
   (pData[0] == '1');
   GPIO_OUTPUT_SET
   (LED_GPIO, v);
}//Конец recvCB

МЕСТНАЯ пустота initDone ()
   {wifi_set_opmode_current
   (STATION_MODE); структура
   station_config stationConfig;
   strncpy (stationConfig.ssid,
   «myssid», 32);
   strncpy        (stationConfig.password,
   «пароль»,                          64);
   wifi_station_set_config_current
   (&stationConfig);  wifi_station_connect
   ();
```



```
}//Конец initDone

МЕСТНАЯ пустота
   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
   («Прислушивающийся data
   \n»);
}//Конец setupUDP


МЕСТНАЯ пустота eventCB
   (System_Event_t *событие)
   {выключатель (событие->
   событие) {
   случай EVENT_STAMODE_GOT_IP:
      os_printf («IP: %d. % d. % d. % d\n», IP2STR (&event->
      event_info.got_ip.ip)); setupUDP ();
      разрыв;
   }
}//Конец eventCB

пустота user_rf_pre_init (пустота) {
}

пустота user_init (пустота)
   {uart_init (BIT_RATE_115200,
   BIT_RATE_115200);
   //Набор GPIO15 как GPIO прикрепляет
   PIN_FUNC_SELECT (PERIPHS_IO_MUX_MTDO_U,
   FUNC_GPIO15);
   //Назовите «initDone», когда ESP8266
   инициализирует system_init_done_cb
   (initDone); wifi_set_event_handler_cb
   (eventCB);
}//Конец user_init
```

Контроль начинает в `user_init ()` функцию где мы установка бод UART. В этом примере мы выбрали GPIO15 в качестве нашей булавки продукции, таким образом, мы наносим на карту функцию физической булавки под названием «`MTDO_U`» к логической функции «`GPIO15`». Мы регистрируем функцию, `вызванную initDone ()`, чтобы быть названными, когда инициализация устройства завершена, и мы также регистрируем функцию, `вызванную eventCB ()`, чтобы быть названными, когда события WiFi прибывают, указывая на изменение состояния.

Так как эти пункты - установка, мы возвращаем контроль назад к OS. Мы ожидаем быть призванными обратно через `initDone ()`, когда устройство будет полностью прочитано для работы. В `initDone ()` мы

определяем нас как Станцию Wi-Fi и называем точку доступа с ее паролем, который мы хотим использовать. Наконец мы просим связь с точкой доступа.

Если все будет подходить, то мы будем связаны с точкой доступа и затем ассигнованы IP-адрес. Оба из них приведут к событиям, производимым, который заставит нас просыпаться в `eventCB ()`. Единственным событием мы интересуемся наблюдением, является распределение IP

адрес. Когда мы уведомлены относительно этого, мы вызываем функцию, `вызванную setupUDP ()`, чтобы инициализировать наш UDP слушающая окружающая среда.

В `setupUDP ()`, мы создаем `структуру espconn` блок управления, определенный для UDP и настроенный, чтобы послушать на нашем выбранном порте 25 867. Мы также регистрируем получить отзыв к функции `recvCB ()`. Это назовут, когда новые данные прибудут. На данном этапе вся наша установка закончена, и нам соединили устройство с сетью WiFi, слушающей на порте UDP 25867 для дейтаграмм.

Когда дейтаграмма прибывает, мы просыпаемся в `recvCB ()` переданный в дейтаграммных данных. Мы проверяем, что у нас на самом деле есть данные и что это - хороший … в противном случае, мы немедленно заканчиваем отзыв.

Наконец, мы смотрим на первый характер данных и, на основе его стоимости, изменяем ценность продукции GPIO. Физический GPIO телеграфирован к светодиоду и резистору.

Если характер `'1'` передан, продукция `GPIO15` идет высоко и светодиодные индикаторы. Если стоимость характера `'0'`, продукция `GPIO15` идет низко, и светодиод погашен.

Образец - Сверхзвуковое измерение расстояния

SR-04 HC - сверхзвуковой датчик измерения расстояния.

Пошлите минимум 10us пульс к Аккуратному (низко к высоко к низкому). Позже, Эхо пойдет низко/высоко/низко. Время, что Эхо высоко, является временем, оно берет звуковой импульс, чтобы достигнуть бэкенда и прийти в норму.

Скорость звука составляет 340,29 м/с (340.29 * 39,3701 дюймов/секунда). Назовите этот Vsound.

Если Techo - время для ответа эха тогда d = (Techo * Vsound) / 2.

Также уравнением для ожидаемых длин Techo дают:

Techo = 2d/Vsound

Например:

Расстояние	Время
1 см	* 0.01 / 340 = 0,058 национальной безопасности = 59 2 usecs

10 см	* 0.1 / 340 = 0,59 национальной безопасности = 590 2 usecs
1 м	* 1 / 340 = 5,9 национальной безопасности = 5900 usecs 2 (5,9 национальной безопасности)

Поскольку ответ `Эха` - 5-вольтовый сигнал, жизненно важно уменьшить это до 3.3 В для входа в в ESP8266. Сепаратор напряжения будет работать. Булавки на устройстве:

- `Vcc` – входное напряжение составляет 5 В.
- `Аккуратный` – Пульс (низко к высокому), чтобы вызвать передачу … минимум 10usecs.
- `Эхо` – Импульсы низко к высоко к низкому, когда эхо получено. Предупреждение, это - 5 В продукция.
- `Gnd` – земля.

Чтобы вести это устройство, мы должны использовать две булавки на ESP8266, который мы логически назовем `Аккуратным` и `Эхо`. В моем дизайне я установил `Аккуратный` быть GPIO4 и `Эхом`, чтобы быть GPIO5.

Наш дизайн для применения не будет включать организации сети, но это должно быть прямо заднице это по мере необходимости. Мы будем установка таймер, который стреляет однажды секунда, которая является, как часто мы хотим провести измерения. Когда таймер проснется, мы будем пульсировать Аккуратные от низко до высокого и назад к низкому холдингу высоко в течение 10 микросекунд. Мы теперь сделаем запись времени и начнем получать голоса булавки Эха, ждущей, чтобы она пошла высоко. Когда это сделает, мы сделаем запись времени снова, и вычитание того от того скажет нам как

долго это брало звук, чтобы прийти в норму. От этого мы можем вычислить расстояние до объекта. Если никакой ответ не будет получен в 20 национальной безопасности, то мы предположим, что не было никакого объекта обнаружить. Мы тогда зарегистрируем результат к Последовательному пульту.

Программу в качестве примера, которая выполняет этот дизайн, показывают затем:

```
#define
TRIG_PIN
4 #define
```

```
ECHO_PIN

5

os_timer_t myTimer;

пустота user_rf_pre_init (пустота) {
}

пустота timerCallback
   (пустота *pArg)
   {os_printf
   («Тиканье! \n»);
   GPIO_OUTPUT_SET
   (TRIG_PIN, 1);
   os_delay_us (10);
   GPIO_OUTPUT_SET
   (TRIG_PIN, 0);
   uint32 val = GPIO_INPUT_GET
   (ECHO_PIN); в то время как
   (val == 0) {
      val = GPIO_INPUT_GET (ECHO_PIN);
   }
   uint32 startTime =
   system_get_time (); val =
   GPIO_INPUT_GET (ECHO_PIN);
   в то время как (val == 1 && (system_get_time () -
      startTime) <(20 * 1000)) {val = GPIO_INPUT_GET
      (ECHO_PIN);
   }
   если (val == 0) {
      дельта uint32 = system_get_time
      () - startTime;//Вычисляют
      расстояние в cm.
      расстояние uint32 = 340.29 * 100 * дельта /
      (1000 * 1000 * 2); os_printf («Расстояние: %d
      \n», расстояние);
   } еще {
      os_printf («Никакое эхо! \n»);
   }
}//Конец timerCallback

пустота user_init (пустота)
   {uart_init (BIT_RATE_115200,
   BIT_RATE_115200);
//    Установка ultrasonics прикрепляет как GPIO setAsGpio (TRIG_PIN);
setAsGpio (ECHO_PIN); setupBlink (15);
//    Установите более аккуратную булавку быть дефолтом низкий
GPIO_OUTPUT_SET (TRIG_PIN, 0); os_timer_setfn (&myTimer, timerCallback,
ПУСТОЙ УКАЗАТЕЛЬ); os_timer_arm (&myTimer, 1000, 5);
      }//Конец user_init
```

Как только это было написано и проверено, мы сделаем второй проход в загадке, но на этот раз используя перерыв, чтобы вызвать ответ на эхо.

См. также:

- GPIOs

Образец - сканер WiFi

Сканер WiFi - применение, которое периодически просматривает для доступных сетей WiFi и показывает их пользователю. В нашем дизайне мы будем периодически просматривать и помнить набор сетей, которые мы находим. Когда мы выполним перепросмотры, мы проверим, чтобы видеть, является ли каждая из расположенных сетей сетью, мы ранее видели и, в противном случае перечисляем ее пользователю. Мы будем также держать «прошлое замеченное» время для каждой сети и если сеть не замечалась в течение минуты, то мы забудем об этом таким образом, что, если это появляется снова, мы еще раз перечислим ее пользователю.

Чтобы проиллюстрировать наш дизайн, мы сломаем решение во многие части. Первая часть должна будет зарегистрировать функцию обратного вызова, которую называют каждые 30 секунд. Этот отзыв будет ответственен за требование просмотра WiFi, используя `wifi_station_scan ()`. Это берет функцию обратного вызова, которая саму будет призвана, когда просмотр будет завершен.

Когда просмотр закончит, у нас будет новый список обнаруженных сетей. Мы будем идти этот список и для каждой обнаруженной сети, определять, видели ли мы его прежде. Если мы будем иметь, то мы обновим в прошлый замеченный раз. В противном случае мы добавим его к списку ранее замеченных сетей и зарегистрируем его пользователю.

Второй отзыв таймера будет бежать однажды минута и будет идти список ранее замеченных сетей. Если какой-либо из них будет старше, чем минута, то мы удалим их.

См. также:

- Просмотр для точек доступа

Образец - Работающий с микро SD-картами

Микро SD-карта - маленькое портативное устройство хранения данных, которое может принять гигабайты данных. С помощью адаптера микро SD-карта может быть усилена вместе с прочитанным обеспечением ESP8266 и написать доступ к данным, которые сохраняются через перезапуски ESP8266.

Образец - Игра аудио от события

В этом образце мы хотим, чтобы событие было обнаружено ESP8266, который, когда это происходит, заставляет аудиофайл играться.

Образец - изменчивый свет настроения

NeoPixels - светодиоды, которые ведет единственная линия данных скоростной передачи сигналов. Большинство NeoPixels имеет +ve и основывает источник напряжения, а также линию данных для входа и а

линия данных для продукции. Продукция Одного неопикселя может быть подана во вход следующего, чтобы произвести ряд таких светодиодов. Входные данные к светодиоду - поток 24 битов закодированных данных, которые должны интерпретироваться как 8 битов для красного канала, 8 битов для зеленого канала и 8 битов для синего канала. У каждого канала может таким образом быть ценность светимости между 0 и 255. Смешивая ценности для каждого из каналов вместе, Вы можете окрасить светодиод к любому цвету, который Вы можете выбрать. После отправки в потоке 24 битов, если мы посылаем во втором потоке 24 битов быстро после первого потока, второй поток «выдвинут» до следующего светодиода в цепи. Это может быть повторено, насколько желаемый. Если мы делаем паузу, посылая в данных, текущую стоимость «запирают» в место, и каждый светодиод их помнит свою собственную стоимость.

Время сигналов данных для этих светодиодов может быть довольно хитрым, но к счастью великие умы уже построили фантастические библиотеки для вождения их правильно, таким образом, мы не должны интересоваться этим временем низкого уровня и можем вместо этого сконцентрироваться на разработке интересных проектов и целей, в которые могут быть помещены светодиоды. Есть много различных типов этих светодиодов с наиболее распространенными, известными как WS2811, WS2812 или PL9823.

В окружающей среде JavaScript Espruino может быть найден метод, названный `neopixelWrite ()`. Это берет два параметра. Первой является булавка ESP8266 GPIO, которая будет использоваться в качестве источника сигналов к светодиодам. Именно к этой булавке светодиоды должны быть телеграфированы. Булавка, используемая для вывода данных от ESP8266 до NeoPixels, должна быть установлена

в способе продукции GPIO. Например:

```
pinMode (булавка, «продукция»);
```

Второй параметр - множество целых чисел. Ценности множества должны поставляться в группах 3 соответствий 3 каналам красного, зеленого и синего цвета. Например, если бы у нас был Один неопиксель, связанный с GPIO4 на ESP8266, и мы хотели установить его в полностью красный, то мы могли бы закодировать:

```
neopixelWrite (новая Булавка (4), [255, 0, 0]);
```

Если бы мы хотели, чтобы следующий пиксель был зеленым, в то время как первое красное, то мы могли бы написать:

```
neopixelWrite (новая Булавка (4),
[255, 0, 0,                          0, 255, 0]);
```

Снова, нет никакого очевидного предела количеству светодиодов, которые мы можем натянуть вместе.

Теперь, когда мы видим, что можем установить яркость и цвет светодиода, давайте посмотрим на то, как мы могли бы проектировать некоторый кодекс, чтобы сделать что-то. Давайте предположим, что мы имели ряд из 16 светодиодов и хотели сделать их тем же цветом …, мы могли бы определить функцию следующим образом:

```
функционируйте цветные
  (красный, зеленый, синий)
  {данные о варе = [];
  для (вар i=0; я
    <16; я ++)
    {data.push
    (зеленый);
    (красный)
    data.push;
```



```
    (синий) data.push;
  }
  esp8266.neopixelWrite (NodeMCU.D2, данные);
}
```

Если мы вызовем эту функцию с правильными красными, зелеными и синими ценностями, то она установит светодиодную последовательность правильно.

Теперь давайте пойдем один шаг вперед. Предположите, что мы получили запрос сети REST, который описал цвет, который мы хотим, чтобы светодиоды показали. Заполнять заявление может быть:

```
вар особенно =
требует («ESP8266»);
вар NodeMCU = {
  //D0:
```

```
  новая
  Булавка
  (16), D1:
  новая
  Булавка
  (5),
  D2: новая Булавка (4),
  D3: новая Булавка (0),
  D4: новая Булавка (2),
  D5: новая Булавка (14),
  D6: новая Булавка (12),
  D7: новая Булавка (13),
  D8: новая Булавка (15),
  D9: новая Булавка (3),
  D10: новая Булавка (1)
};

pinMode (NodeMCU.D2, «продукция»);

функционируйте цветные
  (красный, зеленый, синий)
  {данные о варе = [];
  для (вар i=0; я
    <16; я ++)
    {data.push
    (зеленый);
    (красный)
    data.push;
    (синий)
    data.push;
  }
  esp.neopixelWrite (NodeMCU.D2, данные);
}

функционируйте beServer () {
  вар http = требует («http»);
  вар httpServer = http.createServer (функция (запрос,
    ответ) {печать (запрос);
    вар partsOfUrl =
    request.url.split (»?»); если
    (partsOfUrl.length> 1) {
      варианты вара =
      partsOfUrl[1].split ('& '); вар
      optionsObj = {};
      для (вар i=0; я <options.length; я
        ++) {вар splitEquals = варианты
        [я] .split (' = ');
        optionsObj [splitEquals[0]] = splitEquals[1];
      }
      печать («Финал obj»: + JSON.stringify
      (optionsObj)); если (optionsObj.color! ==
      пустой указатель) {
        вар, красный = parseInt (optionsObj.color.substr
        (0,2), 16); вар, зеленый = parseInt
        (optionsObj.color.substr (2,2), 16); вар, синий =
        parseInt (optionsObj.color.substr (4,2), 16);
```

```
        печать («красный: «+ красный +», зеленый: «+ зеленый
        +», синий»: + синий); цветные (красный, зеленый,
        синий);
      }
    }
    печать («URL
    результата =» +
    URL);
    response.writeHead
    (200, {
      «Контроль доступа
    Позволяет Происхождение»:
    «*»});
    response.end (»»);
  });//Конец по новому запросу браузера

  httpServer.listen (80);
  печать («Теперь быть
сервером HTTP!»);}//Конец
beServer

вар ssid     = «ssid»;
вар   пароль = «пароль»;

//Соединитесь с Wi-Fi
вара точки доступа =,
требуют («Wi-Fi»);
печать («Соединяющийся с точкой доступа».);
wifi.connect (ssid, пароль, пустой указатель,
  функция (допускают ошибку, ipInfo) {если
  (допускают ошибку) {
    печать («Ошибка при соединении с
    точкой доступа».);вернуть;
  }
  вар ESP8266 = требует («ESP8266»);
  печать («Соединяются, говорит, что мы теперь связаны!!!»);
  печать («Стартовый веб-сервер по http://» +
ESP8266.getAddressAsString (ipInfo.ip) + «:80»);
  beServer ();
});
```

Когда это применение бежит, оно соединяется с местной точкой доступа WiFI и затем начинает прислушиваться к поступающим запросам ОТДЫХА. У запроса отдыха, как ожидают, будет параметр вопроса в конце с форматом `color=value`, где стоимость закодирована как 6 шестнадцатеричных знаков, соответствующих цвету. Наконец, мы можем написать веб-страницу, которая представит палитру цветов и, когда мы выберем цвет, отправляем запрос ОТДЫХА к ESP8266, чтобы осветить светодиоды соответственно. Вот типовая веб-страница, чтобы достигнуть этой задачи:

```
<!
HTML
DOCTY
PE>
<HTML
>
<голова>
```

```
<meta кодировка =
«ISO-8859-1»>
<название>
Выбранные
светодиодные цвета
</название>
<связывают href = рэл
   «http://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.11.2/jquery-
   ui.min.css» = тип «таблицы стилей» = «text/css»/>

<сценарий
src =
«http://cdnjs.cloudflare.com/ajax/libs/require.js/2.1.15/require.min.js»>
</сценарий> <связывают рэл ='stylesheet' href ='spectrum.css'/>
<
  с
  ц
  е
  н
  а
  р
  и
  й
  >
  т
  р
  е
  б
  у
  е
  т
```

Страница 292

```
      .config ({
        baseUr
        l:
        «src»,
        пути:
        {
          «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 -
i.min»,
        },
        прокладк
          a:
          {«jQue
          ry-
          ui»: {
```

```
            deps:
            [«jQuery»],
            экспорт:
            'jQueryUI'
          }
        }
      //Кон
      ец
      прокл
      адок}
      );
  потребуйте ([«jQuery», «спектр», «jQuery-ui»],
    функция ($) {$ (функция () {
      вар
      allowHttp =
      верный; $ (»
      #flat») .spe
      ctrum ({
        квартира:
        верный,
        preferredForma
        t: «rgb»,
        двиньтесь:
        функция (цвет)
        {
          если
            (allowHt
            tp)
            {allowHt
            tp =
            ложный;
            $ .ajax
            ({
              URL:
              «http://192.168.1.10
              », данные: {
                цвет: color.toHex ()
              },
              успех:
                функция ()
                {allowHttp
                = верный;
              },
              ошибка:
                функция
                ()
                {allowHt
                tp =
                верный;
              }
            });
          }
        },
        showInput
        : верный,
        showButto
        ns:
        ложный
      });
    });//Конец на
  грузе});//Конец
  требуют
</сценарий>
```

```
</голова>
<тело>
   <id отделения = «квартира» разрабатывают =
«width:500px; высота: 500 пкс»;> </отделение>
</тело>
</HTML>
```

Конечный результат, как замечено на веб-странице смотрит следующим образом:

Выбор нового цвета вызывает данные к посланному в ESP8266, который окрашивает светодиоды соответственно с полным конечным результатом, являющимся способностью изменить свет настроения светодиодной последовательности.

Образец - Улучшающий организацию сети

Предположите, что Вам дают фантастическое применение ESP8266, и Вы устанавливаете его на своем устройстве. Вероятность - то, что это будет использовать основанную на WiFi организацию сети. Теперь прибывает интересный вопрос, как Вы улучшаете устройство?

Типовые библиотеки

Есть времена, когда обычно используемые функции могут быть захвачены и снова использованы много раз. В этом разделе описываются

просто такой набор функций, которые были собраны. Источник для этих функций был помещен в GitHub в <местоположение, которое будет обеспечено>.

Функции, когда они собраны, помещены в библиотеку, названную `libcommon.a`. Это может тогда быть связано в Вашем Makefile так, чтобы нерешенные ссылки на эти функции могли быть удовлетворены.

Заголовочный файл, названный «`common.h`», является всем, что нужно добавить в Ваши собственные заявления.

Список функции

authModeToString

Учитывая AUTH_MODE, возвратите представление последовательности способа.

```
случайная работа *authModeToString (способ AUTH_MODE)
```


checkError

Проверьте код возврата на ошибку.

```
пустота checkError (sint8 допускают ошибку),
```

Проверьте `допускать ошибку` кодекс на ошибку и если это один, зарегистрируйте его.

delayMilliseconds

Задержка сроком на миллисекунды.

```
пустота delayMilliseconds (uint32 миллисекунды)
```

Параметры `миллисекунд` - количество миллисекунд, чтобы задержаться перед возвращением.

dumpBSSINFO

Свалите случай структуры bss_info к регистрации.

```
пустота dumpBSSINFO (структура bss_info *bssInfo)
```

dumpEspConn

Свалка к регистрации расшифрованное представление `структуры espconn`.

```
пустота dumpEspConn (структура espconn *pEspConn)
```

dumpRestart

Свалите информацию о перезапуске к регистрации.

```
пустота dumpRestart ()
```

См. также:

- Обработка исключений

dumpState

Свалите станционное государство WiFi к регистрации.

```
пустота dumpState ()
```

См. также:

- Ошибка: Справочный источник, не найденный
- Ошибка: Справочный источник, не найденный
- Ошибка: Справочный источник, не найденный
- Ошибка: Справочный источник, не найденный
- Ошибка: Справочный источник, не найденный
- Ошибка: Справочный источник, не найденный
- Ошибка: Справочный источник, не найденный

Страница 295

errorToString

Учитывая код ошибки, возвратите представление последовательности из него.

```
случайная работа *errorToString (sint8 допускают ошибку),
```

eventLogger

Напишите событие WiFi регистрации.

```
пустота eventLogger (System_Event_t *событие)
```

Мы можем зарегистрировать эту функцию как отзыв для события WiFi.

Напишите данные событий регистрации.

См. также:

- Обработка событий WiFi

eventReasonToString

Преобразуйте причину событий для представления последовательности.

```
случайная работа *eventReasonToString (международная причина)
```

Некоторые отзывы WiFi событий могут возвратить причину стоимость, которая является кодированием причины что что-то подведенное. Эта функция возвращает представление последовательности международного кодекса стоимости.

flashSizeAndMapToString

Возвратите представление последовательности размера вспышки и карты.

```
случайная работа *flashSizeAndMapToString ()
```

setAsGpio

Установите булавку использоваться в качестве GPIO.

```
пустота setAsGpio (uint8 булавка)
```

Установите GPIO, поставляемый как

`булавка` быть функцией GPIO.См.

также:

- GPIOs
- GPIOs

setupBlink

Установка мигание Вовлекла данную булавку.

```
пустота setupBlink (uint8 blinkPin)
```


`blinkPin` параметр - булавка, чтобы использовать для мигания.

toHex

Преобразуйте множество байтов к шестнадцатеричной последовательности.

```
uint8 *toHex (uint8 *ptr, международный размер, uint8 *буфер)
```

Преобразуйте байты, на которые указывает `ptr` для байтов `размера` в шестнадцатеричную последовательность. `Буферный` параметр будет то, где результат будет сохранен. Это должно быть 2 * размер + 1 байт в

длине (или больше). Каждый байт - 2 шестнадцатеричных знака плюс единственный терминатор ПУСТОГО УКАЗАТЕЛЯ байта в конце. Функция возвращает запуск буфера.

Использование FreeRTOS

Когда мы думаем о современном компьютере, мы быстро понимаем, что у него есть какая-то операционная система. Общие примеры их - Microsoft Windows или Linux. Цель операционной системы состоит в том, чтобы обеспечить интерфейс между программными приложениями и основной инфраструктурой аппаратных средств. Если бы это не было для операционной системы, то каждое применение должно было бы, вероятно, выполнить свое собственное подобное внедрение таких функций, которые будут отходами. Почему бы не написать это однажды и обеспечить слой абстракции на который, более высокий уровень функционирует (такие как заявления), может быть построен. Возможности операционных систем на PC очень похожи. Они обращаются с управлением памятью, ввод/вывод аппаратных средств (читающий от клавишных инструментов и мышей и ведущий видеокарты), управление задачей (несколько программ, бегущих одновременно), диск и взаимодействия файловой системы и многое другое. Ранние операционные системы обеспечили основные функции, в то время как сегодняшние операционные системы стали более богатыми и более богатыми до такой степени, когда, их больше нельзя рассматривать как просто операционные системы. С тех пор, когда операционная система должна была обеспечить Freecell или Minesweeper?

Если мы перематываем часы и начинаем снова и обращаемся к основным аспектам операционной системы, мы приезжаем в сегодняшний FreeRTOS. FreeRTOS - общедоступная операционная система, которая предоставляет очень простые функции высокоуровневым заявлениям … снова … основное понятие цели операционной системы во-первых. Однако FreeRTOS разработан для встроенных систем, таких как ESP8266. Это - порядки величины, более простые, чем другие операционные системы, такие как Linux, но это дизайном.

FreeRTOS был перенесен к большому разнообразию платформ аппаратных средств включая центральные процессоры Xtensa, используемые в ESP8266. Когда собрано, это приводит к библиотеке, которая находится под 5K Байтами в размере.

Основные функции, которые это обеспечивает:

- управление памятью

- управление задачей
    - Синхронизация API Видит также:

- Свободная домашняя страница RTOS
- Исследование операционной системы: FreeRTOS
- GitHub:espressif/ESP32_RTOS_SDK

Архитектура задачи в FreeRTOS

Давайте начнем с понятия задачи. Задача - обрабатываемая деталь, которую мы хотим выполнить. При необходимости Вы можете думать об этом как о функции языка C. Например:

```
интервал
  добавляет
  (интервал
  a, интервал
  b)
  {возвращают
  ся + b;
}
```

мог считаться задачей …, хотя это будет смехотворно просто. В общем думайте о задаче как о выполнении части кодекса C, который Вы создали. Мы обычно думаем о кодексе, бегущем с его начала полностью до его конца … однако, это - не обязательно самый эффективный способ продолжиться. Рассмотрите идею применения, которое хочет послать некоторые данные по сети. Это может хотеть послать мегабайт данных … однако, это может также найти, что может только послать 100K за один раз, прежде чем это должно будет ждать переданных данных, которые будут поставлены. В той истории это послало бы 100K и ждало бы, чтобы передача закончила, послала следующий 100K, и ждите, чтобы та передача закончила и так далее. Но что из тех промежутков времени, где кодекс ждет, чтобы предыдущая передача закончила? Что центральный процессор делает в те времена?

Возможности состоят в том, что это делает только контроль для флага, который указывает, что передача закончила. Это - отходы. В теории центральный процессор мог выполнять другую работу (предполагающий, что есть на самом деле другая работа, которая могла быть выполнена). Если есть действительно другая доступная работа, мы могли «контекст переключаться» между этими пунктами работы, таким образом, что, когда каждый блокирует ожидание, что что-то произойдет, контроль мог быть передан другому, чтобы сделать что-то

полезное.

Если мы называем каждую обрабатываемую деталь «задачей», которая является ценностью задачи в FreeRTOS. Задача представляет обрабатываемую деталь, которая будет выполняться, но вместо того, чтобы предположить, что работа быстро пойдет от начала до конца, мы объявляем, что могут быть времена в рамках работы, где это может оставить контроль другой работе (задачи).

Принимая это во внимание, мы должны думать о том, как создана задача. Есть API, предоставленный FreeRTOS, названным «`xTaskCreate()`», который создает случай задачи.

Страница 298

Здесь важно понять, что задача - логическая абстракция. Нет ничего определенного, обеспеченного в центральном процессоре, который знает, какова задача. Вместо этого это - операционная система (FreeRTOS в нашем случае), который предоставляет модель задачи для нас.

Если мы думаем глубоко о задаче, мы можем забеременеть задачи, имеющей государство. В любой момент времени или задача бежит, или она не бежит. Задачей, которая бежит, является та, которая активно использует центральный процессор (т.е. не ждет, чтобы что-либо еще произошло). Задачей, которая не бежит, является та, у которой нет центрального процессора. Например, если бы мы создали две задачи, то один из них бежал бы и другой не управление. Если тот, который бежит, достигнет точки, где он больше не может выполнять значащую работу, он оставит контроль за центральным процессором и станет не управлением. У другой задачи тогда есть возможность стать управлением.

Идя еще глубже, когда задача не бежит, она может не «бежать» по конкретной причине …, такой как:

- Заблокированное ожидание чего-то, чтобы закончить
- Приостановленный пользователем
- Готовый бежать таким образом, что, когда задача, которая, больше бежит не бежит, эта задача имеет право стать управлением

В FreeRTOS мы определяем задачу как функцию C, которая берет

`пустоту *` параметр. Например,

```
пустота myTask (пустота *myParameters)
```

могла бы быть подпись для функции задачи.

Функция задачи, как ожидают, будет бежать навсегда. Если это должно закончиться, это должно вымыться прежде, чем возвратиться, призвав `vTaskDelete ()`.

Когда задача оставляет контроль назад OS, у OS тогда может быть выбор между несколькими задачами, относительно которых должен стать управлением. Этот процесс выбора называют, «намечая». FreeRTOS использует понятие «приоритета» определить который задача бежать затем. Каждую задачу, которая готова бежать, считают потенциальным кандидатом и тем, у которого есть самый высокий приоритет, станет тем, который бежит.

Кодируя непосредственно к FreeRTOS в non-ESP8266 окружающей среде, нужно было бы обычно звонить `vTaskStartScheduler ()`, чтобы гарантировать, что планировщик задачи готов к эксплуатации. Это не должно быть предпринято в окружающей среде ESP8266, поскольку внутренности окружающей среды ESP8266 уже зарегистрировали другие задачи и уже запустили планировщик.

Есть много связанных с таймером функций в FreeRTOS, которые работают над понятием «клещей», где тиканье - единица времени. В ESP8266 FreeRTOS интервал тиканья 1/100-й из секунды (10 национальной безопасности).

Блокирование и синхронизация в RTOS

С понятием параллельных задач обработки в RTOS у нас должен быть механизм, чтобы синхронизировать действия между задачами. Например, вообразите задачу, которая производит данные и вторую задачу, которая потребляет данные, произведенные первым. У задачи производства должен быть механизм, который описывает те данные, был произведен, и у задачи потребления должен быть механизм, чтобы заблокировать ожидание данных, которые будут произведены.

Один способ достигнуть этого через понятие «группы событий». Думайте о группе событий как о ряде флагов, у которых может быть стоимость «0» или «1». Задача может установить значение флага, и вторая задача может быть настроена, чтобы ждать (блокируют) до флага переходы от «0» до «1». То, что это означает, - то, что есть асинхронная и свободно

двойная коммуникация с помощью этих флагов. С точки зрения внедрения RTOS обеспечивает тип данных, названный «ручкой группы событий», которая осуществлена непрозрачным типом данных под названием «`EventGroupHandle_t`». Случай этого создан посредством требования к `xEventGroupCreate ()`. Мы должны предположить, что это, ручка группы событий может содержать максимум 8 отличных флагов, которые идентифицированы как 0 до 7. Мы можем установить флаги в группе событий, использующей `xEventGroupSetBits ()` и ясные флаги, используя `xEventGroupClearBits ()`. Если мы должны получить ценности группы событий, мы можем назвать `xEventGroupGetBits ()`. Просто переключение битов не настолько полезно, но мы входим в ядро истории с `xEventGroupWaitBits ()` вызов функции. Когда призвано, это заставляет посетителя быть заблокированным, пока названный бит или биты не становятся установленными.

Списки в RTOS

FreeRTOS предоставляет список, обрабатывающий функции.

ESP8266 – Строительство приложений для RTOS

Espressif распределяют SDK для строительства приложений RTOS для ESP8266. Этот SDK доступен от GitHub. Мой личный выбор для восстановления SDK состоит в том, чтобы использовать последнюю версию Затмения и использовать его встроенные поисковые инструменты Мерзавца.

Версия FreeRTOS, поставляемая Espressif, кажется, v7.5.2. Последнее доступное от FreeRTOS самостоятельно, кажется, v8.2.3.

Затмение также обеспечивает окружающую среду для компиляции программы C. Предложенные флаги компиляции C→Object:

Параметр/флаг	Значение
`-g`	

- Wpointer-арифметика
- Wundef
- Werror
- Wl, - El
- fno-inline-functions
- nostdlib
- mlongcalls
- mtext-section-literals
- ffunction-разделы
- fdatasections

Для соединения мы используем следующие флаги компоновщика:

- L$ (SDK_PATH) / lib
- WI, - разделы GC
- nostdlib
- T$ (LD_FILE)
- WI, - «никакие клетчатые разделы»
- u call_user_start
- WI, - статичный
- WI, - группа начала
- lminic
- lgcc
- lhal
- lphy
- lpp
- lnet80211
- lwpa
- lcrypto
- lmain
- lfreertos
- llwip

Настраивая CDT в Затмении, следующие параметры настройки предложены:

• C/C ++ Строят → Окружающую среду – ESP8266_SDK_ROOT = <Путь к RTOS SDK>

• C/C ++ Общие → Пути и Символы – GNU C – {ESP8266 _SDK_ROOT}/include/espressif
• C/C ++ Общие → Пути и Символы – GNU C – {ESP8266 _SDK_ROOT}/include/lwip
• C/C ++ Общие → Пути и Символы – GNU C – {ESP8266 _SDK_ROOT}/include/lwup/ipv4

Собрав исходный код в файлы объекта и затем связанный файлы объекта в ЭЛЬФА отформатировали исполняемый файл, что остается, должен разделить этот исполняемый файл на разделы, которые будут загружены во флэш-память ESP8266 в различных местоположениях. Это приведет к двум файлам для погрузки во вспышку. Один файл будет данными, которые будут в конечном счете загружены в RAM во времени выполнения, в то время как другой будет доступно как адресуемая флэш-память.

Мы можем использовать `esptool_ck` для этой задачи:

```
esptool_ck-eo bin.elf - филиал app_0x00000.bin - бакалавр наук .text -
бакалавр наук .data - бакалавр наук .rodata - бакалавр наук
.iram0.text - до н.э - EC
esptool_ck-eo $bin.elf-es .irom0.text ap_0x10000.bin - EC
```

После высвечивания к ESP8266 мы можем найти, что следующие сообщения ботинка созданы:

```
pp_task_hdl: 3ffef4e0, prio:13, stack:512
pm_task_hdl: 3ffefdb0, prio:1, stack:176
tcpip_task_hdl: 3ffef260, prio:10,
stack:512 idle_task_hdl: 3ffef300,
prio:0, stack:176 tim_task_hdl: 3fff1428,
prio:2, stack:256 xPortStartScheduler
frc2_timer_task_hdl:3fff1938, prio:12,
stack:512

OS SDK ver: 1.3.0 (68c9e7b) собранный 2 ноября
2015 18:53:21 phy ver: 484, стр ver: 9.9
SDK version:1.3.0 (68c9e7b)
```

См. также:

- GitHub:[espressif/ESP8266_RTOS_SDK]
-

Пульты с RTOS

Поток отладки по умолчанию написан UART0 при скорости передачи в бодах 74 880.

Отладка подсказок

Если вещи становятся странными, стирают всю вспышку Вашего устройства и начала снова.

В FreeRTOS, чтобы заставить отладку, которая будет написана UART1, может использоваться следующее:

```
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);
```

Развитие решений на Linux

Работая в Linux окружающая среда, есть определенные подсказки и методы, которые могли бы быть полезными/ценными.

• Соединяясь с правлением SP8266, использующим соединитель USB→UART, устройство может обнаружиться под`/dev` как `ttyUSB0`. Если мы исследуем разрешения на этот файл, мы можем найти, что он настроен как:

```
crw-rw----внедряют dialout
```

Это означает, что это доступно корнем и пользователями в `dialout` группе. Если Вы хотите высветить ESP8266 через это устройство, Ваш userid должен таким образом быть членом этой группы. Чтобы добавить Вашего пользователя к группе, следующая команда Linux может использоваться:

```
sudo usermod-a-G dialout <yourUserid>
```

после внесения изменения Вы должны выйти из системы, и регистрация въезжают задним ходом снова.

• Полезный неизлечимо больной клиент - `GtkTerm`. Этот инструмент предоставляет неизлечимо больному зрителю, который может использоваться, чтобы контролировать соединитель USB→UART, чтобы рассмотреть сообщения регистрации и отладки. Это создает конфигурационный файл в `$HOME/.gtktermrc`, который может быть отредактирован, чтобы изменить последовательный порт по умолчанию (например,`/dev/ttyUSB0`), а также изменение скорости передачи в бодах к Вашему требуемому значению.

• Другой хороший неизлечимо больной клиент - `экран`. Экран - полноэкранный эмулятор терминала.

• Еще один неизлечимо больной клиент - классическая команда `cu`. Снова, очень простой в использовании. Пример использования был бы:

```
$ cu - линия/dev/ttyUSB0 - скорость 115200
```

Чтобы оставить cu сессию, войдите в «~»..

Строительство окружающей среды Linux

Если Вы не управляете Linux с рождения, Вы можете хотеть полагать, что бегущая Oracle VirtualBox принимает окружающую среду Linux на Вашем Windows или машине Mac. Oracle VirtualBox - Общедоступное внедрение продукта виртуализации операционной системы. Можно загрузить VirtualBox отсюда:

https://www.virtualbox.org/

В моих тестах я управлял Ubuntu 15.10.

Я определяю дисковый размер по крайней мере 20 Гбайт и 2 Гбайт RAM. Если у Вас есть несколько ядер, Вы можете хотеть определить тех, которые как являются доступным.

После строительства изображения удостоверьтесь, что Вы позволяете способности скопировать и приклеить между хозяином OS и гостем OS.

Также удостоверьтесь, что инструменты гостя VirtualBox установлены.

Есть некоторые пакеты, включая которые Вы действительно не можете обойтись без:

- мерзавец

Вы можете установить новые пакеты с:

```
склонные sudo - добираются, устанавливают <пакет>
```

Как только у Вас есть установленный OS Linux, затем мы хотим построить окружающую среду компиляции. Популярный `pfalcon/esp-open-sdk` - то, что мы проиллюстрируем.

Прежде, чем начать остаток от нашего строить, мы должны гарантировать, что много дополнительных компонентов к Linux установлены, поскольку они обязательны для строительства toolchain. Следующей командой можно управлять, чтобы установить набор компонентов, в которых мы нуждаемся:

```
склонные sudo - добираются, устанавливают, делают unrar autoconf,
    автоделают libtool-мусорное-ведро gcc g ++ gperf \, сгибают
    бизона texinfo простофиля ncurses-dev libexpat-dev питон
    последовательный питоном sed \, мерзавец расстегивает молнию на
    help2man wget bzip2 удара
```

Сначала мы должны выполнить команду, чтобы загрузить основанный на GitHub проект:

```
клон мерзавца - рекурсивный https://github.com/pfalcon/esp-open-sdk.git
```

После того, как загруженный, мы можем построить решение с:

```
сделайте STANDALONE=y
```

Обратите внимание, что построить доступ сети потребностей и получит доступ к внешним веб-сайтам включая:

- www.mpfr.org

Строение/компиляция займет приблизительно час, чтобы закончить.

В заключении может быть найден ряд новых справочников. Среди них:

- `xtensa-lx106-elf/bin` – собранные инструменты включая gcc, objcopy и gdb.
- `sdk` – символическая связь с последним Espressif SDK

Мы хотим добавить некоторые переменные окружения в профиль нашего пользователя:

- Добавьте `xtensa-lx106-elf/bin` к ПУТИ

- Экспортируйте `ESP8266_SDK_ROOT` в корень SDK

Вы также захотите установить `esptool-ck` в свою местную папку мусорного ведра. Вы также захотите добавить свой userid к группе, названной dialout. Вот типовой Makefile для Linux:

```
PROJ_NAME=test1
СООТВЕТСТВУЮТ
=/dev/ttyUSB0
OBJS=user_main.o
uart.o
#
CC=xtensa-lx106-
elf-gcc
OBJS=user_main.o
uart.o APP=a.out
ESPTOOL_CK=espto
ol
CCFLAGS =-Wimplicit-function-declaration-fno-inline-functions-mlongcalls-mtext-section-literals \-
mno-serialize-volatile-I$ (ESP8266_SDK_ROOT) / включают-I.-D __ ETS __-DICACHE_FLASH-DXTENSA-
DUSE_US_TIMER

LDFLAGS =-nostdlib \
- L$ (ESP8266_SDK_ROOT)/-L$ lib (ESP8266_SDK_ROOT)/ld-T$ (ESP8266
_SDK_ROOT)/ld/eagle.app.v6.ld \-Wl, - «никакие клетчатые разделы»-u call_user_start-
Wl, - статический-Wl, - группа начала \
- lc-lgcc-lhal-lphy-lpp-lnet80211-llwip-lwpa-lmain-ljson-lupgrade-lssl \-
lpwm-lsmartconfig-Wl, - группа конца

все: $ (PROJ_NAME) _0x00000.bin $ (PROJ_NAME) _0x40000.bin

a.out: $ (OBJS)
        $ (CC)-o a.out $ (LDFLAGS) $ (OBJS)

$ (PROJ_NAME) _0x00000.bin: a.out
        $ (ESPTOOL_CK)-eo $ <-$ филиала - бакалавр наук .text - бакалавр наук .data - бакалавр
        наук .rodata - бакалавр наук .iram0.text - до н.э - EC || верный

$ (PROJ_NAME) _0x40000.bin: a.out
        $ (ESPTOOL_CK)-eo $ <$-es .irom0.text - EC || верный

.c.o:
        $ (CC) $ (CCFLAGS)-c $ <

чисто:
        комната-f a.out *.o *.bin


вспышка: все
        $ (ESPTOOL_CK) - $ CP (СООТВЕТСТВУЮТ) - CD nodemcu-cb 115200 - приблизительно $0x00000-
        cf (PROJ_NAME) _0x00000.bin $ (ESPTOOL_CK) - $ CP (СООТВЕТСТВУЮТ) - CD nodemcu-cb
        115200 - приблизительно $0x40000-cf (PROJ_NAME) _0x40000.bin
```

И также простое «`привет мировое`» применение:

Страница 306

```
#include
«osapi.h»
#include
«user_interface.h
```

```
» #include
«uart.h»
#include «espmissingincludes.h»

пустота uart_init_2 (UartBautRate uart0_br, UartBautRate uart1_br);

пустота systemInitDoneCB
       () {os_printf
       («Привет World
       \n»);
}
пустота user_init ()
       {uart_init_2
       (115200, 115200);
       system_init_done_cb (systemInitDoneCB);
}
```

Затем мы устанавливаем Яву 8. Загрузите с Oracle и извлечения в/usr/java. Например, смола zxvf файл. Обновите JAVA_HOME и ПУТЬ. Теперь, когда у нас есть окружающая среда компиляции, возможности высоки, мы захотим IDE …. мой фаворит - Затмение.

https://eclipse.org/do

wnloads/Пробег

инсталлятор

Затмения


Oomph License Confirmation

Начав Затмение, мы хотим установить обновления. Однако для Марса, не будет никого, поскольку это настолько новое.
Установите GtkTerm

Начните его однажды, чтобы создать `~/.gtktermrc` файл.
Отредактируйте тот файл и изменение:

- `порт =/dev/ttyUSB0`
- `скорость = 115200`

Добавьте нынешнего пользователя к группе dialout

```
sudo usermod-a-G dialout <yourUserid>
```

Вы будете нуждаться к выходу из системы и входить в систему снова, чтобы это вступило в силу. Вы можете утвердить тот свой userid, имеет правильную группу, управляя `идентификационной` командой от раковины:

```
Id $
uid=1000 (kolban) gid=1000 (kolban) groups=1000 (kolban), 4 (adm), 20
(dialout), 24 (CD-ROM), 27 (sudo), 30 (падение), 46 (plugdev), 108 (lpadmin),
124 (sambashare)
```

Загрузитеigrr/esptool-ck как `esptool` от выпусков GitHub. Гарантируйте, что это находится в Вашей папке `мусорного ведра` и что папка `мусорного ведра` находится на Вашем пути.

Теперь мы наконец готовы построить приложение. Мы можем загрузить образец для этой истории от GitHub в следующем URL:

https://github.com/nkolban/Sample-ESP8266-App.git

Этот образец предполагает, что xtensa инструменты находятся на Вашем `ПУТИ` и что переменная окружения `ESP8266_SDK_ROOT` правильно определена.

Я также установил бы:

- Хром

Страница 312

См. также:

- Oracle Virtual Box
- Загрузки Ubuntu

Ссылка API

Теперь у нас есть мини-ссылка на синтаксис многих выставленных API ESP8266. Не используйте эту ссылку исключительно. Пожалуйста, также обратитесь к изданному Руководству по программированию Espressif SDK.

Некоторые акронимы и другие имена используются в обозначении API и, возможно, нуждаются в некотором объяснении, чтобы осознать их:

- dhcpc – клиент DHCP
- dhcps – сервер DHCP
- softap – Точка доступа осуществлена в программном обеспечении
- wps – WiFi Защищенная Установка
- sntp – Простой Сетевой Протокол Времени
- mdns – Система доменных имен Передачи
- uart – Универсальный асинхронный приемник/передатчик
- pwm – модуляция Ширины импульса

Ссылка FreeRTOS API

См. также:

- Использование FreeRTOS

eTaskGetState

Восстановите состояние задачи.

```
eTaskState eTaskGetState (TaskHandle_t xTask)
```

pcTaskGetName

Получите название задачи.

```
случайная работа *pcTaskGetTaskName (TaskHandle_t xTaskToQuery)
```


xEventGroupClear

Ясный один или несколько битов в группе событий.

```
EventBits_t xEventGroupClearBits (EventGroupHandle_t
   eventGroup, константа EventBits_t bitsToClear)
```

- `eventGroup` – группа событий, которая содержит биты, которые будут очищены.
- `bitsToClear` – набор битов, которые будут очищены в группе событий.

Включает:

- <freertos/event_groups.h>

xEventGroupCreate

Создайте новую группу FreeRTOS событий.

```
EventGroupHandle_t xEventGroupCreate ()
```

Включает:

- <freertos/event_groups.h>

xEventGroupSetBits

Установите один или несколько битов в группе событий.

```
EventBits_t xEventGroupSetBits (EventGroupHandle_t
eventGroup, константа EventBits_t bitsToSet)
```

- `eventGroup` – группа событий, которая содержит биты, которые будут установлены.
- `bitsToSet` – набор битов, которые будут установлены в группе событий.

Включает:

- <freertos/event_groups.h>

xEventGroupWaitBits

Блок, ждущий, чтобы один или несколько «битов» стали установленными.

```
EventBits_t xEventGroupWaitBits (
   константа EventGroupHandle_t
   eventGroup, константа
   EventBits_t bitsToWaitFor,
   константа BaseType_t
   clearOnExit,
   константа BaseType_t
   waitForAllBits,
   TickType_t ticksToWait)
```

Вызывание этой функции может заставить посетителя блокировать, пока биты не установлены, или перерыв встречен. Биты могут быть установлены через `xEventGroupSetBits ()`.

- `eventGroup` – группа событий, которая ссылается на биты, которые будут наблюдаться.
- `bitsToWaitFor` – набор битов в группе событий, на которую мы ждем.
- `clearOnExit` – Должен биты, что мы ждем на быть очищенными автоматически
  когда установлено и этот метод прибыль?
- `waitForAllBits` – Должен мы открывать на первом бите, который установлен, что мы
  наблюдая за или альтернативно мы должны ждать на всех устанавливаемых битах?
- `ticksToWait` – Сколько клещей мы должны ждать перед

возвращением? Это
обеспечивает перерыв (в случае необходимости). Переменная RTOS «portMAX_DELAY» может использоваться, чтобы определить, что мы хотим ждать неопределенно.

Включает:

- <freertos/event_groups.h>

xTaskCreate

Создайте новый случай задачи.

```
BaseType_t xTaskCreate
   (pdTASK_CODE
   pvTaskCode,
   константа подписала
   portCHAR *pcName,
   неподписанный
   portSHORT
   usStackDepth, пустота
   *pvParameters,
   неподписанный
   portBASE_TYPE
   uxPriority, xTaskHandle
   *pxCreatedTask)
```

- `pvTaskCode` – Указатель на функцию задачи. В программировании C мы можем просто
  поставляйте название функции или, как был замечен в некоторых образцах, адресе названия функции. По-видимому, они равняются пунктам, которые достаточно близки, чтобы использоваться interchangablly.
- `pcName` – Отладка названия задачи.
- `usStackDepth` – Размер стека для задачи.
- `pvParameters` – Параметры для случая задачи. Это может быть ПУСТЫМ.
- `uxPriority` – Приоритет случая задачи.
- `xTaskHandle` – Ссылка на недавно созданный случай задачи. Это может быть
  переданный в как ПУСТОЙ УКАЗАТЕЛЬ, если никакая ручка задачи не требуется, чтобы быть возвращенной.

Когда созданная задача призвана и затем решает закончиться через возвращение, важно, что требование задачи `vTaskDelete (ПУСТОЙ УКАЗАТЕЛЬ)`, прежде чем это закончит. Запрос этого является признаком

к

FreeRTOS, которым задача закончена и потребность больше не быть рассмотренной для переключения контекста.

См. также:

- vTaskDelete
- xTaskCreate

vTaskDelay

Задержите задачу для конкретного количества клещей.

```
пустота vTaskDelay (константа TickType_t xTicksToDelay)
```

Постоянный названный `portTICK_PERIOD_MS` обеспечивает число клещей в миллисекунде. Если бы мы хотели задержаться в течение 1 секунды, то мы могли бы тогда поставлять 1000 / `portTICK_PERIOD_MS`.

См. также:

- vTaskDelay

vTaskDelayUntil

Задержите задачу до указанного абсолютного времени.

```
пустота vTaskDelayUntil (константа TickType_t
*pxPreviousWakeTime, константа TickType_t xTimeIncrement)
```

Это функциональные блоки задача до некоторого абсолютного времени в будущем.

- `pxPreviousWakeTime` – норматив времени, от которого приращение будет относительно.
- `xTimeIncrement` – время у клещей, который, когда оно добавлено к `pxPreviousWakeTime`, будет временем, когда задача готова бежать снова.

См. также:

- vTaskDelayUntil

vTaskDelete

Удалите случай задачи.

```
пустота vTaskDelete (TaskHandle_t pxTask)
```

Эта функция удалит случай задачи. Если ручка `pxTask` будет `ПУСТОЙ` тогда, то текущая задача будет удалена.

См. также:

- xTaskCreate
- vTaskDelete

Страница 316

xTaskGetCurrentTaskHandle

Получите текущую ручку задачи.

xTaskGetTickCount

Получите текущий подсчет тиканья.

```
portTickType xTaskGetTickCount ()
```

Возвратите число клещей, которые произошли, так как планировщик задачи был запущен.

vEventGroupDelete

Удалите группу событий.

```
пустота vEventGroupDelete (EventGroupHandle_t eventGroup)
```

Включает:

- <freertos/event_groups.h>

vTaskList

```
пустота vTaskList (случайная работа *pcWriteBuffer)
```

НЕ ДОСТУПНЫЙ

vTaskPrioritySet

```
пустота vTaskPrioritySet (TaskHandle_t pxTask, UBaseType_t uxNewPriority)
```

См. также:

- vTaskPrioritySet

vTaskResume

```
пустота vTaskResume (TaskHandle_t pxTaskToResume)
```

xTaskResumeAll

См. также:

- vTaskResumeAll

vTaskResumeFromISR

```
пустота xTaskResumeFromISR (TaskHandle_t pxTaskToResume)
```


vTaskSuspend

```
пустота vTaskSuspend (TaskHandle_t pxTaskToSuspend)
```

vTaskSuspendAll

См. также:

- vTaskSuspendAll

xQueueCreate

Создайте очередь для удерживания пунктов.

```
xQueueHandle xQueueCreate (
  неподписанный portBASE_TYPE
  uxQueueLength, неподписанный
  portBASE_TYPE uxItemSize)
```

- `uxQueueLength` – максимальное количество пунктов, которые может содержать очередь.
- `uxItemSize` – размер в байтах зарезервирован для каждого элемента в очереди.

См. также:

- xQueueCreate

vQueueDelete

```
пустота vQueueDelete (xQueueHandle xQueue)
```

xQueuePeek

```
portBASE_TYPE
  xQueuePeek
  (xQueueHandle
```

```
  xQueue, пустота
  *pvBuffer,
  portTickType xTicksToWait)
```

xQueueReceive

```
portBASE_TYPE xQueueReceive
   (xQueueHandle xQueue,
   пустота *pvBuffer,
   portTickType
   xTicksToWait)
```

xQueueSend

```
portBASE_TYPE xQueueSend
   (xQueueHandle xQueue,
   пустота константы *
   pvItemToQueue,
   portTickType
   xTicksToWait)
```

Страница 318

xQueueSendToBack

```
portBASE_TYPE xQueueSendToBack (xQueueHandle xQueue,
   пустота константы * pvItemToQueue, portTickType
   xTicksToWait);
```

xQueueSendToFront

```
portBASE_TYPE xQueueSendToFront (xQueueHandle xQueue,
 пустота константы * pvItemToQueue, portTickType xTicksToWait)
```

vSemaphoreCreateBinary

xSemaphoreCreateCounting

vSemaphoreGive

xSemaphoreGiveFromISR

vSemaphoreTake

pvPortMalloc

pvPortFree

Обработка списка

vListInitialise

Инициализируйте список.

```
пустота vListInitialise (xList * константа pxList)
```

`pxList` - список, который должен быть инициализирован.

vListInitialiseItem

Инициализируйте пункт для вставки в список.


```
пустота vListInitialiseItem (xListItem * константа pxItem)
```

Инициализируйте пункт, который может быть добавлен к списку.

vListInsert

Вставьте пункт в список.

```
пустота vListInsert (xList * константа pxList, xListItem * константа
pxNewListItem)
```

vListInsertEnd

Вставьте пункт в конце списка

```
пустота vListInsertEnd (xList * константа pxList, xListItem * константа
pxNewListItem)
```

Ссылка lwip

Гнезда

- принять

- связать
- закрытие
- closesocket
- соединиться
- getsocketname
- getpeername
- setsockopt
- getsockopt
- послушать
- recv
- recvfrom
- послать
- sendto
- гнездо
- выбрать

Страница 320

- ioctlsocket
- читать
- написать
- закрыть
- fcntl

Функции таймера

Функции таймера позволяют нам регистрировать функции, которые будут выполнены за один раз в будущем или периодически после проходов времени. Мы также группируем функции, которые управляют или восстанавливают временные стоимости в этом наборе.

os_delay_us

Задержка в течение многих микросекунд.

```
пустота os_delay_us (uint16 нас)
```

Задержка максимального интервала

65 535 микросекунд. Включает:

- osapi.h Видят также:

- Таймеры и время
- system_set_os_print

os_timer_arm

Позвольте таймер степени детализации миллисекунды.

```
пустота os_timer_arm
        (os_timer_t
        *pTimer,
        uint32_t
        миллисекунды,
        bool
        повторение)
```

Вооружите таймер, таким образом, который пометка запусков

и стреляет, когда часы достигают ноля. `pTimer` параметр -

указанный структура контроля за таймером.

Параметр `миллисекунд` - продолжительность таймера, измеряемого в

миллисекундах. `Повторный` параметр - перезапустит ли таймер, как

только он достиг ноля. Включает:

- osapi.h Видят также:

- Таймеры и время

- os_timer_disarm
- os_timer_setfn

os_timer_disarm

Разоружитесь/Аннулируйте ранее вооруженный таймер.

```
пустота os_timer_disarm (os_timer_t *pTimer)
```

Остановите ранее запущенный таймер, который был запущен

требованием к `os_timer_arm ().pTimer` параметр - указатель на структуру

контроля за таймером.

Включает:

- osapi.h Видят также:

- Таймеры и время

- os_timer_arm
- os_timer_setfn

os_timer_setfn

Определите функцию, которую назовут, когда таймер будет стрелять

```
пустота os_timer_setfn (os_timer_t
        *pTimer,
        os_timer_func_t
        *pFunction, пустота *pArg)
```

Определите функцию обратного вызова, которую назовут, когда таймер достигнет ноля. `pTimer` параметры - указатель на структуру контроля за таймером.

`pFunction` параметры - указатель на функцию обратного вызова.

`pArg` параметр - стоимость, которая будет передана в вызванную заднюю функцию. У функции обратного вызова должна быть подпись:

```
пустота (*functionName) (пустота *pArg)
```

`pArg` параметр - стоимость, зарегистрированная в функции обратного вызова. Включает:

- osapi.h Видят также:

- Таймеры и время
- os_timer_arm
- os_timer_disarm

Страница 322

system_timer_reinit

Используемый, чтобы установить микро второй таймер

os_timer_arm_us

Позвольте микро второй таймер

hw_timer_init

Инициализируйте таймер аппаратных средств

hw_timer_arm

Установите более аккуратную задержку

hw_timer_set_func

Установите отзыв таймера

Системные функции

system_adc_read

Прочитайте стоимость конвертера A/D.

```
uint16 system_adc_read ()
```

Прочитайте ценность аналого-цифрового преобразователя. Степень детализации - 1 024 дискретных шага.См. также:

- Аналог к цифровому преобразованию

system_deep_sleep_set_option

Определите то, что сделает чип, когда он затем проснется.

```
bool system_deep_sleep_set_option (uint8 выбор)
```

system_get_boot_mode

Получите текущий способ ботинка

```
uint8 system_get_boot_mode ()
```


Возвращаемое значение указывает на текущий способ ботинка и будет одним из:

- `SYS_BOOT_ENHANCE_MODE` – 0
- `SYS_BOOT_NORMAL_MODE` – 1

На моих устройствах возвращаемая стоимость «0».

system_get_boot_version

Версия загрузчика операционной системы.

```
uint8 system_get_boot_version ()
```

Текущая стоимость, возвращенная посредством тестирования моих устройств, равняется «5».

system_get_chip_id

Получите id чипа

```
длинный system_get_chip_id ()
```

Например: 0xf94322

system_get_cpu_freq

Получите текущую частоту центрального процессора

```
интервал system_get_cpu_freq ()
```

Возвращает частоту центрального процессора в MHz. Стоимость или будет 80 или 160.

system_get_flash_size_map

Получите текущий размер вспышки и карту

```
enum flash_size_map system_get_flash_size_map ()
```

Стоимость возвратилась, enum, у которого есть следующие определения:

- 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 Видят также:

- Погрузка программы в ESP8266

system_get_rst_info

Информация о текущем запуске.

```
структура rst_info* system_get_rst_info ()
```

Восстановите информацию о текущем запуске

устройства. Включает:

- user_interface.h Видят также:

- Обработка исключений
- структура rst_info

system_get_userbin_addr

Получите адрес пользовательского мусорного ведра

```
uint32 system_get_userbin_addr ()
```

Текущая стоимость, возвращенная на моих устройствах, является `0x0`.

system_get_vdd33

Напряжение меры

Неизвестный …, но связанный с аналогом к цифровому

преобразованию.См. также:

- Аналог к цифровому преобразованию
- Ошибка: Справочный источник, не найденный

system_init_done_cb

Зарегистрируйте функцию, которую назовут, когда системная инициализация будет завершена

```
пустота system_init_done_cb (init_done_cb_t callbackFunction)
```

Эта функция разработана только быть названной в `user_init ()`. названный одним временем после того, как ESP8266 был инициализирован. функция:

Это зарегистрируется, функция, чтобы быть `init_done_cb_t` определяет а

```
пустота (*functionName) (пустота)
```

Страница 325

Включает:

- user_interface.h Видят также:

- Таможенные программы

system_os_post

Разместите сообщение на задачу.

```
bool system_os_post (uint8
       приоритет,
       os_signal_t сигнал,
       os_param_t параметр)
```

Разместите сообщение на задачу. Задача не будет бежать немедленно, но будет бежать, как только она может.

`Приоритетная` область - приоритет запроса задачи. Три ценности определены –
`USER_TASK_PRIO_0`, `USER_TASK_PRIO_1` и `USER_TASK_PRIO_2`.

Параметр `сигнала` используется укладчиком задачи, чтобы определить, кто должен обработать сигнал. Это - на самом деле `uint32_t`.

Параметр `параметра` используется, чтобы пройти в дополнительных данных укладчику. Возвращение `верное` на успехе и `ложное` на неудаче.

Включает:

- user_interface.h Видят также:

- Обработка Задачи ESP8266

system_os_task

Установка задача для выполнения в более позднее время.

```
bool system_os_task
       (os_task_t задача,
       uint8 приоритет,
       os_event_
       t
       *очередь,
       uint
       queueLeng
       th)
```

«`Os_task_t`» - указатель на функцию укладчика задачи, у которой есть подпись:

```
пустота (*functionName) (os_event_t *событие)
```

Эта функция определена, чтобы быть укладчиком задачи, который получит все различные почтовые уведомления о том же приоритетном уровне.

`os_event_t` - структура, которая содержит:

- `сигнал os_signal_t`

Страница 326

- `os_param_t param`

Оба из них - неподписанные 32-битные целые числа.

`Приоритетная` область - приоритет запроса задачи. Определены три ценности:

- USER_TASK_PRIO_0
- USER_TASK_PRIO_1
- USER_TASK_PRIO_2

Возвращение `верное` на успехе и `ложное` на неудаче.

Включает:

- user_interface.h Видят также:
- Обработка Задачи ESP8266

system_phys_set_rfoption

Позвольте РФ после пробуждения от сна (или не)

system_phys_set_max_tpw

Установите максимальную силу передачи

system_phys_set_tpw_via_vdd33

Установите силу передачи как функцию напряжения

system_print_meminfo

Информация о памяти печати

```
пустота system_print_meminfo ()
```

Информация о памяти для диагностики написана потоку продукции, который обычно является UART1. Формат данных смотрит следующим образом:

```
данные:  0x3ffe8000  ~  0x3ffe853c,
len:  1340  rodata:  0x3ffe8540  ~
0x3ffe8af0,    len:    1456    bss:
0x3ffe8af0   ~   0x3fff1c18,   len:
37 160     куч:     0x3fff1c18     ~
```

`0x3fffc000, len: 41960`

`.data` раздел - то, где глобальные и статические местные инициализированные переменные сохранены. `.rodata` раздел - то, где глобальные и статические данные только для чтения сохранены.

`.bss`, где не инициализировано, глобальные и местные статические данные сохранены.

`os_printf ()`.
`.heap` - то, где куча программы может быть найдена.

См. также:

- Википедия – .bss
- Википедия – сегмент Данных

system_restart_enhance

Перезапускает систему в расширенном способе ботинка

system_rtc_clock_cali_proc

Калибровка часов.

```
uint32 system_rtc_clock_cali_proc (пустота)
```

Восстановите оперативную калибровку часов. Это - стенная продолжительность часов такта, измеряемого в микро секунды. 16-битное число возвратилось, имеет биты представление 11-0 стоимости после десятичной запятой. Мы можем умножиться, стоимость возвратилась сюда против количества циклов начиная с предыдущего перезапуска, и определите истекшую стенную стоимость часов.

system_set_os_print

Включите или выключите регистрацию.

```
пустота system_set_os_print (unint8 релейный)
```

Ценность 0 выключает его, в то время как ценность 1 включает его. это управляло уровнем OS, регистрирующимся однако, это, кажется, управляет Включает:

Об этом первоначально думали это

вся регистрация через

- user_interface.h Видят также:

- os_printf
- os_install_putc1
- Регистрация к UART1

system_show_malloc

Отладьте потенциальные проблемы утечки памяти.

```
пустота system_show_malloc ()
```

Этот API должен также быть активирован, явно определив `MEMLEAK_DEBUG`.

Документация относительно этой функции в руководстве по программированию SDK обеспечивает много предупреждений и протестов, которые полностью еще не поняты так использование с осторожностью.

-

system_rtc_clock_cali_proc

Калибровка часов.

```
uint32 system_rtc_clock_cali_proc (пустота)
```

Восстановите оперативную калибровку часов. Это - стенная продолжительность часов такта, измеряемого в микро секунды. 16-битное число возвратилось, имеет биты представление 11-0 стоимости после десятичной запятой. Мы можем умножиться, стоимость возвратилась сюда против количества циклов начиная с предыдущего перезапуска, и определите истекшую стенную стоимость часов.

system_uart_swap

Обменяйте последовательный UARTs.

Когда ESP8266 запускает, он использует определенные булавки для контроля за UART0. А именно, этому нужны булавки для функций TX, RX, CTS и PTC. Вызывая эту функцию, физические булавки, используемые для UART0, передвинуты.

Функция	Дефолт	Обменянный

U0TXD	U0TXD	MTDO
U0RXD	U0RXD	MTCK
U0CTS	MTCK	U0RXD
U0RTS	MTDO	U0TXD

system_soft_wdt_feed

Накормите контрольную комиссию программного обеспечения.

```
пустота system_soft_wdt_feed ()
```

Накормите контрольную комиссию программного обеспечения. Функция только имеет ценность, когда сторожевому псу программного обеспечения позволяют. Если мы должны выполнить перекручивание в рамках нашего кодекса, мы должны периодически вызывать эту функцию так, чтобы мы не морили время выполнения WiFi голодом. Мотив здесь, голодают и еда и следовательно понятие охранительного таймера, который проверяет, что мы не тратим слишком долго далеко от WiFi …, таким образом, мы должны накормить собаку. Интересные метафоры.

Однако … эксперименты показывают, что это, кажется, ничего на самом деле не ДЕЛАЕТ. Тайна его цели продолжается.

См.:http://bbs.espressif.com/viewtopic.php?f=7&t=1055

system_soft_wdt_stop

Искалечьте сторожевой пес программного обеспечения.

```
пустота system_soft_wdt_stop ()
```

Остановите контрольную комиссию программного обеспечения. Рекомендуется не остановить этот таймер слишком долго (8 секунд или меньше) иначе, сторожевой пес аппаратных средств вызовет сброс.

См. также:

- Охранительный таймер

system_soft_wdt_restart

Перезапустите контрольную комиссию программного обеспечения.

```
пустота system_soft_wdt_restart ()
```

Перезапустите контрольную комиссию программного обеспечения после предыдущего требования остановить его.См. также:

- Охранительный таймер

system_uart_de_swap

Вернитесь к оригинальному UART.

system_update_cpu_freq

Установите частоту центрального процессора

```
пустота system_update_cpu_freq (международная частота)
```

Установите частоту центрального процессора. Или 80 или 160.

os_memset

Установите значения памяти

```
пустота os_memset (пустота *pBuffer, международная стоимость, size_t
размер)
```

Установите память, на которую указывает `pBuffer` к `стоимости` для байтов `размера`. Включает:

- osapi.h Видят также:

- Работа с памятью
- os_memcpy

os_memcmp

Сравните две области памяти.

```
интервал os_memcmp (uint8 *ptr1, uint8 *ptr2, международный размер)
```

Сравните две области памяти. Возвращение 0, если они равны.

Включает:

- osapi.h

os_memcpy

Скопируйте ценности памяти.

```
пустота os_memcpy (пустота *место назначения, пустота *источник, size_t
размер)
```

Копия память от буфера указала `с разбивкой по источникам` к буферу, на который указывает `место назначения` для числа байтов, определенных `размером`.

Включает:

- osapi.h Видят также:

- Работа с памятью
- os_memset

os_malloc

Ассигнуйте хранение от кучи.

```
пустота *malloc (size_t размер)
```

Ассигнуйте байты `размера` от кучи и возвратите указатель на ассигнованное хранение. Включает:

- mem.h Видят также:

- Работа с памятью
- os_zalloc
- os_free

os_calloc

Ассигнуйте хранение для ряда элементов.

```
пустота *calloc (size_t цифра, size_t размер)
```

Здесь мы ассигнуем `цифровые` случаи измеренных объектов `размера` в смежной памяти. Включает:

- mem.h

os_realloc

Перераспределите ранее полученную часть памяти с новым размером.

```
пустота *os_realloc (пустота *buf, size_t newSize)
```

Включает:

- mem.h

os_zalloc

Ассигнуйте хранение от кучи и ноля ее ценности.

```
пустота *os_zalloc (size_t размер)
```

Ассигнуйте байты `размера` от кучи и возвратите указатель на ассигнованное хранение. Перед возвращением обнулен склад.

Включает:

- mem.h Видят также:

- Работа с памятью
- os_malloc
- os_free

os_free

Выпуск ранее ассигновал хранение назад куче.

```
пустота os_free (пустота *pBuffer)
```

Выпустите хранение, ранее ассигнованное `os_malloc ()` или `os_zalloc ()` назад к куче.

Включает:

- mem.h Видят также:

- Работа с памятью
- os_malloc
- os_zalloc

os_bzero

Установите значения памяти нолю.

```
пустота os_bzero (пустота *pBuffer, size_t размер)
```

Устанавливает указатель данных на `pBuffer` к нолю для байтов `размера`. Включает:

- osapi.h Видят также:

- Работа с памятью

os_delay_us

Задержка в течение многих микросекунд.

```
пустота os_delay_us (uint16 нас)
```

Задержка максимального интервала 65 535 микросекунд. Включает:

- osapi.h Видят также:

- Таймеры и время
- system_set_os_print

os_printf

Напечатайте последовательность к UART.

```
пустота os_printf (случайная работа *формат, …)
```

Флаги формата, которые, как известно, работают, включают:

- `%d` – показывают целое число
- `%ld` – показывают длинное целое число
- `%lu` – показывают длинное неподписанное целое число
- `%x` – показывают как шестнадцатеричное число
- `%s` – показывают как последовательность
- «`\n`» – показывают newline (включает снабженный префиксом перевод каретки),

Обратите внимание, что нет **никакого** `%f`, чтобы напечатать плавание или дважды.

Синтезируемый текст посылают в функцию, зарегистрированную в `os_install_putc1 ()`. По умолчанию это - UART0, но может быть изменено на UART1, установив `uart1_write_char ()` функция.

Включает:

- osapi.h Видят также:

- Отладка
- os_install_putc1
- system_set_os_print

os_install_putc1

Зарегистрируйтесь функция печатают характер

```
пустота os_install_putc1 (пустота (*pFunc) (случайная работа c));
```


Зарегистрируйте функцию, которая будет вызвана функциями продукции, такими как `os_printf ()`, который зарегистрирует продукцию. Например, это может использоваться, чтобы написать последовательным портам. Когда звонок сделан поставляемому `uart_init ()` метод, функция письма собирается написать UART1.

Включает:

- osapi.h Видят также:

- os_printf
- system_set_os_print

os_random

```
неподписанный длинный os_random ()
```

Включает:

- osapi.h

os_get_random

```
интервал os_get_random (неподписанная случайная работа *buf, size_t len)
```

Включает:

- osapi.h

os_strlen

Получите длину последовательности.

```
интервал os_strlen (случайная работа *последовательность)
```

Возвратитесь длина пустого

указателя закончила

последовательность. Включает:

- osapi.h

os_strcat

Свяжите две последовательности вместе.

```
случайная работа *os_strcat (случайная работа *str1, случайная работа
*str2)
```

Свяжите законченное жало пустого указателя, на которое указывает `str1` с последовательностью, на которую указывает `str2`, и сохраните результат в `str1`.

Включает:

- osapi.h

os_strchr

Включает:

- osapi.h

os_strcmp

Сравните две последовательности.

```
интервал os_strcmp (случайная работа *str1, случайная работа *str2)
```

Сравните пустой указатель закончился, последовательность, на которую указывает `str1` с пустым указателем, закончила последовательность, на которую указывает `str2`. Если `str1` <`str2` тогда возвращение <0. Если `str1`> `str2` тогда возвращение> 0 иначе, они равны, и возвращение 0.

Включает:

- osapi.h

os_strcpy

Скопируйте одну последовательность другому.

```
случайная работа *os_strcpy (случайная работа *dest, случайная работа *src)
```

Скопируйте законченную последовательность пустого указателя,

на которую указывает `src` к памяти, расположенной в `dest`.

Включает:

- osapi.h

os_strncmp

Включает:

- osapi.h

os_strncpy

Скопируйте одну последовательность другому, но будьте чувствительны на сумму памяти, доступной в целевом буфере.

```
случайная работа *os_strncpy (случайная работа *dest, случайная работа
*источник, size_t sizeOfDest)
```

Поймите, что получающаяся последовательность в `dest`

может **не** быть пустая законченный. Включает:

- osapi.h

os_sprintf

```
sprintf (случайная работа * буфер, случайная работа *формат...)
```

Формат не так богат как нормальный `sprintf ()` в библиотеке C.

Например, никакое плавание или дважды не поддерживает.

Включает:

- osapi.h

os_strstr

Включает:

- osapi.h

Вспышка SPI

Пчела Вспышки SPI позволяет нам читать, писать и стирать сектора, содержавшие во флэш-памяти. Обратите внимание, что есть определенный документ от Espressif, который покрывает функции Вспышки SPI исключительно.

spi_flash_get_id

GE информация о ID вспышки SPI

```
uint32 spi_flash_get_id (пустота)
```

Это немного закодировано стоимость, которая представляет информацию о чипе вспышки, используемом вместе с ESP8266.

esptool также включает команду (flash_id), который может использоваться, чтобы восстановить и показать информацию о вспышке.

Включает:

- spi_flash.h Видят также:
- esptool.py

spi_flash_erase_sector

Сотрите сектор вспышки. Каждый сектор - 4k в размере.

```
SpiFlashOpResult spi_flasg_erase_sector (uint16 секунда)
```

Параметр секунды - число сектора (сектор составляет

4 096 байтов в размере). Включает:

-

spi_flash.

h Видят

также:

spi_flash_read

Прочитайте данные из вспышки

```
SpiFlashOpResult spi_flash_read (uint32 src_addr, uint32 des_addr, uint32
размер)
```

`src_addr` параметр - адрес во вспышке, которая будет прочитана. `des_addr` - адрес в памяти, которая будет написана. Параметр `размера` - размер данных, которые будут прочитаны.

Включает:

-

spi_flash.

h Видят

также:

spi_flash_set_read_func

```
пустота spi_flash_set_read_func (user_spi_flash_read прочитанный)
```

Включает:

-

spi_flash.

h Видят

также:

system_param_save_with_protect

Экономия памяти

```
bool system_param_save_with_protect (uint16 start_sec, пустота *param,
uint16 len)
```

Включает:

-

spi_flash.

h Видят

также:

spi_flash_write

Напишите данные, чтобы вспыхнуть

```
SpiFlashOpResult spi_flash_write (uint32 destAddr, uint32 *srcAddr, uint32
размер)
```

`destAddr` - адрес во вспышке, которая должна быть написана. адрес в памяти от того, где новые данные должны быть взяты. размер данных, которые будут написаны.

Включает:

`srcAddr` - источник, который параметр `размера`

-

spi_flash.h

Видят

также:

system_param_load

Прочитайте данные, сохраненные с защитой вспышки

```
bool system_param_load (uint16 start_sec, uint16 погашение, пустота *param,
uint16 len)
```

Включает:

-

spi_flash.h

Видят

также:

WiFi - ESP8266

wifi_fpm_close

wifi_fpm_do_sleep

wifi_fpm_do_wakeu

p

wifi_fpm_get_sleep_

type wifi_fpm_open

wifi_fpm_set_sleep_

type

wifi_fpm_set_wakeu

p_cb

wifi_get_channel

wifi_get_ip_info

Восстановите текущую IP информацию о станции.

```
bool wifi_get_ip_info
       (uint8 if_index,
       структура
       ip_info
       *информация)
```

`if_index` параметр определяет интерфейс, чтобы восстановить. Определены две ценности:

- `STATION_IF` – 0 – станционный интерфейс
- Интерфейс SOFTAP_IF – 1 – The Soft Access Point

`Информационный` параметр населен с деталями текущего IP-адреса, netmask и ворот.

Включает:

- user_interface.h Видят также:

- Текущий IP-адрес, netmask и ворота
- структура ip_info

wifi_get_macaddr

Получите MAC-адрес.

```
bool wifi_get_macaddr (uint8 if_index, uint8 *macaddr)
```

MAC-адрес составляет 6 байтов.

Включает:

- user_interface.h

wifi_get_opmode

Получите рабочий режим WiFi

```
uint8 wifi_get_opmode ()
```

Возвратите текущий рабочий режим устройства. Есть

четыре определенные ценности:

- `NULL_MODE` – Пустой способ. (0)
- `STATION_MODE` – Станционный способ. (1)
- `SOFTAP_MODE` – Мягкий способ Точки доступа (AP). (2)
- `STATIONAP_MODE` – Станция + Мягкий способ Точки доступа (AP). (3)

Включает:

- user_interface.h Видят также:

- Определение рабочего режима
- wifi_get_opmode_default

wifi_get_opmode_default

Получите рабочий режим по умолчанию

```
uint8 wifi_get_opmode_default ()
```

Возвратите рабочий режим по умолчанию устройства после запуска.

Есть три определенные ценности:

- `STATION_MODE` – Станционный способ
- `SOFTAP_MODE` – Мягкий способ Точки доступа (AP)
- `STATIONAP_MODE` – Станция + Мягкий способ Точки доступа (AP)

Включает:

- user_interface.h Видят также:

- Определение рабочего режима
- wifi_get_opmode
- wifi_set_opmode
- wifi_set_opmode_current

wifi_get_phy_mode

Получите физический уровень способ WiFi.

```
enum phy_mode wifi_get_phys_mode ();
```

Это используется, чтобы восстановить, сеть IEEE 802.11 печатают такой b/g/n. Включает:

- user_interface.h Видят также:

- enum phy_mode
-

wifi_get_sleep_type

Включает:

- user_interface.h

wifi_get_user_fixed_rate

интервал wifi_get_user_fixed_rate (uint8 *enable_mask, uint8 *уровень)

wifi_get_user_limit_rate_mask uint8

wifi_get_user_limit_rate_mask ()

wifi_set_broadcast_if

bool wifi_set_broadcast_if (uint8 интерфейс)

Включает:

- user_interface.h Видят также:

- Передача с UDP

wifi_get_broadcast_if

uint8

wifi_get_broadcast_if

()

Включает:

- user_interface.h Видят также:
- Передача с UDP

wifi_set_sleep_type

Включает:

- 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_rec

v_cb wifi_rfid_locp_recv_close

wifi_rfid_locp_recv_open

wifi_send_pkt_freedom

wifi_set_channel

wifi_set_event_handle_cb

Определите функцию обратного вызова, чтобы ощутить события WiFi.

```
пустота wifi_set_event_handler_cb (wifi_event_handler_cb_t
callbackFunction)
```

Регистрирует функцию, которую назовут, когда событие обнаружено подсистемой WiFi. Подпись зарегистрированной функции обратного вызова:

```
пустота (*functionName) (System_Event_t *событие)
```


Включает:

- user_interface.h Видят также:

- Обработка событий WiFi
- System_Event_t

wifi_set_ip_info

Установите интерфейсные данные для устройства.

```
bool wifi_set_ip_info (uint8 if_index, структура ip_info *информация)
```

`if_index` параметр определяет интерфейс, чтобы восстановить. Определены две ценности:

- `STATION_IF` – 0 – станционный интерфейс
- Интерфейс SOFTAP_IF – 1 – The Soft Access Point

`Информационный` параметр - указатель на `структуру ip_info`, который содержит значения, которые мы хотим установить.

Включает:

- user_interface.h Видят также:

- Текущий IP-адрес, netmask и ворота
- структура ip_info

wifi_set_macaddr

Установите MAC-адрес.

```
bool wifi_set_macaddr (uint8 if_index, uint8 *macaddr)
```

MAC-адрес составляет 6 байтов.

Включает:

- user_interface.h

wifi_set_opmode

Установите рабочий режим WiFi включая экономию вспыхивать.

```
bool wifi_set_opmode (uint8 opmode)
```

Есть три определенные ценности:

- `STATION_MODE` – Станционный способ
- `SOFTAP_MODE` – Мягкий способ Точки доступа (AP)

- `STATIONAP_MODE` – Станция + Мягкий способ Точки доступа (AP)

Включает:

- user_interface.h Видят также:

- Определение рабочего режима
- wifi_get_opmode
- wifi_get_opmode_default

wifi_set_opmode_current

Установите рабочий режим WiFi, но не экономьте, чтобы вспыхнуть.

```
bool wifi_set_opmode_current (uint8 opmode)
```

Есть три определенные ценности:

- `STATION_MODE` – Станционный способ
- `SOFTAP_MODE` – Мягкий способ Точки доступа (AP)
- `STATIONAP_MODE` – Станция + Мягкий способ Точки доступа (AP)

Включает:

- user_interface.h Видят также:

- Определение рабочего режима
- wifi_get_opmode
- wifi_get_opmode_default

wifi_set_phy_mode

Установите физический уровень способ WiFi.

```
bool wifi_set_phy_mode (enum phy_mode способ)
```

Это используется, чтобы установить тип сети IEEE 802.11 такой b/g/n.

Включает:

- user_interface.h Видят также:

- enum phy_mode

wifi_set_promiscuous_rx_cb

wifi_set_sleep_type

wifi_set_user_fixed_rate

интервал wifi_set_user_fixed_rate (uint8 enable_mask, uint8

уровень) позволить маска может быть одним из:

- FIXED_RATE_MASK_NONE
- FIXED_RATE_MASK_STA
- FIXED_RATE_MASK_AP
    - FIXED_RATE_MASK_ALL уровень может быть одним из:
- 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 (uint8 способ, uint8 ifidx, uint8 макс., uint8 минута)

wifi_set_user_sup_rate
интервал wifi_set_user_sup_rate (uint8 минута, uint8 макс.)

- RATE_11B5M
- RATE_11B11M
- RATE_11B1M
- RATE_11B2M
- RATE_11G6M

- RATE_11G12M
- RATE_11G24M
- RATE_11G48M

- RATE_11G54M
- RATE_11G9M
- RATE_11G18M
- RATE_11G36M

wifi_status_led_install

Свяжите булавку GPIO со светодиодом статуса WiFi.

```
пустота
        wifi_status_led_inst
        all (uint8 gpio_id,
        uint32
        mux_name
        , uint8
        gpio_fun
        c)
```

Когда трафик WiFi течет, мы можем хотеть, чтобы светодиод статуса мерцал или мигнул указанием на плавный трафик. Эта функция позволяет нам определять GPIO, который должен пульсироваться, чтобы указать на трафик WiFi.

`gpio_id` параметр - числовой ПИН-код.

`mux_name` - название мультиплексора логическое имя.

`gpio_func` - функция, которая будет позволена для

того мультиплексора. Включает:

- user_interface.h Видят также:
- wifi_status_led_uninstall

wifi_status_led_uninstall

Разъедините светодиод статуса с булавкой GPIO.

```
пустота wifi_status_led_uninstall ()
```

Разъединяет предыдущую установку ассоциации с требованием к

`wifi_status_led_install ()`. Включает:

- user_interface.h Видят также:
- wifi_status_led_install

`wifi_station_set_auto_connect ()`

wifi_unregister_rfid_locp_recv_cb

wifi_unregister_send_pkt_freedom_cb

wifi_unregister_user_ie_manufacturer_recv_cb

Станция WiFi

Следующие API касаются ESP* устройство, действующее как станция и соединяющееся с внешней точкой доступа.

wifi_station_ap_change

Измените связь с другой точкой доступа

```
bool wifi_station_ap_change (uint newApId)
```

Включает:

- user_interface.h

wifi_station_ap_number_set

Количество станций, которые припрячутся про запас

```
bool wifi_station_ap_number_set (uint8 ap_number)
```

Включает:

- user_interface.h

wifi_station_connect

Соедините станцию с точкой доступа.

```
bool wifi_station_connect ()
```

Если мы уже связаны с другой точкой доступа тогда, мы сначала должны разъединить от нее, используя `wifi_station_disconnect ()`. Есть также автомобиль, соединяют признак, который может использоваться, чтобы позволить устройству пытаться соединиться с последней замеченной точкой доступа
когда это приведено в действие на.

Это может быть установлено с

функцией.

Включает:

- user_interface.h Видят также:
- Соединение с точкой доступа
-

wifi_station_dhcpc_start

Начните клиента DHCP.

```
bool wifi_station_dhcpc_start ()
```

Если DHCP будет позволен, то IP, netmask и ворота будут восстановлены от сервера DHCP, в то время как, если отключено, мы будем использовать статические ценности.

Включает:

- user_interface.h Видят также:

- Текущий IP-адрес, netmask и ворота
- wifi_station_dhcpc_stop

wifi_station_dhcpc_status

Получите статус клиента DHCP

```
enum dhcp_status wifi_station_dhcpc_status ()
```

Один из:

- DHCP_STOPPED
- DHCP_STARTED включает:
- user_interface.h

wifi_station_dhcpc_stop

Остановите клиента DHCP

```
bool wifi_station_dhcpc_stop ()
```

Если DHCP будет позволен, то IP, netmask и ворота будут восстановлены от сервера DHCP, в то время как, если отключено, мы будем использовать статические ценности.

Включает:

- user_interface.h Видят также:

- Текущий IP-адрес, netmask и ворота

wifi_station_disconnect

Разъедините станцию от точки доступа.

```
bool wifi_station_disconnect ()
```

Мы должны предположить, что ранее соединились через `wifi_station_connect ()`. Мы можем определить наш текущий статус связи через `wifi_station_get_connect_status ()`.
Возвращение `верное` на успехе и ложное на `ошибке`. Включает:

- user_interface.h

wifi_station_get_ap_info

Получите информацию припрятавших про запас точек доступа

```
uint8 wifi_station_get_ap_info (структура station_config конфигурации [])
```

Включает:

- user_interface.h

wifi_station_get_auto_connect

Определите, будет ли ESP автомобиль соединяться с последней точкой доступа на ботинке.

```
uint8 wifi_station_get_auto_connect ()
```

Определите, попытается ли устройство автосоединиться с последней точкой доступа на перезапуске. Стоимость, если 0 средств это не будет, в то время как не 0 средств это будет.
Включает:

- user_interface.h Видят также:
- wifi_station_set_auto_connect

wifi_station_get_config

Получите текущую станционную конфигурацию

```
bool wifi_station_get_config (структура station_config *конфигурация)
```

Восстановите текущие станционные параметры настройки конфигурации.
Включает:

- user_interface.h Видят также:
- Станционная конфигурация

- station_config
- wifi_station_set_config

- wifi_station_set_config_current

wifi_station_get_config_default

Доберитесь станционная конфигурация по

умолчанию Включает:

- user_interface.h Видят также:
- Станционная конфигурация

wifi_station_get_connect_status

Получите статус связи станции.

```
uint8 wifi_station_get_connect_status ()
```

Результат - enum со следующими возможными ценностями:

Название Enum	Стоимость
STATION_IDLE	0
STATION_CONNECTING	1
STATION_WRONG_PASSWORD	2
STATION_NO_AP_FOUND	3
STATION_CONNECT_FAIL	4
STATION_GOT_IP	5
Не в станционном способе	255

Включает:

- user_interface.h Видят также:
- WiFi.printDiag

wifi_station_get_current_ap_id

Получите текущий id точки доступа

```
uint8 wifi_station_get_current_ap_id ()
```

Включает:

- user_interface.h

wifi_station_get_hostname

Получите имя хоста DHCP устройства WiFi.

```
случайная работа* wifi_station_get_hostname ()
```

Включает:

- user_interface.h

wifi_station_get_reconnect_policy

wifi_station_get_rssi

Получите полученный признак силы сигнала (rssi).

```
sint8 wifi_station_get_rssi ()
```

Получите полученный признак силы

сигнала (rssi). Включает:

- user_interface.h

wifi_station_scan

Просмотр для доступных точек доступа

```
bool wifi_station_scan (
       структура scan_config
       *конфигурация,
       scan_done_cb_t
       callbackFunction)
```

Мы можем просмотреть частоты WiFi, ища точки доступа. Мы должны быть в станционном способе, чтобы выполнить команду. Когда функция выполнена, мы обеспечиваем функцию обратного вызова, которая будет асинхронно призвана в некоторое время в будущем с результатами.

`scan_config` структура содержит:

- uint8 *ssid
- uint8 *bssid
- канал uint8

- uint8 show_hidden

Если мы поставляем эту структуру, то только точки доступа, что матч возвращен.

`scan_config` параметр может быть `ПУСТЫМ`, в этом случае, никакая фильтрация не будет выполняться, и все точки доступа будут возвращены.

Страница 350

`scan_done_cb_t` - функция со следующей структурой:

```
пустота (*functionName) (пустота *аргумент, статус СТАТУСА)
```

Параметр `аргумента` - указатель на `структуру bss_info`.

Важно отметить, что **через первый** вход в цепи нужно перескочить, поскольку это - заголовок списка.

Чтобы получить следующий вход, мы можем использовать `STAILQ_NEXT (pBssInfoVar, затем)`.

AUTH_MODE - enum

Название Enum	Стоимость
AUTH_OPEN	0
AUTH_WEP	1
AUTH_WPA_PSK	2
AUTH_WPA2_PSK	3
AUTH_WPA_WPA2_PSK	4

`СТАТУС` - enum, содержащий:

Название Enum	Стоимость
Хорошо	0
ТЕРПЯТ НЕУДАЧУ	1
ОЖИДАНИЕ	2
ЗАНЯТЫЙ	3
ОТМЕНИТЬ	4

На успехе функция возвращается верный и ложный на неудаче.

Название этой функции странное. Учитывая, что это, кажется, определяет местонахождение точек доступа и не станций, я полагаю,

что более соответствующее имя было бы
`wifi_access_point`

`_scan ().`

Включает:

- user_interface.h Видят также:

- Просмотр для точек доступа
- структура bss_info
- СТАТУС

wifi_station_set_auto_connect

Набор, будет ли ESP автомобиль соединяться с последней точкой доступа на ботинке.


```
bool wifi_station_set_auto_connect (uint8 setValue)
```

Набор, попытается ли устройство к автомобилю - соединяется с последней точкой доступа на перезапуске. Ценность 0 средств, это не будет, в то время как не 0 стоимостей означают это, будет. Если названо в `user_init ()`, урегулирование немедленно будет эффективным. Если названо в другом месте, урегулирование вступит в силу на следующем перезапуске.

Включает:

- user_interface.h Видят также:

- wifi_station_get_auto_connect

wifi_station_set_cert_key

Свидетельство набора и частный ключ для соединения с точкой доступа WPA2-предприятия.

```
bool wifi_station_set_cert_key (
  uint8 *client_cert, интервал
  client_cert_len, uint8
  *private_key, интервал
  private_key_len,
  uint8 *private_key_passwd, интервал private_key_passwd_len)
```

wifi_station_clear_cert_key

Высвободите средства и ясный статус после соединения с точкой доступа WPA2-предприятия.

```
пустота wifi_station_clear_cert_key (пустота)
```

wifi_station_set_config

Установите конфигурацию станции.

```
bool wifi_station_set_config (структура station_config *конфигурация)
```

Эта функция может только быть вызвана, когда способ устройства включает Станционную поддержку. А именно, детали которого точка доступа взаимодействовать с поставляется здесь. Детали сохранены через перезапуск устройства.

Возвращаемое значение истинных указывает на успех, и

ценность ложных указывает на неудачу. Включает:

- user_interface.h Видят также:

- Станционная конфигурация
- station_config

- wifi_station_get_config
- wifi_station_get_config_default

wifi_station_set_config_current

Установите конфигурацию станции, но не экономьте, чтобы вспыхнуть.

```
bool wifi_station_set_config_current (структура station_config
*конфигурация)
```

Эта функция может только быть вызвана, когда способ устройства включает Станционную поддержку. А именно, детали которого точка доступа взаимодействовать с поставляется здесь. Детали не сохранены через перезапуск устройства.

Возвращаемое значение истинных указывает на успех, и

ценность ложных указывает на неудачу. Включает:

- user_interface.h Видят также:

- Станционная конфигурация
- station_config
- wifi_station_get_config
- wifi_station_get_config_default

wifi_station_set_reconnect_policy

Что должно произойти, когда ESP разъединен от AP

```
bool wifi_station_set_reconnect_policy (bool набор)
```

Включает:

- user_interface.h

wifi_station_set_hostname

Установите имя хоста DHCP устройства WiFi.

```
bool wifi_station_set_hostname (случайная работа *имя)
```

Включает:

- user_interface.h
-

WiFi SoftAP

Следующие API касаются ESP* устройство, действующее как точка доступа на внешние станции.

wifi_softap_dhcps_start

Начните обслуживание сервера DHCP.


```
bool wifi_softap_dhcps_start ()
```

Начните обслуживание сервера DHCP в

устройстве. Включает:

- user_interface.h Видят также:

- Сервер DHCP
- wifi_softap_dhcps_stop
- wifi_softap_dhcps_offer_option

wifi_softap_dhcps_status

Возвратите статус обслуживания сервера DHCP.

```
enum dhcp_status wifi_softap_dhcps_status ()
```

Восстановите статус обслуживания сервера DHCP. Возвращенная стоимость будет одним из:

- DHCP_STOPPED
- DHCP_STARTED включает:
- user_interface.h Видят также:

- Сервер DHCP
- wifi_softap_dhcps_stop
- wifi_softap_dhcps_offer_option

wifi_softap_dhcps_stop

Остановите обслуживание сервера DHCP.

```
bool wifi_softap_dhcps_stop ()
```

Остановите обслуживание сервера

DHCP в устройстве. Включает:

- user_interface.h Видят также:

- Сервер DHCP
- wifi_softap_dhcps_offer_option

wifi_softap_free_station_info

Выпустите данные, связанные со `структурой station_info`.

```
пустота wifi_softap_free_station_info ()
```

Страница 354

После требования к `wifi_softap_get_station_info ()` нам можно было возвратить данные нам. Данные были ассигнованы OS, и мы должны возвратить его с этим вызовом функции. Обратите внимание, что эта функция **не** берет в данных, которые были возвращены.

Включает:

- user_interface.h Видят также:

- Быть точкой доступа
-

wifi_softap_get_config

Восстановите ток softAP детали конфигурации.

```
bool wifi_softap_get_config (структура softap_config *pConfig)
```

Когда ее назвали, `структура softap_config` указала, чтобы быть `pConfig`, будет заполнено в деталями тока softAP конфигурация. Детали возвратились, детали на самом деле в употреблении и могут отличаться от тех спасенных для дефолта.

Ценность 1 будет возвращена на успехе и 0

иначе. Включает:

- user_interface.h Видят также:

- структура softap_config
- wifi_softap_get_config_default
- wifi_softap_set_config_current

wifi_softap_get_config_default

Восстановите дефолт softAP детали конфигурации.

```
bool wifi_softap_get_config_default (структура softap_config *конфигурация)
```

Когда ее назвали, `структура softap_config` указала, чтобы быть `pConfig`, будет заполнено в деталями дефолта softAP конфигурация. Детали возвратились, используемые в ботинке и могут отличаться от тех использующихся в настоящее время.

Ценность 1 будет возвращена на успехе и 0

иначе. Включает:

- user_interface.h Видят также:

- структура softap_config
- wifi_softap_set_config_current

wifi_softap_get_dhcps_lease

wifi_softap_get_dhcps_lease_time

Получите временную стоимость арендного договора сервера DHCP.

```
uint32 wifi_softap_get_dhcps_lease_time ()
```

Возвратите число минут, которыми будет проводиться сервер, который DHCP арендуют IP-адресу.

wifi_softap_get_station_info

Возвратите детали всех связанных станций.

```
структура station_info *wifi_softap_get_station_info ()
```

Данные о возвращении - связанный список `структуры`

`station_info` структуры данных. Включает:

- user_interface.h Видят также:

- Быть точкой доступа
- wifi_softap_get_station_num

wifi_softap_get_station_num

Возвратите количество станций, в настоящее время связанных.

```
uint8 wifi_softap_get_station_num ()
```

Возвращает количество станций, в настоящее время связанных. Максимальное количество связей на ESP8266 равняется 4, но мы можем уменьшить это в softAP конфигурации в случае необходимости.

Включает:

- user_interface.h Видят также:

- Быть точкой доступа

wifi_softap_reset_dhcps_lease_time

Перезагрузите время арендного договора сервера DHCP к значению по умолчанию.

```
bool wifi_softap_reset_dhcps_lease_time ()
```

Перезагрузите время арендного договора сервера DHCP к значению по умолчанию, которое является в настоящее время 120 минутами.

wifi_softap_set_config

Установите ток и дефолт softAP конфигурация.

```
bool wifi_softap_set_config (структура softap_config *конфигурация)
```

Когда ее назвали, `структура softap_config` указала, чтобы быть `pConfig`, будет использоваться в качестве деталей дефолта и тока softAP конфигурация.

Ценность 1 будет возвращена на успехе и 0

иначе. Включает:

- user_interface.h Видят также:

- структура softap_config
- wifi_softap_get_config_default
- wifi_softap_set_config_current

wifi_softap_set_config_current

Установите дефолт softAP конфигурация.

```
bool wifi_softap_set_config_current (структура softap_config *конфигурация)
```

Когда ее назвали, `структура softap_config` указала, чтобы быть `pConfig`, будет использоваться в качестве деталей тока softAP конфигурация, но не будет сохранен как дефолт.

Включает:

- user_interface.h Видят также:

- структура softap_config
- wifi_softap_get_config_default
-

wifi_softap_set_dhcps_lease

Определите диапазон IP-адреса, который будет арендован этим сервером DHCP.

```
bool wifi_softap_set_dhcps_lease (структура dhcps_lease *, пожалуйста)
```

`Пожалуйста`, параметр - указатель на `структуру dhcps_lease`, который содержит диапазон IP-адреса IP-адресов, которые будут арендованы этим сервером DHCP. Различие между верхним и более низким, связанным IP-адресов, должно быть 100 или меньше. Эта функция не вступит в силу, пока сервер DHCP не будет остановлен и перезапущен (предположение, что это уже бежит).

Включает:

-

user_interface

.h Видят

также:

- Сервер DHCP
- wifi_softap_dhcps_stop
- wifi_softap_dhcps_offer_option
- структура dhcps_lease

wifi_softap_set_dhcps_lease_time

Установите время арендного договора сервера DHCP.

```
bool wifi_softap_set_dhcps_lease_time (uint32 минуты)
```

Набор, сколько времени арендный договор IP-адреса DHCP хорош для. дефолт составляет 120 минут. Параметр - число минут, которыми должен быть проведен арендный договор. У этого есть допустимый диапазон 1-2880.

wifi_softap_dhcps_offer_option

Набор варианты сервера DHCP.

```
bool wifi_softap_set_dhcps_offer_option (uint8 уровень, пустота *optarg)
```

В настоящее время параметр `уровня` может только быть `OFFER_ROUTER` с `optarg`, являющимся небольшой маской с ценностями:

- `0b0` – Отключают информацию о маршрутизаторе.
- `0b1` – Позволяют информацию о маршрутизаторе.

Включает:

- user_interface.h Видят также:
- wifi_softap_dhcps_stop

WiFi WPS

wifi_wps_enable

```
bool wifi_wps_enable (WPS_TYPE_t wps_type)
```

Параметр типа может быть одним из следующего:

- `WPS_TYPE_DISABLE` – неподдержанный
- `WPS_TYPE_PBC` – конфигурация кнопки – поддержанный
- `WPS_TYPE_PIN` – неподдержанный
- `WPS_TYPE_DISPLAY` – неподдержанный
- `WPS_TYPE_MAX` – неподдержанный

См. также:

- WiFi защищенная установка – WPS

wifi_wps_disable

```
bool wifi_wps_disable ()
```

См. также:

- WiFi защищенная установка – WPS

wifi_wps_start

```
bool wifi_wps_start ()
```

См. также:

- WiFi защищенная установка – WPS

wifi_set_wps_cb

```
bool wifi_set_wps_cb (wps_st_cb_t отзыв)
```

Подпись функции обратного вызова:

```
пустота (*functionName) (международный статус)
```

Параметр статуса будет одним из:

- WPS_CB_ST_SUCCESS
- WPS_CB_ST_FAILED
- WPS_CB_ST_TIMEOUT Видят также:
- WiFi защищенная установка – WPS

API модернизации

system_upgrade_flag_check

Восстановите флаг статуса модернизации.

```
uint8 system_upgrade_flag_check ()
```

Возвращенная стоимость будет одним из:

- UPGRADE_FLAG_IDLE
- UPGRADE_FLAG_START
- UPGRADE_FLAG_FINISH

system_upgrade_flag_set

Установите флаг статуса модернизации.

```
пустота system_upgrade_flag_set (uint8 флаг)
```

Флаг может быть одним из:

- UPGRADE_FLAG_IDLE
- UPGRADE_FLAG_START
- UPGRADE_FLAG_FINISH

system_upgrade_reboot

Перезагрузите ESP8266 и управляйте новым встроенным микропрограммным обеспечением.

```
пустота system_upgrade_reboot ()
```

system_upgrade_start

Начните загружать новое встроенное микропрограммное обеспечение с сервера.

```
bool system_upgrade_start (структура upgrade_server_info *сервер)
```

Параметр сервера - структура...

system_upgrade_userbin_check

Определите, какое из двух возможных микропрограммных изображений может быть модернизировано.

```
uint8 system_upgrade_userbin_check ()
```

Результатом будет или `UPGRADE_FW_BIN1` или `UPGRADE_FW_BIN2`.

API наркомана

wifi_promiscuous_enable

пустота wifi_promiscuous_enable (uint8 неразборчивый)

wifi_promiscuous_set_mac

пустота wifi_promiscuous_set_mac (константа uint8_t *адрес)

wifi_promiscuous_rx_cb

пустота wifi_promiscuous_rx_cb (wifi_promiscuous_cb_t cb)

wifi_get_channel

wifi_set_channel

Умные API конфигурации

smartconfig_start

bool smartconfig_start (sc_callback_t cb, uint8 регистрация)

smartconfig_stop

bool smartconfig_stop (пустота)

API SNTP

Обработайте Простой Сетевой запрос Протокола Времени.

sntp_setserver

Установите адрес сервера SNTP.

```
пустота sntp_serverserver (неподписанный индекс случайной работы, ip_addr_t
*addr)
```

Установите адрес одного из трех возможных серверов SNTP использоваться.

Параметр `индекса` должен быть или 0, 1 или 2 и определяет, какое из мест сервера SNTP должно быть установлено.

`addr` параметр - IP-адрес сервера SNTP, который будет зарегистрирован. Включает:

- sntp.h Видят также:
- Работа с SNTP

sntp_getserver

Восстановите IP-адрес сервера SNTP.

```
ip_addr_t sntp_getserver (неподписанный индекс случайной работы)
```

Восстановите IP-адрес ранее зарегистрированного сервера SNTP.

Параметр `индекса` - индекс сервера SNTP, который будет восстановлен. Это может быть или 0, 1 или 2.

Включает:

- sntp.h Видят также:
- Работа с SNTP

sntp_setservername

Установите имя хоста целевого сервера SNTP.

```
пустота sntp_setservername (неподписанный индекс случайной работы,
случайная работа *сервер)
```

Определите сервер SNTP его именем хоста.

Параметр `индекса` - индекс сервера SNTP, который будет установлен. Это может быть или 0, 1 или 2.

Параметр `сервера` - законченная последовательность ПУСТОГО УКАЗАТЕЛЯ, которая называет хозяина, который является сервером SNTP.

См. также:

- Работа с SNTP

sntp_getservername

Получите имя хоста целевого сервера SNTP.

```
случайная работа *sntp_setservername (неподписанный индекс случайной работы)
```

Восстановите имя хоста определенного сервера SNTP, который был ранее зарегистрирован.

Параметр `индекса` - индекс сервера SNTP, который был ранее установлен. Это может быть или 0, 1 или 2.

Возвращение из этой функции - законченная последовательность ПУСТОГО УКАЗАТЕЛЯ.

Включает:

- sntp.h Видят также:

- Работа с SNTP

sntp_init

```
пустота sntp_init ()
```

Инициализируйте функции SNTP.

Включает:

- snt p.h Вид ят так

же:

Страница 362

- Работа с SNTP

sntp_stop

пустот

a

sntp_s

top ()

Включ

ает:

- sntp.h Видят также:

- Работа с SNTP

sntp_get_current_timestamp

Получите текущую метку времени как неподписанные 32 битовых значения, представляющие число секунд[Св. с 1 января 1970] UTC.

```
uint32 sntp_get_current_timestamp ()
```

Включает:

- sntp.h Видят также:

- Работа с SNTP

sntp_get_real_time

случайная работа *sntp_get_real_time (длинный t)

????

Включает:

- sntp.h Видят также:

- Работа с SNTP

sntp_set_timezone

Установите текущий местный часовой пояс.

```
bool sntp_set_timezone (sint8 часовой пояс)
```

Призыв этой функции объявляет наш местный часовой пояс как

подписанное погашение в течение многих часов до UTC. Это нужно только назвать, когда функции SNTP не бегут что касается примера за требованием к `sntp_stop ()`.
Параметр `часового пояса` - часовой пояс в диапазоне-11 к 13.

Возвращаемое значение верное на успехе и ложное иначе.

Включает:

- sntp.h Видят также:
- Работа с SNTP

sntp_get_timezone

Получите текущий часовой пояс.

```
sint8 sntp_get_timezone ()
```

Восстановите текущую стоимость для часового пояса, как ранее установлено с требованием к `sntp_set_timezone ()`.
Включает:

- sntp.h Видят также:
- Работа с SNTP

Универсальные API TCP/UDP

espconn_delete

Удалите структуру блока управления.

```
sint8 espconn_delete (структура espconn *espconn)
```

Устройство поддерживает данные и хранение для каждого разговора (TCP и UDP). Когда эти разговоры закончены, и мы больше не собираемся общаться с партнерами, мы можем указать, что, вызывая эту функцию, которая выпустит внутреннюю память. Ожидается, что отказ сделать это приведет к утечкам памяти.

Код возврата 0 на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_ARG` – Незаконный аргумент

Этот API отменяет эффект `espconn_create` или `espconn_accept`.См. также:

- UDP

- espconn_create
- espconn_accept

espconn_dns_setserver

Установите дефолт сервер DNS.

Страница 364

```
пустота espconn_dns_setserver (случайная работа numdns, ip_addr_t
*dnsservers)
```

`numdns` - количество серверов DNS, поставляемых, который должен быть 1 или 2. Не больше, чем 2 сервера DNS могут поставляться. Эта функция не должна быть вызвана, если DHCP используется.

`dnsservers` параметр - множество 1 или 2 IP-адресов.См. также:

- Служба имен

espconn_gethostbyname

```
err_t espconn_gethostbyname (структура
       espconn *espconn, случайная работа
       константы *имя хоста,
       ip_addr_t *addr,
       dns_found_callba
       ck найденный)
```

Параметры:

- `espconn` – Уход и понимание необходимы, исследуя этот параметр.

  Так как это - `структура espconn`, мы немедленно думали бы, что это имеет некоторое отношение к коммуникациям и так или иначе используется, чтобы управлять `espconn_gethostbyname ()` функция. Ответ намного намного более прост. Это проигнорировано. Да … операция `gethostbyname ()` **не** зависит от этого параметра вообще. Это однако обнаруживается в еще одном месте. Когда функция обратного вызова призвана в результате того, что закончила `gethostbyname` …, параметр `аргумента` к отзыву установлен, чтобы быть ценностью этого `espconn` параметра. Так в действительности было бы, возможно, лучше определить тип данных этого первого параметра, чтобы быть «`пустотой *`» как в основном, именно так это использовало.
- `имя хоста` – имя хозяина поиска.
- `addr` – адрес склада, куда IP-адрес будет помещен **только если**

это было недавно подвергнуто сомнению прежде и проводится в тайнике. Адрес, найденный здесь, действителен, если `ESPCONN_OK` возвращен.

- `найденный` – функция обратного вызова, которая будет призвана, когда адрес будет
  решенный. Отзыв будет призван, только если

`ESPCONN_INPROGRESS` возвращен. `dns_found_callback` - функция со следующей подписью:

```
пустота      (*functionName) (случайная работа константы *имя, ip_addr_t
*ipAddr, пустота *аргумент)
```

то, где параметр `аргумента` - указатель на `структуру espconn`, `имя` - разыскиваемое имя хоста, и `ipAddr` - адрес IP-адреса, раньше хранило результат.

Когда имя хоста не может быть найдено, `ipAddr` возвращен как ПУСТОЙ УКАЗАТЕЛЬ … однако, Ваш поставщик DNS может обеспечить IP-адрес поисковой системы, и следовательно Вы вернете адрес …, но не один хозяину, которого Вы ожидали!!

Код возврата `0` на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_OK` – следовавший.
- `ESPCONN_INPROGRESS` – Указывает, что у нас нет тайника, и мы должны
  поиск.
- `ESPCONN_ARG` – Незаконный аргумент.

См. также:

- Служба имен
- lwIP –DNS

espconn_port

uint32 espconn_port ()

espconn_regist_sentcb

Зарегистрируйте функцию обратного вызова, которую назовут, когда

данные послали.

```
sint8 espconn_regist_sentcb
        (структура espconn
        *espconn,
        espconn_sent_callback sent_cb)
```

Формат функции обратного вызова:

```
пустота (*functionName) (пустота *аргумент)
```

Параметр `аргумента` - указатель на `структуру espconn`, который описывает связь.См. также:

- Отправка и получение данные TCP
- структура espconn

espconn_regist_recvcb

Зарегистрируйте функцию, которую назовут, когда данные станут доступными на соединении по протоколу TCP или дейтаграмме UDP.

```
sint8 espconn_regist_recvcb
        (структура espconn
        *espconn,
        espconn_recv_callback recv_cb)
```

Формат функции обратного вызова:

```
пустота (*functionName) (пустота *аргумент, случайная работа *pData,
короткое целое без знака len)
```


Где `args` - указатель на `структуру espconn`, `pData` - указатель на полученные данные, и `len` - длина полученных данных.
Код возврата 0 на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_ARG` – Незаконный аргумент

См. также:

- Отправка и получение данные TCP
- UDP
- espconn_create
- структура espconn

espconn_send

Пошлите данные через связь с партнером.

```
sint8 espconn_send (
        структура
        espconn
        *pEspconn, uint8
```

```
*pBuffer,
длина uint16)
```

`pEspconn` параметр определяет связь, через которую можно передать данные. `pBuffer` параметр указывает на буфер данных, который будет передан.

Параметр `длины` поставляет длину данных в байтах, которые должны быть переданы.

Обратите внимание, что данные не должны быть немедленно переданы. Мы можем быть уведомлены, когда данные были переданы отзывом к функции, зарегистрированной в `espconn_regist_sentcb ()`.

Код возврата 0 на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_MEM (-1)` – Из памяти
- `ESPCONN_ARG (-12)` – Незаконный аргумент

См. также:

- Отправка и получение данные TCP
- UDP
- espconn_regist_sentcb

espconn_sendto

sin16 espconn_sendto (структура espconn *espconn, uint8 *потраченный, uint16 длина)

ipaddr_addr

Постройте адрес TCP/IP из пунктирного десятичного представления последовательности.

```
uint32 ipaddr_addr (случайная работа *addressString)
```

Страница 367

Возвратите IP-адрес (4-байтовая) стоимость из пунктирного десятичного представления последовательности, поставляемого в `addressString` параметре. Обратите внимание, что тип uint32 не присваиваемый адресам в esp_tcp или esp_udp структуре. Вместо этого мы должны использовать местную переменную и затем скопировать содержание. Например:

```
uint32 addr =
ipaddr_addr (сервер);
```

```
memcpy
(m_tcp.remote_ip,
&addr, 4);
```

IP4_ADDR

Установите значение переменной к IP-адресу от его десятичного представления.

```
IP4_ADDR (структура ip_addr * addr, a, b, c, d)
```

`addr` параметр - указатель на хранение, чтобы держать IP-адрес. Это может быть случаем `структуры ip_addr`, `uint32`, `uint8[4]`. Это уже `должно быть брошено к указателю на структуру ip_addr` если не того типа.

Параметры `a`, `b`, `c` и `d` являются частями IP-адреса, если он был написан в пунктирном десятичном примечании.

Включает:

- ip_addr.h Видят также:
- структура ip_addr

IP2STR

Произведите четыре международных ценности, используемые в `os_printf` заявлении

```
IP2STR (ip_addr_t *адрес)
```

Это - макрос, который берет указатель на IP-адрес и возвращается, четыре запятая отделила десятичные значения, представляющие 4 байта IP-адреса. Это обычно используется в кодексе, таком как:

```
os_printf (» %d. % d. % d. % d\n», IP2STR (&addr));
```

API TCP

espconn_abort

Вызовите завершение связи TCP/IP.

```
sint8 espconn_abort (структура espconn *espconn)
```

Этот API нельзя назвать ни в каких espconn

функциях обратного вызова. Код возврата 0

указывает на успех.

espconn_listen.

espconn_accept

Прислушайтесь к поступающему соединению по протоколу TCP.

```
sint8 espconn_accept (структура espconn *espconn)
```

После вызывания этой функции ESP8266 начинает прислушиваться к поступающим связям. Любые функции обратного вызова, зарегистрированные в `espconn_regist_connectcb ()`, будут призваны, когда новые связи прибудут.

Код возврата 0 на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_MEM` – Из памяти
- `ESPCONN_ISCONN` – Уже соединился
- `ESPCONN_ARG` – Незаконный аргумент

Примечание: После некоторого размышления я думаю, что мне действительно не нравится название этого. То, что делает эта функция, заставить ESP8266 начинать слушать на местном порте для новых поступающих запросов. По существу делая ESP8266 сервером. Когда мы изучаем API гнезд, мы находим, что эквивалентный вызов функции достигнуть этой задачи называют, `слушают`. Так мой предложил/рекомендовал, что новое название этой функции будет

Таким образом, где тогда сделал `принять` имя, прибывшее от? Ответ - то, что в API гнезд есть вызванная функция партнера, `принимают`. Когда они выполнены против гнезда, которое ранее имело, `слушают` названные против него, что это делает заблокировать, пока партнер на самом деле не пытается соединиться. В ESP8266 нет никакого эквивалента. Вместо этого после того, как `espconn_accept` называют, ESP8266 немедленно начинает слушать и когда партнер соединяется, мы просыпаемся в соединить отзыве. Так... `действительно ли espconn_accept`, гнезда `слушают ()` требование или, гнезда `внимают ()` призыву? Мой ум говорит, что это НАМНОГО ближе к `слушанию ()` требование.

См. также:

- TCP
- espconn_regist_connectcb
- espconn_delete

espconn_get_connection_info

```
sint8
        espconn_get_connection_
        info (структура espconn
        *espconn, remot_info **
```

```
pcon_info, uint8
typeFlags)
```

`espconn` - указатель на блок управления TCP.

`pcon_info` параметр - информация о партнере.

`typeFlags` определяет, о каком партнере мы получаем информацию:

- 0 – постоянный партнер
- 1 – партнер SSL

Код возврата 0 на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_ARG` – Незаконный аргумент

espconn_connect

Соединитесь с удаленным применением, используя TCP.

```
sint8 espconn_connect (структура espconn *espconn)
```

Код возврата `0` на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_RTE (-4)` – Направление проблемы
- `ESPCONN_MEM (-1)` – Из памяти
- `ESPCONN_ISCONN (-15)` – Уже соединился
- `ESPCONN_ARG (-12)` – Незаконный аргумент

Поймите, что после совершения этого звонка, мы все еще можем не соединиться. Это - асинхронное требование, которое будет выполняться в более позднее время. Если будет неудача в том пункте, то мы найдем, что отзыв, зарегистрированный в `espconn_regist_reconcb ()`, будет призван.

Когда связь будет установлена, любой зарегистрированный отзыв, сделанный с `espconn_regist_connect ()`, будет призван.

См. также:

- TCP
- espconn_disconnect
- espconn_regist_connectcb
- espconn_regist_disconcb
- espconn_regist_reconcb

espconn_disconnect

Разъедините соединение по протоколу TCP.

```
sint8 espconn_disconnect (структура espconn *espconn)
```

Разъедините соединение по протоколу TCP, которое было ранее сформировано с `espconn_connect ()` или `espconn_accept ()`. Когда разъединение будет иметь успех, мы будем видеть отзыв к функции, зарегистрированной в `espconn_regist_disconcb ()`.
Код возврата `0` на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_ARG` – Незаконный аргумент

См. также:

- TCP
- espconn_accept
- espconn_connect
- espconn_regist_disconcb

espconn_regist_connectcb

Зарегистрируйте функцию, которая будет вызвана, когда соединение по протоколу TCP будет сформировано.

```
sint8 espconn_regist_connectcb
        (структура espconn
        *espconn,
        espconn_connect_callback connect_cb)
```

Код возврата `0` на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_ARG` – Незаконный аргумент

У функции обратного вызова должна быть следующая подпись:

```
пустота (*functionName) (пустота *аргумент)
```

Где параметр `аргумента` - указатель на `структуру espconn`

случай. Вопрос: действительно ли это - НОВАЯ `структура`

`espconn` или оригинальная?

См. также:

- espconn архитектура
- espconn_accept
- espconn_connect

espconn_regist_disconcb

Зарегистрируйте функцию, которая будет призвана обратно после разъединения TCP.

```
sint8 espconn_regist_disconcb
        (структура espconn
        *espconn,
        espconn_connect_callback discon_cb)
```

Подпись функции обратного вызова разъединения совпадает с соединить отзывом:

```
пустота (*functionName) (пустота *аргумент)
```

где `аргумент` - `структура`

`espconn` указатель.См. также:

- TCP
- espconn архитектура
- espconn_accept
- espconn_connect
- espconn_disconnect

espconn_regist_reconcb

Зарегистрируйте функцию, которая будет вызвана, когда ошибка будет обнаружена.


```
sint8 espconn_regist_reconcb
        (структура espconn
        *espconn,
        espconn_reconnect_callback recon_cb)
```

Этот отзыв призван, когда ошибка обнаружена. Например, пытаясь соединиться с партнером, который не слушает. Вероятно, что название этой функции было просто ужасно выбрано. См.:

http://bbs.espressif.com/viewtopic.php?f=66

&t=1063 подпись функции обратного

вызова:

```
пустота (*functionName) (пустота *аргумент, sint8 допускают ошибку),
```

Параметр `аргумента` - указатель на

`структуру espconn. Допускать ошибку`

параметр - одно из следующего:

- ESPCONN_TIMEOUT (-3)
- ESPCONN_ABRT (-8)
- ESPCONN_RST (-9)
- ESPCONN_CLSD (-10)

- `ESPCONN_CONN (-11)` – Неудавшееся соединение с партнером
- ESPCONN_HANDSHAKE (-28)
- ESPCONN_PROTO_MSG??

Вопрос: Что означает к статусу связи, если мы получаем ошибочный признак? Мы должны тогда попытаться разъединить, или мы уже разъединены? См.:
http://www.esp8266.com/viewtopic.php?f=9&t=5864

См. также:

- espconn архитектура
- TCP
- espconn_accept
- espconn_connect
- структура espconn

espconn_regist_write_finish

Зарегистрируйте функцию обратного вызова, которая будет призвана, когда данные будут успешно переданы партнеру.

```
sint8 espconn_regist_write_finish (структура espconn
      *espconn, espconn_connect_callback
      write_finish_cb);
```


Подпись отзыва:

```
пустота (*functionName) (пустота *аргумент)
```

Параметр `аргумента` - указатель на

`структуру espconn`.См. также:

- espconn архитектура
- espconn_send

espconn_set_opt

Определите который варианты включить для связи.

```
sint8 espconn_set_opt (
      структура
      espconn
      *espconn, uint8
```

```
выбирают),
```

Эта функция должна быть вызвана в `espconn_connect_callback`.
espconn параметр - блок управления для связи, которая должна быть изменена.
Выбирать параметр немного кодирует флагов, которые должны быть установлены на. Выбирать параметр - enum типа `espconn_option`:

Имя Enum	Стоимость
ESPCONN_REUSEADDR	0x01
ESPCONN_NODELAY	0x02
ESPCONN_COPY	0x04
ESPCONN_KEEPALIVE	0x08

Биты, которые не установлены на, оставлены без изменений от их текущих существующих ценностей. Код возврата 0 на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_ARG` – Незаконный аргумент

См. также:

- espconn_clear_opt
- espconn_set_keepalive
- espconn_get_keepalive

espconn_clear_opt

Определите который варианты выключить для связи.

```
sint8 espconn_clear_opt
       (структура
       espconn
       *espconn, uint8
       выбирают),
```

Код возврата 0 на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_ARG` – Незаконный аргумент

Выбирать стоимость - enum типа `espconn_option`:

Имя Enum	Стоимость
ESPCONN_REUSEADDR	0x01

ESPCONN_NODELAY	0x02
ESPCONN_COPY	0x04
ESPCONN_KEEPALIVE	0x08

См. также:

- Обработка ошибок TCP
- espconn_set_opt
- espconn_set_keepalive
- espconn_get_keepalive

espconn_regist_time

Определите неработающую стоимость перерыва связи.

```
sint8
        espconn_regist_t
        ime (структура
        espconn
        *espconn, uint32
        интервал,
        uint8 typeFlag)
```

Если связь без работы сроком на время, ESP8266 настроен, чтобы автоматически закрыть связь. Кажется, что дефолт составляет 10 секунд.

`espconn` параметр описывает связь, которой нужно было изменить ее перерыв.

Параметр `интервала` определяет интервал перерыва в секундах. Максимальное значение составляет 7 200 секунд (2 часа).

`typeFlag` параметр может быть 0, чтобы указать, что все связи должны быть изменены или 1, чтобы установить просто эту связь.

Код возврата 0 на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_ARG` – Незаконный аргумент

См. также:

- TCP

espconn_set_keepalive

sint8 espconn_set_keepalive (структура espconn *espconn, uint8 уровень, пустота *optArg)

espconn_get_keepalive

sint8 espconn_get_keepalive (структура espconn *espconn, uint8 уровень, пустота *optArg)

???

espconn_secure_accept

Прислушайтесь к поступающему соединению по протоколу TCP SSL

```
sint8 espconn_secure_accept (структура espconn *espconn)
```

Код возврата `0` на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_MEM` – Из памяти
- `ESPCONN_ISCONN` – Уже соединился
- `ESPCONN_ARG` – Незаконный аргумент

espconn_secure_ca_disable

bool espconn_secure_ca_disable (uint8 уровень)

espconn_secure_ca_enable

bool espconn_secure_ca_enable (uint8 уровень, uint16 flash_sector)

espconn_secure_set_size

espconn_secure_get_size

espconn_secure_delete

Удалите связь SSL, бегая, поскольку SSL основывал сервер.

```
sint8 espconn_secure_delete (структура espconn *espconn)
```

Код возврата 0 указывает на успех.

espconn_secure_connect

Сформируйте связь SSL с партнером.

```
sint8 espconn_secure_connect (структура espconn *espconn)
```

Сформируйте связь SSL с партнером.

Код возврата 0 на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_MEM` – Из памяти

- `ESPCONN_ISCONN` – Уже соединился
- `ESPCONN_ARG` – Незаконный аргумент

espconn_secure_send

Пошлите данные посредством безопасного соединения.

```
sint8 espconn_secure_send (структура espconn *espconn, uint8 *pBuf, uint16 длина)
```

Пошлите данные посредством безопасного соединения.

espconn_secure_disconnect

Обеспечьте разъединение TCP.

```
sint8 espconn_secure_disconnect (структура espconn *espconn)
```

Обеспечьте разъединение TCP.

Не вызывайте эту функцию из функции обратного вызова ESP.

espconn_tcp_get_max_con

Возвратите максимальное количество параллельных соединений по протоколу TCP.

```
uint8 espconn_tcp_get_max_con ()
```

espconn_tcp_set_max_con

Определите максимальный номер параллельных

соединений по протоколу TCP sint8

espconn_tcp_set_max_con (uint8 цифра)

espconn_tcp_get_max_con_allow

Доберитесь максимальное количество клиентов TCP позволило соединяться прибывающий.

•

espconn_tcp_set_max_con_allow

Определите максимальный номер клиентов TCP, разрешенных

соединиться прибывающий.

espconn_recv_hold

Приостановите получение данные TCP.

```
sint8 espconn_recv_hold (структура espconn *espconn)
```


Приостановите получающие новые данные из-за TCP. Чтобы продолжить получать данные, можно использовать `espconn_recv_unhold` вызов функции.

•espconn_recv_unhold

espconn_recv_unhold

Откройте получение данные TCP.

```
sint8 espconn_recv_unhold (структура espconn *espconn)
```

Продолжите получать новые данные по TCP. Этот метод должен использоваться вместе с `espconn_recv_hold`, который приостанавливает квитанцию данных.

См. также:

•espconn_recv_hold

API UDP

espconn_create

Создайте блок управления UDP при подготовке к отправке дейтаграмм.

```
sint8 espconn_create (структура espconn *espconn)
```

Код возврата 0 на успехе иначе кодекс указывает на ошибку:

- `ESPCONN_ARG` – Незаконный аргумент
- `ESPCONN_ISCONN` – Уже соединился
- `ESPCONN_MEM` – Из памяти

См. также:

- UDP
- espconn_regist_sentcb
- espconn_regist_recvcb
- espconn_send
- espconn_delete
- espconn_connect

espconn_igmp_join

Присоединитесь к группе передачи.

espconn_igmp_leave

Оставьте группу передачи.

API звона

ping_start

```
bool ping_start (структура ping_option *ping_opt)
```

Включает:

- ping.h Видят также:

- Запрос звона
- структура ping_option

ping_regist_recv

```
bool ping_regist_recv (структура ping_option *ping_opt, ping_recv_function
ping_recv)
```

Зарегистрируйте функцию, которая будет вызвана, когда звон будет получен. Подпись функции:

```
пустота (*functionName) (пустота* pingOpt, пустота *pingResp)
```

Параметры прошли в, `pingOpt`, который является указателем на `структуру ping_option` и `pingResp`, который является указателем на `структуру ping_resp`.

Включает:

- ping.h Видят также:

- Запрос звона
- структура ping_option
- структура ping_resp

ping_regist_sent

```
bool ping_regist_sent (структура ping_option *ping_opt, ping_sent_function
ping_sent)
```

Зарегистрируйте функцию, которая будет вызвана, когда звон пошлют. Подпись функции:

```
пустота (*functionName) (пустота* pingOpt, пустота *pingResp)
```

Параметры прошли в, `pingOpt`, который является указателем на `структуру ping_option` и `pingResp`, который является указателем на `структуру ping_resp`.

Включает:

- ping.h Видят также:

- Запрос звона
- структура ping_option

Страница 378

API mDNS

См. также:

- Системы доменных имен передачи

espconn_mdns_init

Intialize mDNS на ESP8266.

```
пустота espconn_mdns_init (структура mdns_info *информация)
```

Структура `структуры` типа `mdns_info` содержит жизненную информацию об инициализации и должна быть закончена прежде, чем вызвать эту функцию.

См. также:

- структура mdns_info

espconn_mdns_close

Закройте поддержку mDNS.

```
пустота espconn_mdns_close ()
```

Закройте поддержку mDNS. Это может использоваться после требования к `espconn_mdns_init ()`.См. также:

- espconn_mdns_init

espconn_mdns_server_register

Зарегистрируйте mDNS сервер.

```
пустота espconn_mdns_server_register ()
```

espconn_mdns_server_unregister

Не зарегистрируйте mDNS сервер.

```
пустота espconn_mdns_server_unregister ()
```

espconn_mdns_get_servername

Получите mDNS имя сервера.

```
случайная работа *espconn_mdns_get_servername ()
```

espconn_mdns_set_servername

Установите mDNS имя сервера.


```
случайная работа *espconn_mdns_set_servername ()
```

espconn_mdns_set_hostname

Установите mDNS имя хоста.

```
пустота espconn_mdns_set_hostname (случайная работа *имя)
```

espconn_mdns_get_hostname

Получите mDNS имя хоста

```
случайная работа *espconn_mdns_get_hostname ()
```

espconn_mdns_disable

Отключите mDNS.

```
пустота espconn_mdns_disable ()
```

См. также:

- espconn_mdns_enable

espconn_mdns_enable

Позвольте mDNS

```
пустота espconn_mdns_enable ()
```

См. также:

- espconn_mdns_disable

GPIO - ESP32

Функции gpio в ESP32 обеспечены через ESP-IDF. Нужно включать `«driver/gpio.h»` заголовок.

gpio_config

```
esp_err_t gpio_config (gpio_config_t *pGPIOConfig)
```

`gpio_config_t` структура данных содержит:


```
        способ uint64
     _t pin_bit_mask
     gpio_mode_t
   gpio_pullup_t
 pull_up_en
 gpio_pulldown_t
 pull_down_en
 gpio_int_type_t
 intr_type
```

`pin_bit_mask` определяет, какие булавки мы настраиваем. Константы определены, чтобы помочь нам здесь. Например, если мы настраиваем GPIO34 и GPIO16, мы можем установить `pin_bit_mask` в `GPIO_Pin_16 | GPIO_Pin_34`, который является булевым «или» этих двух постоянных величин.

`Способ` используется, чтобы установить способ всех булавок, которые мы настраиваем. Допустимые ценности:

- `GPIO_MODE_INPUT`
- `GPIO_MODE_OUTPUT`
- `GPIO_MODE_OUTPUT_OD`
- `GPIO_MODE_INPUT_OUTPUT_OD`
- `GPIO_MODE_INPUT_OUTPUT`

`pull_up_en` позволяет внутренний резистор подтягивания. Допустимые ценности:

- `GPIO_PULLUP_ENABLE`
- `GPIO_PULLUP_DISABLE`

`pull_down_en` позволяет внутренний резистор со спуском. Допустимые

ценности:

- `GPIO_PULLDOWN_ENABLE`
- `GPIO_PULLDOWN_DISABLE`

`intr_type` настраивает, как перерывы обработаны для булавки. Допустимые ценности:

- `GPIO_INTR_DISABLE`
- `GPIO_INTR_POSEDGE`
- `GPIO_INTR_NEGEDGE`
- `GPIO_INTR_ANYEDGE`
- `GPIO_INTR_LOW_LEVEL`
- `GPIO_INTR_HIGH_LEVEL`

gpio_get_level

Восстановите уровень сигнала на булавке.

```
интервал gpio_get_level (gpio_num_t gpioNum)
```

Получите уровень сигнала на указанной булавке. Или 0 или 1.

gpio_input_get

Восстановите bitmask ценностей первых 32 GPIOs (0-31).

```
uint32_t gpio_input_get ()
```

gpio_input_get_high

Восстановите bitmask ценностей последних 10 GPIOs (32-39).

```
uint32_t gpio_input_get_high ()
```

gpio_intr_enable

Позвольте перерывы на указанной булавке.

```
esp_err_t gpio_intr_enable (gpio_num_t gpioNum)
```

gpio_intr_disable

Отключите перерывы на указанной булавке.

```
esp_err_t gpio_intr_disable (gpio_num_t gpioNum)
```

gpio_isr_register

Зарегистрируйте укладчика перерыва.

```
esp_err_t gpio_isr_register (uint32_t gpioIntr, пустота (*fn) (пустота *),
пустота *аргумент)
```


gpio_output_set

Набор GPIOs, который должен быть введен, произвел, высоко или низко на большой части

```
пустота gpio_output_set
    (uint32_t setMask,
    uint32_t clearMask,
    uint32_t
    enableMask, uint32
    _t disableMask)
```

Работа с первыми 32 GPIOs определение (0-31), которые введены, которые произведены, которые должны быть оставлены в покое и которые должны быть установлены высокий/низкий. Этот API позволяет нам установить партию GPIOs в одной операции.

gpio_output_set_high

Набор GPIOs, который должен быть введен, произвел, высоко или низко на большой части

```
пустота
    gpio_output_set_high
    (uint32_t setMask,
    uint32_t clearMask,
    uint32_t enableMask,
    uint32_t disableMask)
```

Работа с последними 10 GPIOs определение (32-39), которые введены, которые произведены, которые должны быть оставлены в покое и которые должны быть установлены высокий/низкий. Этот API позволяет нам установить партию GPIOs в одной операции.

gpio_set_direction

ESP32

Установите направление булавки.

```
esp_err_t gpio_set_direction (gpio_num_t gpioNum, gpio_mode_t способ)
```

`Способ` используется, чтобы установить способ булавки, которую мы настраиваем. Допустимые ценности:

- `GPIO_MODE_INPUT`
- `GPIO_MODE_OUTPUT`
- `GPIO_MODE_OUTPUT_OD`
- `GPIO_MODE_INPUT_OUTPUT_OD`
- `GPIO_MODE_INPUT_OUTPUT`

gpio_set_intr_type

ESP32

Установите тип перерыва булавки.

```
esp_err_t gpio_set_intr_type (gpio_num_t gpioNum, gpio_int_type_t intrType)
```

`intr_type` настраивает, как перерывы обработаны для булавки.
Допустимые ценности:

- `GPIO_INTR_DISABLE`
- `GPIO_INTR_POSEDGE`
- `GPIO_INTR_NEGEDGE`
- `GPIO_INTR_ANYEDGE`

- GPIO_INTR_LOW_LEVEL
- GPIO_INTR_HIGH_LEVEL

gpio_set_level

ESP32

Установите уровень булавки.

```
esp_err_t gpio_set_level (gpio_num_t gpioNum, uint32_t уровень)
```

Se уровень булавки продукции. Уровень должен быть или 0 или 1.

gpio_set_pull_mode

ESP32

Установите способ усилия / способ со спуском булавки.

```
esp_err_t gpio_set_pull_mode (gpio_num_t gpioNum, gpio_pull_mode_t
напряжение)
```

Допустимые ценности для напряжения:

- GPIO_PULLUP_ONLY
- GPIO_PULLDOWN_ONLY
- GPIO_PULLUP_PULLDOWN
- GPIO_FLOATING

GPIO - ESP8266

Имена булавки:

- 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

Имя булавки	Функция 1	Функция 2	Функция 3	Функция 4	Физическая булавка
MTDI_U	MTDI	I2SI_DATA	МИСО HSPIQ	GPIO12	10
MTCK_U	MTCK	I2SI_BCK	HSPID 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	SPID		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

Функции булавки:

- FUNC_GPIO0
- FUNC_GPIO12
- FUNC_GPIO13
- FUNC_GPIO14
- FUNC_GPIO15
- FUNC_U0RTS
- FUNC_GPIO3

Страница 385

- 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

Отключите подтягивание булавки PIN_PULLUP_DIS (PIN_NAME)

См. также:

- GPIOs

PIN_PULLUP_EN

Позвольте подтягивание булавки

PIN_PULLUP_EN (PIN_NAME)

См. также:

- GPIOs

PIN_FUNC_SELECT

Установите функцию определенной булавки.

PIN_FUNC_SELECT (PIN_NAME, FUNC)

См. также:

- GPIOs

GPIO_ID_PIN

Получите id логической булавки.

```
GPIO_ID_PIN (pinNum)
```

Преобразуйте логический ПИН-код в идентичность булавки. Это - интересная функция, поскольку `GPIO_ID_PIN (x)` закодирован, чтобы равняться «x». Вопрос теперь становится, нужно ли все еще закодировать `GPIO_ID_PIN ()`, получая доступ к функциям GPIO.

GPIO_OUTPUT_SET

Установите значение продукции определенной булавки.

```
GPIO_OUTPUT_SET (GPIO_NUMBER, стоимость)
```

Это - макрос помощника, который призывает `gpio_output_set ()`. Всего хорошего, проходя в стоимости, которая является частью выражения, такого как `pData == '1'`. Стоимость оценена неоднократно, так не должен иметь побочных эффектов. Есть также текущая ошибка, связанная с предшествованием оператора …, сильно рекомендуется поместить стоимость в дополнительную круглую скобку, кодируя. Например:

```
GPIO_OUTPUT_SET (GPIO_NUMBER, (pData == '1'))
```

Включает:

- gpio.h

Страница 387

См. также:

- GPIOs

GPIO_DIS_OUTPUT

Установите булавку быть введенной (отключенная продукция).

```
GPIO_DIS_OUTPUT (GPIO_NUMBER)
```

Это - макрос помощника, который призывает `gpio_output_set ()`.

Включает:

- gpio.h Видят также:

- GPIOs

GPIO_INPUT_GET

Прочитайте ценность булавки.

```
GPIO_INPUT_GET (GPIO_NUMBER)
```

Это - макрос помощника, который призывает `gpio_input_get ()`.

Включает:

- gpio.h Видят также:

- gpio_input_get

gpio_output_set

Измените ценности булавок GPIO в одной операции.

```
пустота gpio_output_set
       (uint32 set_mask,
       uint32 clear_mask,
       uint32
       enable_output,
       uint32 enable_input)
```

Параметры:

- `set_mask` – Биты с «1» установлены высоко, биты с «0» оставлены без изменений.
- `clear_mask` – Биты с «1» установлены низко, биты с «0» оставлены без изменений
- `enable_output` – Биты с «1» собираются произвести
- `enable_input` – Биты с «1» собираются ввести

Включает:

- gpio.h Видят также:

- GPIOs

gpio_input_get

Получите ценности GPIOs.

```
uint32 gpio_input_get ()
```

Восстановите ценности от GPIOs и возвратите bitmask из их

ценностей. Включает:

- gpio.h Видят также:

- GPIOs

gpio_intr_handler_register

Зарегистрируйте функцию обратного вызова, которая будет призвана, когда перерыв GPIO произойдет.

```
пустота gpio_intr_handler_register
        (gpio_intr_handler_fn_t callbackFunction,
        пустота *аргумент)
```

Подпись функции укладчика должна быть:

```
пустота (*functionName) (uint32 interruptMask, пустота *аргумент)
```

Включает:

- gpio.h Видят также:

- Обработка Перерыва GPIO

gpio_pin_intr_state_set

```
пустота gpio_pin_intr_state_set
        (uint32 pinId, GPIO_INT_TYPE
        intr_state)
```

`Тосковавшей` является идентификационная стоимость булавки GPIO, возвращенная из `GPIO_ID_PIN (цифра)`.

`intr_state` параметр определяет то, что вызывает

перерыв. Включает:

- gpio.h

Видят

также:

- Обработка Перерыва GPIO
- GPIOs

- GPIO_INT_TYPE

gpio_intr_pending

Получите набор ожидания перерывов

```
uint32 gpio_intr_pending ()
```

Включает:

- gpio.h Видят также:

- Обработка Перерыва GPIO

gpio_intr_ack

Флаг ряд перерывов, как обработанных. Это нужно назвать от функции укладчика перерыва.

```
пустота gpio_intr_ack (uint32 ack_mask)
```

Включает:

- gpio.h

gpio_pin_wakeup_enable

Определите это, устройство может пробуждение от легкого режима ожидания, когда перерыв IO происходит.

```
пустота
        gpio_pin_wakeup_en
        able (uint32
        булавка,
        GPIO_INT_TYPE
        intr_state)
```

Параметр `булавки` определяет ПИН-код, используемый, чтобы разбудить устройство.

`intr_state` определяет, какой тип перехода разбудит устройство. Выбор:

- GPIO_PIN_INTR_LOLEVEL
  - GPIO_PIN_INTR_HILEVEL включает:
- gpio.h

См. также:

- GPIOs
- GPIO_INT_TYPE

пустота
gpio_pin_wakeup_disabl
e
gpio_pin_wakeup_disabl
e ()
Включает:

- gpio.h

API UART

Эти функции должны быть собраны в из uart файлов в `driver_lib`.

UART_CheckOutputFinished

```
bool UART_CheckOutputFinished (uint8 uart_no, uint32 time_out_us)
```

UART_ClearIntrStatus

```
недействительный UART_ClearIntrStatus (uint8 uart_no, uint32 clr_mask);
```

UART_ResetFifo

```
недействительный UART_ResetFifo (uint8 uart_no);
```

UART_SetBaudrate

Установите скорость передачи в бодах.

```
недействительный UART_SetBaudrate (uint8 uart_no, uint32 baud_rate)
```

Установите скорость передачи в бодах, используемую UART. `uart_no` определяет UART, чтобы установить (0 или 1), и `baud_rate` - желаемая скорость передачи в бодах. У UARTs есть определения UART0 и UART1.

UART_SetFlowCtrl

```
недействительный UART_SetFlowCtrl (uint8 uart_no, UART_HwFlowCtrl
flow_ctrl, uint8 rx_thresh)
```

UART_SetIntrEna

```
недействительный UART_SetIntrEna (uint8 uart_no, uint32 ena_mask)
```

UART_SetLineInverse

```
недействительный UART_SetLineInverse (uint8 uart_no, UART_LineLevelInverse
inverse_mask)
```


UART_SetParity

Установите паритет.

```
недействительный UART_SetParity (uint8 uart_no, UartParityMode parity_mode)
```

Установите паритет, используемый UART. `uart_no` определяет UART, чтобы установить (0 или 1), и `parity_mode` определяет, что использовать.

UART_SetPrintPort

Установите терминал продукции.

```
недействительный UART_SetPrintPort (uint8 uart_no)
```

Установите терминал продукции. Установите UART использоваться, сочиняя отладку через `os_printf ()`. У UARTs есть определения UART0 и UART1.

UART_SetStopBits

Набор, какой длины биты остановки должны быть.

```
недействительный UART_SetStopBits (uint8 uart_no, UartStopBitsNum bit_num)
```

Набор, какой длины биты остановки должны быть. `Цифра` определяет число битов остановки, чтобы использовать.

UART_SetWordLength

Определите номер битов в единице передачи.

```
недействительный UART_SetWordLength (uint8 uart_no, UartBitsNum4Char len)
```

Определите номер битов в единице передачи. `uart_no` определяет UART, чтобы установить (0 или 1), и `len` параметр определяет сколько битов.

UART_WaitTxFifoEmpty

Ждите, чтобы буфер TX опустел.

```
недействительный UART_WaitTxFifoEmpty (uint8 uart_no, uint32 time_out_us)
```

Ждите, чтобы буфер TX опустел. `uart_no` определяет UART, чтобы установить (0 или 1), и `time_out_us` определяет, сколько времени ждать перед отказом. Стоимость поставляется в микросекундах.

uart_init

```
пустота uart_init (UartBautRate uart0BaudRate, UartBautRate uart1BaudRate)
```


Кажется, есть опечатка в типе данных …, но вероятно мы застрянем с этим теперь. `UartBautRate` - enum, который содержит:

- 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 Видят также:
- Работа с сериалом

uart0_tx_buffer

Передайте буфер данных через UART0.

```
пустота uart0_tx_buffer (uint8 *буфер, uint16 длина)
```

Передайте данные, на которые указывает `буфер` для данной `длины`.См. также:

- Работа с сериалом

uart0_sendStr

Передайте ряд данных через UART0.

```
пустота uart0_sendStr (случайная работа константы *ул.)
```

Передайте ряд данных через UART0. Последовательность, чтобы послать поставляется в параметре `ул`.

uart0_rx_intr_handler

Обращайтесь с получением данных через UART0.

```
пустота uart0_rx_intr_handler (пустота *параметр)
```


Параметр - указатель на структуру `RcvMsgBuff`. Мое лучшее предположение о том, как использовать эту функцию, должно создать ее в `user_main.c`, и его простое существование заставит ее быть призванной в подходящее время. Смотря на поставляемый образец, мы видим, что ему нужно подробное внедрение низкого уровня.

См. также:

- Работа с сериалом

API владельца I2C

Эти функции должны быть собраны в из i2c_master файлов в driver_lib.См. также:

- Работа с I2C

i2c_master_checkAck

Восстановите ack от шины данных и возвратитесь верный или ложный.

```
bool i2c_master_checkAck ()
```

Восстановите ack от шины данных и возвратитесь верный или ложный.

i2c_master_getAck

Восстановите ack от шины данных и возвратите ее стоимость.

```
uint8 i2c_master_getAck ()
```

Восстановите ack от шины данных и возвратите ее стоимость. Не ясно, почему эта функция могла бы быть выставлена, а также `i2c_master_checkAck ()`.

i2c_master_gpio_init

Настройте GPIOs и затем назовите `i2c_master_init ()`.

```
пустота i2c_master_gpio_init ()
```

Настройте GPIOs и затем назовите `i2c_master_init ()`.

i2c_master_init

Инициализируйте функции I2C.

```
пустота i2c_master_init ()
```

Инициализируйте функции I2C.

Страница 394

i2c_master_readByte uint8

i2c_master_readByte ()

i2c_master_send_ack

пустота i2c_master_send_ack ()

пустота i2c_master_send_nack

i2c_master_send_nack ()

i2c_master_setAck

Набор ack к i2c автобусу как стоимость уровня.

```
пустота i2c_master_setAck (uint8 уровень)
```

Набор ack к i2c автобусу как стоимость уровня.

i2c_master_start

Набор I2C, чтобы послать государство.

```
пустота i2c_master_start ()
```

Набор I2C, чтобы послать государство.

i2c_master_stop

Набор I2C, чтобы прекратить посылать государство.

```
пустота i2c_master_stop ()
```

Набор I2C, чтобы прекратить посылать государство.

i2c_master_writeByte

пустота i2c_master_writeByte (uint8 wrdata)

API SPI

Эти функции должны быть собраны в из файлов SPI в driver_lib.

cache_flush

spi_lcd_9bit_write

spi_mast_byte_write

spi_byte_write_espsl

ave spi_slave_init

spi_slave_isr_handler

hspi_master_readwrit

e_repeat

spi_test_init

API PWM

pwm_init

Инициализируйте PWM.

```
пустота
       pwm_i
       nit
       (uint
       32
       перио
       д,
       uint3
       2
       *обяз
       аннос
       ть,
       uint32
       num_pwm_channels,
       uint32
       (*pin_info_list)
       [3])
```

Параметр `периода` - период PWM. Стоимость измеряется в микросекундах с минимальным значением 1 000 предоставления периода на 1 кГц (есть 1 000 периодов 1 000 микросекунд за секунду).

Параметр `обязанности` - отношение обязанности каждого канала PWM.

`num_pwm_channels` - количество определяемых каналов PWM. Может быть до каналов `PWM_CHANNEL_NUM_MAX`. В настоящее время это определено как 8.

`pin_info_list` - указатель на множество uint32s, который обеспечивает отображения булавки PWM.

`num_pwm_channels * 3` случая параметров за канал PWM:

- Регистр GPIO
- Повторное использование IO соответствующей булавки
- Число GPIO

Например:


```
(обязанность * 45) /
uint32 pinInfoList [] [3] =
  {{PERIPHS_IO_MUX_MTDI_U,
  FUNC_GPIO12, 12},
```

```
  {PERIPHS_IO_MUX_MTDO_U,
  FUNC_GPIO15, 15},
  {PERIPHS_IO_MUX_MTCK_U,
  FUNC_GPIO13, 13}
};
```

См. также:

- Модуляция ширины импульса – PWM
- pwm_set_duty
- pwm_set_period
- pwm_start

pwm_start

```
пустота pwm_start ()
```

После формирования параметров для PWM должна быть вызвана эта функция.См. также:

- Модуляция ширины импульса – PWM

pwm_set_duty

```
пустота pwm_set_duty (uint32 обязанность, uint8 канал)
```

Разрешение шага обязанности составляет 45 наносекунд. Здесь мы можем определить номер шагов обязанности в цикле. Например, предположите, что у нас есть период 1 кГц. Это означает, что 1 цикл составляет 1 000 микросекунд. Если мы хотим, чтобы рабочий цикл составил 50%, то производительность должна быть высокой в течение 500 микросекунд. 500 микросекунд - 11 111 единиц 45 наносекунд и это стало бы стоимостью обязанности. Шаблонным образом отношение обязанности

```
(период *1000).
```

Параметр `обязанности` поставляет количество интервалов с 45 наносекундами, что производительность будет высока в один период.

```
обязанность = 1000000 / 0.045 / частота
```

Параметр `канала` определяет, какой из каналов PWM изменяется.

После изменения стоимости обязанности требование к `pwm_start ()` требуется, чтобы повторно вычислять ценности.

См. также:

- Модуляция ширины импульса – PWM
- pwm_get_duty
- pwm_init

pwm_get_duty

```
uint32 pwm_get_duty (uint8 канал)
```

Получите ценность обязанности указанного канала.

См. также:

- Модуляция ширины импульса – PWM
- pwm_get_duty
- pwm_init

pwm_set_period

Установите период для операций PWM.

```
пустота pwm_set_period (uint32 период)
```

Параметр `периода` - период PWM. Стоимость измеряется в микросекундах с минимальным значением 1 000 предоставления периода на 1 кГц (есть 1 000 периодов 1 000 микросекунд за секунду).

См. также:

- Модуляция ширины импульса – PWM
- pwm_get_period
- pwm_init

pwm_get_period

```
uint32 pwm_get_period ()
```

Получите текущее

урегулирование периода

PWM.См. также:

- Модуляция ширины импульса – PWM
- pwm_set_period
- pwm_init

get_pwm_version

```
uint32 get_pwm_version ()
```

См. также:

- Модуляция ширины импульса – PWM

set_pwm_debug_en (uint8 print_en)

Используемый, чтобы позволить или отключить печать отладки.

Битовое жонглирование

- БИТ (b) – 2^b стоимость

ESP теперь

esp_now_add_pe

er

esp_now_deinit

esp_now_del_pee

r

esp_now_get_pee

r_key

esp_now_get_pee

r_role

esp_now_get_self

_role

esp_now_init

esp_now_register

_recv_cb

esp_now_register

_send_cb

esp_now_send

Максимальный объем данных, который можно послать как единица, составляет 256 байтов.

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

МАГАРЫЧИ ПРИ ТОРГОВОЙ СДЕЛКЕ

Когда звонок API сделан МАГАРЫЧАМ ПРИ ТОРГОВОЙ СДЕЛКЕ, он может возможно установить код ошибки, который может быть восстановлен требованием к `SPIFFS_errno ()`. Возвращенная стоимость может быть одним из следующего:

Символ	Стоимость	Значение
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	

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	Возможно попытка создать файл, который уже существует. Не мог также означать «ошибку».

См. также:

- Файловая система магарычей при торговой сделке

esp_spiffs_deinit

esp_spiffs_init

Инициализируйте МАГАРЫЧИ ПРИ ТОРГОВОЙ СДЕЛКЕ.


```
sint32 esp_spiffs_init (структура esp_spiffs_config *конфигурация)
```

Параметр конфигурации - структура, определяющая информацию об инициализации для МАГАРЫЧЕЙ ПРИ ТОРГОВОЙ СДЕЛКЕ. Это содержит:

- phys_size
- phys_addr
- phys_erase_block
- log_block_size
- log_page_size
- fd_buf_size
- cache_buf_size

Конфигурация в качестве примера могла бы быть:

```
структура esp_spiffs_config
```

```
конфигурация; конфигурация
phys_size = FS1_FLASH_SIZE;
конфигурация phys_addr = FS1
_FLASH_ADDR; конфигурация
phys_erase_block = SECTOR_SIZE;
конфигурация log_block_size =
LOG_BLOCK; конфигурация
log_page_size = LOG_PAGE;
конфигурация fd_buf_size =
FD_BUF_SIZE * 2; конфигурация
cache_buf_size = CACHE_BUF_SIZE;
```

Код возврата 0 успехов средств.

SPIFFS_check

Управляет проверкой на непротиворечивость в данной файловой системе.

```
s32_t SPIFFS_check (магарычи при торговой сделке *фс)
```

SPIFFS_clearerr

Очищает последнюю ошибку.

```
недействительный SPIFFS_clearerr (магарычи при торговой сделке *фс)
```

SPIFFS_close

Закрывает filehandle. Если там находятся на рассмотрении, пишут операции, они завершены перед закрытием.

```
недействительный SPIFFS_close (магарычи при торговой сделке *фс,
spiffs_file filehandle)
```

Закройте filehandle, который был ранее открыт с требованием к SPIFFS_open ().См. также:

- SPIFFS_open

SPIFFS_closedir

Закрывает директивный поток.

```
s32_t SPIFFS_closedir (spiffs_DIR *spiffsDir)
```

Директивный поток должен был быть ранее открыт с требованием к `SPIFFS_ope ndir ()`.См.

также:

- SPIFFS_opendir
- SPIFFS_readdir

SPIFFS_creat

Создайте определенный файл.

```
s32_t SPIFFS_creat (магарычи при торговой сделке *фс, случайная работа
*путь, spiffs_mode способ)
```

Каждый обычно использует `SPIFFS_open ()`, чтобы создать файл.

SPIFFS_erase_deleted_block

Сотрите удаленные блоки в файловой системе.

```
s32_t SPIFFS_erase_deleted_block (магарычи при торговой сделке *фс)
```

SPIFFS_errno

Получите последний код ошибки.

```
s32_t SPIFFS_errno (магарычи при торговой сделке *фс)
```

Восстановите последний код ошибки.

SPIFFS_fflush

Вспыхните все пишут операции от тайника до файловой системы.

```
s32_t SPIFFS_fflush (магарычи при торговой сделке *фс, spiffs_file
filehandle)
```

SPIFFS_format

Форматирует всю файловую систему.

```
s32_t SPIFFS_format (магарычи при торговой сделке *фс);
```

Все данные будут потеряны. Файловая система не должна быть установлена, называя это. Форматирование NB: неловкое. Из-за назад совместимости, `SPIFFS_mount` НУЖНО назвать

до форматирования, чтобы настроить файловую систему. Если `SPIFFS_mount` имеет успех, `SPIFFS_unmount` нужно назвать прежде, чем назвать `SPIFFS_format`. Если

`SPIFFS_mount` терпит неудачу,
`SPIFFS_format` можно назвать непосредственно, не называя `SPIFFS_unmount` сначала.

SPIFFS_fremove

Удалите файл его дескриптором.

s32_t SPIFFS_fremove (магарычи при торговой

сделке *фс, spiffs_file filehandle) Удаляют файл

его дескриптором.

SPIFFS_fstat

Получите статус файла дескриптором.

```
s32_t SPIFFS_fstat
  (магарычи при торговой
  сделке *фс, spiffs_file
  filehandle, spiffs_stat
  *spiffsStat)
```

spiffs_stat содержит:

- obj_id
- `размер` – размер содержания файла.
- напечатать
- `имя` – название файла.

SPIFFS_gc

Выполните явную сборку мусора.

```
s32_t SPIFFS_gc (магарычи при торговой сделке *фс, u32_t размер)
```

Призовите сборку мусора, чтобы гарантировать, что есть достаточно пространства для байтов размера.

SPIFFS_gc_quick

Выполните явную сборку мусора.

```
s32_t SPIFFS_gc_quick (магарычи при торговой сделке *фс, u16_t
max_free_pages)
```

SPIFFS_info

Возвратите объем хранения всего и означайте на самом деле используемые.

```
s32_t SPIFFS_info (магарычи при торговой сделке *фс, u32_t *общее
```

```
количество, u32_t *используемый)
```

`Полный` параметр - общее количество байтов в файловой системе.

`Используемый` параметр - сумма использованного пространства.

SPIFFS_lseek

Переместите погашение чтения-записи для файла.

```
s32_t SPIFFS_lseek (магарычи при торговой сделке *фс, spiffs_file
filehandle, s32_t погашение, интервал откуда)
```

- `фс` – файловая система, которая владеет файлом.
- `filehandle` – открытая ручка к файлу.
- `погашение` – сумма, чтобы переместиться в рамках файла.
- `откуда` – направление движения:
  - `SPIFFS_SEEK_SET` – Движение к определенному местоположению.
  - `SPIFFS_SEEK_CUR` – Движение относительно текущего местоположения.
  - `SPIFFS_SEEK_END` – Движение относительно конца файла.

SPIFFS_mount

Инициализирует файловую систему динамические параметры и устанавливает файловую систему. Если SPIFFS_USE_MAGIC позволяют, установка может потерпеть неудачу с `SPIFFS_ERR_NOT_A_FS`, если вспышка не содержит распознаваемую файловую систему. В этом случае `SPIFFS_format` нужно назвать до переустановки.

```
s32_t
  SPIFFS_mount
  (магарычи
  при торговой
  сделке *фс,
  spiffs_confi
  g
  *конфигураци
  я, u8_t
  *работа,
  u8_t *fd_space, u32_t
  fd_space_size, пустота
  *тайник, u32_t cache_size,
  spiffs_check_callback
  check_cb_f);
```

- `фс` – структура файловой системы.

- `конфигурация` – физическая и логическая конфигурация файловой системы.
- работа
- fd_space
- `fd_space_size` – Пример 32*4.
- тайник
- `cache_size` – Пример (128 + 32) * 8.
- check_cb_f

spiffs_config структура содержит:

- `hal_read_f` – физическая прочитанная функция. Это - функция с подписью:

s32_t func (u32_t addr, u32_t размер, u8_t *dst)

- `hal_write_f` – физический пишут функцию. Это - функция с подписью:

s32_t func (u32_t addr, u32_t размер, u8_t *src)

- `hal_erase_f` – физический стирают функцию. Это - функция с подписью:

s32_t func (u32_t addr, u32_t размер)

- `phys_size` – физический размер вспышки spi.
- `phys_addr` – физическое погашение во вспышке spi, используемой для магарычей при торговой сделке, должно быть на блоке
  граница.
- `phys_erase_block` – физический размер, стирая блок.
- `log_block_size` – логический размер блока, должен быть на физическом размере блока
  граница и никогда не должна быть меньше, чем физический блок.
  Пример 4*1024.
- `log_page_size` – логический размер страницы, должен быть, по крайней мере, log_block_size / 8.
  Пример 128.

SPIFFS_mounted

Проверки, установлена ли

файловая система. u8_t

SPIFFS_mounted (магарычи при

торговой сделке *фс)

Прибыль 0, если **не** установленный.

SPIFFS_open

Откройте файл.

```
spiffs_file
  SPIFFS_open
  (магарычи при
  торговой сделке *фс,
  случайная
  работа *путь,
  spiffs_flags
  флаги,
  spiffs_mode
  способ)
```

Откройте файл. Это может также включать создание файла, когда это открыто.

- `фс` – файловая система, чтобы открыться.
- `путь` – путь к файлу, чтобы открыться.
- `флаги` – флаги Контроля для открытия файла. Комбинация:
  - SPIFFS_APPEND
  - SPIFFS_CREAT

  - SPIFFS_DIRECT
  - SPIFFS_RDONLY
  - SPIFFS_RDWR
  - SPIFFS_TRUNC
  - SPIFFS_WRONLY
- `способ` – способ для открытого. Проигнорированный в этом выпуске.

См. также:

- SPIFFS_close

SPIFFS_open_by_dirent

Откройте файл его статьей каталога.

```
spiffs_file SPIFFS_open_by_dirent (магарычи при
  торговой сделке *фс,
  структура spiffs_dirent *spiffsDirEnt,
  spiffs_flags флаги,
  способ spiffs_mode)
```

Откройте файл.

- `фс` – файловая система, чтобы открыться.
- `spiffsDirEnt` – путь к файлу, чтобы открыться.
- `флаги` – флаги Контроля для открытия файла. Комбинация:
  - SPIFFS_APPEND
  - SPIFFS_DIRECT
  - SPIFFS_RDONLY
  - SPIFFS_RDWR
  - SPIFFS_TRUNC
  - SPIFFS_WRONLY

SPIFFS_opendir

Откройте директивный поток для определенного имени каталога.

```
spiffs_DIR *SPIFFS_opendir (магарычи при
  торговой сделке *фс,
  случайная работа *directoryName,
  spiffs_DIR *spiffsDir)
```

- `фс` – файловая система МАГАРЫЧЕЙ ПРИ ТОРГОВОЙ СДЕЛКЕ, которая будет работаться против.

- `directoryName` – название справочника, который будет прочитан.
- `spiffsDir` – структура каталогов, чтобы быть населяет.

См. также:

SPIFFS_read

Прочитайте данные из файла.

s32_t SPIFFS_read (магарычи при торговой сделке *фс,

spiffs_file filehandle, пустота *buf, s32_t len) данные Рида из

файла и места в буфере.

SPIFFS_readdir

Прочитайте справочник.

```
структура spiffs_dirent
  *SPIFFS_readdir (spiffs_DIR
  *spiffsDir,
  структура spiffs_dirent *spiffsDirEnt)
```

Прочитайте справочник, определенный `spiffsDir`, который был ранее открыт с

`SPIFFS_opendir ().`

`Структура spiffs_dirent` содержит:

- obj_id
- имя:
- напечатать
- размер
    - ящик для пробной монеты Видит также:

- Файловая система магарычей при торговой сделке
- SPIFFS_opendir
- SPIFFS_closedir
- SPIFFS_open_by_dirent

SPIFFS_remove

Удалите файл по имени.

s32_t SPIFFS_remove (магарычи при торговой сделке *фс, случайная работа *путь)

SPIFFS_rename

Переименуйте файл.

s32_t SPIFFS_rename (магарычи при торговой сделке *фс, случайная работа *старый, случайная работа *newPath)

SPIFFS_stat

Получите статус файла путем.

```
s32_t
  SPIFFS_st
  at
  (магарычи
  при
  торговой
  сделке
  *фс,
  случайная работа *путь,
  spiffs_stat *spiffsStat)
```

- `фс` – файловая система, держащая файл.
- `путь` – путь к файлу.
- `spiffsStat` – данные о статистике файла.

spiffs_stat содержит:

- obj_id
- размер
- напечатать
- имя:

SPIFFS_unmount

Не установите файловую систему.

недействительный SPIFFS_unmount (магарычи при торговой сделке *фс)

SPIFFS_write

Напишите данные в открытый файл.

```
s32_t
  SPIFFS_writ
  e
  (магарычи
  при
  торговой
  сделке *фс,
  spiffs_file
  filehandle,
```

пустота
*buf, s32_t
len)

Lib-C

Окружающая среда FreeRTOS обеспечивает ряд C установленный порядок библиотеки во время выполнения, который определен в «esp_libc.h».

Страница 408

atoi

интервал atoi (случайная работа константы *s)

атолл

длинный атолл (случайная работа константы *s)

bzero

пустота bzero (пустота *s, size_t n)

calloc

пустота *calloc (size_t c, size_t n)

свободный

свободная пустота (пустота *p)

malloc

пустота *malloc (size_t n)

memcmp

интервал memcmp (пустота константы *m1, пустота константы *m2, size_t n)

memcpy

пустота *memcpy (пустота *dst, пустота константы *src, size_t n)

memmove
пустота *memmove (пустота *dst, пустота константы *src, size_t n)

memset
пустота *memset (пустота *dst, интервал c, size_t n)

os_get_random
интервал os_get_random (неподписанная случайная работа *buf, size_t len)

os_random
неподписанный длинный os_random (пустота)

printf
```
интервал printf (случайная работа константы *формат, ...)
```

Потребность включать «stdio.h».

помещает
интервал помещает (случайная работа константы *ул.)

рэнд
Произведите случайное число.

```
международный рэнд ()
```

Возвратите случайное число. Обратите внимание, что результат - целое число, которое подписано.

realloc
пустота *realloc (пустота *p, size_t n)

snprintf

интервал snprintf (случайная работа *buf, неподписанное международное количество, случайная работа константы *формат, …)

sprintf

интервал sprintf (случайная работа *, случайная работа константы *формат, …)

strcat

случайная работа *strcat (случайная работа *dst, случайная работа константы *src)

strchr

случайная работа *strchr (случайная работа константы *s, интервал c)

strcmp

интервал strcmp (случайная работа константы *s1, случайная работа константы *s2)

strcpy

случайная работа *strcpy (случайная работа *dst, случайная работа константы *src)

strcspn

size_t strcspn (случайная работа константы *s, случайная работа константы *отклоняет),

strdup

случайная работа *strdup (случайная работа константы *s)

strlen

Возвратитесь длина пустого указателя закончила последовательность.

```
size_t strlen (случайная работа константы *s)
```

Возвратитесь длина пустого указателя закончила последовательность.

strncat

случайная работа *strncat (случайная работа *dst, случайная работа константы *src, size_t количество)

strncmp

интервал strncmp (случайная работа константы *s1, случайная работа константы *s2, size_t n)

strncpy

случайная работа *strncpy (случайная работа *dst, случайная работа константы *src, size_t n)

strrchr

случайная работа *strrchr (случайная работа константы *s, интервал c)

strspn

size_t strspn (случайная работа константы *s, случайная работа константы *принимает),

strstr

случайная работа *strstr (случайная работа константы *s1, случайная работа константы *s2)

strtok

случайная работа *strtok (случайная работа *s, случайная работа константы *delim)

strtok_r

случайная работа *strtok_r (случайная работа *s, случайная работа константы *delim, случайная работа ** ptrptr)

стертол

длинный стертол (случайная работа константы *ул., случайная работа ** endptr, международная основа)

zalloc

пустота *zalloc (size_t n)

Структуры данных

esp_spiffs_config

- `phys_size` – Физический размер Вспышки SPI.
- `phys_addr` – Физическое погашение во вспышке SPI используется для магарычей при торговой сделке. Должен быть на блоке граница.
- `phys_erase_block` – Физический размер, стирая блок.
- `log_block_size` – Логический размер блока. Должен соответствовать физическому размеру а блок.
- `log_page_size` – Логический размер страницы.
- `fd_buf_size` – дескрипторный размер области памяти Файла.
- `cache_buf_size` – размер буфера тайника.

station_config

Описание станционной конфигурации. Содержит следующие поля:

- `uint8 ssid[32]` – SSID точки доступа.
- `пароль [64] uint8` – пароль, чтобы получить доступ к точке доступа.
- `uint8 bssid_set` – Флаг, чтобы указать, использовать ли `bssid` собственность. А ценность 1 средства использовать и ценность 0 средств не использовать.

- `uint8 bssid[6]` – Если у нескольких точек доступа есть тот же SSID, BSSID, может
  содержите MAC-адрес, чтобы указать который из точек

доступа, чтобы соединиться с.См. также:

- Станционная конфигурация
- wifi_station_get_config_default
- wifi_station_set_config_current

структура softap_config

Структура контроля за конфигурацией для softAP.

- uint8 ssid[32]
- пароль [64] uint8
- uint8 ssid_len – длина SSID. Если 0, то ssid пустой законченный.
- `канал uint8` – канал, который будет использоваться для коммуникации. Ценности равняются 1 - 13.
- `uint8 authmode` – способ идентификации требуется. Выбор:
  - AUTH_OPEN
  - AUTH_WPA2_PSK
  - AUTH_WPA_PSK
  - AUTH_WPA_WPA2_PSK

AUTH_WEP не поддержан.

- `uint8 ssid_hidden` – Скрыт ли этот SSID. Ценность 1 делает его
  скрытый.
- `uint8 max_connection` – максимальное количество станционных связей.
  максимум и дефолт равняются 4.
- `uint16 beacon_interval` – интервал маяка в миллисекундах. Ценности равняются 100 –
  6

000

0.C

м.

так

же:

- wifi_softap_get_config
- wifi_softap_get_config_default

- wifi_softap_set_config_current

структура station_info

Эта структура предоставляет информацию о станциях, связанных с ESP8266, в то время как это - точка доступа. Это - связанный список со свойствами:

- uint8 bssid[6] –???

- `структура ipaddr IP` – IP-адрес связанной станции,

Чтобы получить следующий вход, мы можем использовать `STAILQ_NEXT (pStationInfo, затем)`.См. также:

- Быть точкой доступа

структура dhcps_lease

Эта структура используется `wifi_softap_dhcps_lease ()` функция, чтобы определить начало и диапазон конца доступных IP-адресов.

Области, содержавшие в:

- структура ip_addr start_ip
- структура ip_addr end_ip Включает:
- user_interface.h

См. также:

- Сервер DHCP

структура bss_info

Эта структура содержит:

- STAILQ_ENTRY (bss_info) затем
- uint8 bssid[6]
- uint8 ssid[32]
- канал uint8
- `sint8 rssi` – полученный признак силы сигнала
- AUTH_MODE authmode
- uint8 is_hidden

- sint16 freq_offset

Чтобы получить следующий вход, мы можем использовать

`STAILQ_NEXT (pBssInfoVar, затем)`. AUTH_MODE - enum

- `AUTH_OPEN` – Никакая идентификация. Никакая проблема на любой станции не соединяется.
- AUTH_WEP = 1
- AUTH_WPA_PSK = 2

- AUTH_WPA2_PSK = 3
- AUTH_WPA_WPA2_PSK =4 Видит также:
- Просмотр для точек доступа

структура ip_info

Эта структура определяет информацию об интерфейсе, находившемся в собственности ESP8266. Это содержит следующие поля:

- `структура ip_addr IP` – IP-адрес интерфейса.
- `структура ip_addr netmask` – netmask используется интерфейсом.
- `структура ip_addr gw` – IP-адрес ворот используется интерфейсом.

См. также:

- структура ip_addr
- IP4_ADDR

структура rst_info

Информация о текущем

ботинке/перезапуске Эта

структура содержит:

- причина uint32
- uint32 exccause
- uint32 epc1
- uint32 epc2
- uint32 epc3
- uint32 excvaddr

- uint32 depc

`Причиной` область является enum со следующими ценностями:

- `0` – перезапуск По умолчанию – Нормальный запуск на власти
- `1` – таймер собаки Часов – контрольная комиссия Аппаратных средств перезагружена
- `2` – Исключение – исключение было обнаружено
- `3` – таймер собаки часов программного обеспечения – контрольная комиссия программного обеспечения перезагружена
- `4` – Мягкий перезапуск

- `5` – Глубокие сны просыпаются

См. также:

- Обработка исключений
- Ошибка: Справочный источник, не найденный

структура espconn

Эта структура данных - представление связи между ESP8266 и партнером. Это содержит «блоки управления» и информацию об идентификации … однако, важно отметить, что это - не всегда непрозрачная часть данных.

- `enum espconn_type тип` – тип может быть одним из
  - `ESPCONN_INVALID`
  - `ESPCONN_TCP` – Определяет эту связь, как являющуюся типа TCP.
  - `ESPCONN_UDP` – Определяет эту связь, как являющуюся типа UDP.
- `enum espconn_state` – государство может быть одним из
  - `ESPCONN_NONE` – государство для в начальной связи.
  - ESPCONN_WAIT
  - ESPCONN_LISTEN
  - ESPCONN_CONNECT
  - ESPCONN_WRITE
  - ESPCONN_READ

- ESPCONN_CLOSE

- `союз {`
  `esp_`
  `tcp`
  `*tcp`
  `esp_`
  `udp`
  `*udp`
  `} первичный` – Эта область - союз `tcp` и `udp` подразумевать, что только один из них должен когда-либо использоваться для случая этой структуры данных. Если структура данных используется для TCP тогда, `tcp` собственность должна использоваться, в то время как для UDP, `udp` собственность должна использоваться.
- `пустота *перемена` – В комментариях, это сигнализируется как область, *зарезервированная* для пользователя кодекс. Возможно, что выбранное имя *(полностью изменяет)*, на самом деле опечатка в заголовочном файле!!
- Другие области … есть другие области в структуре, но они не предназначены, чтобы быть прочитанными или написанными пользовательскими заявлениями. Проигнорируйте их. Используя их ценности не определено и может иметь неожиданные эффекты.

См. также:

- TCP
- esp_tcp
- esp_udp

esp_tcp

- `uint8 local_ip[4]` – местный IP-адрес
- `интервал local_port` – местный порт
- `uint8 remote_ip[4]` – отдаленный IP-адрес
- `интервал remote_port` – отдаленный порт
- Другие области … есть другие области в структуре, но они не предназначены, чтобы быть читайте или написанный пользовательскими заявлениями. Проигнорируйте их. Используя их ценности не определено и может иметь неожиданные эффекты.

См. также:

- структура espconn

esp_udp

Эта структура данных используется в `первичной` собственности `структуры espconn` блок управления.

- `интервал remote_port` – местный IP-адрес
- `интервал local_port` – местный порт
- `uint8 local_ip[4]` – отдаленный IP-адрес
- `uint8 remote_ip[4]` – отдаленный порт

См. также:

- структура espconn
- UDP

структура ip_addr

Представление IP-

адреса. Это содержит

следующее поле:

- `uint32 addr` – фактический 4-байтовый IP-адрес.

Включает:

- ip_addr.h Видят также:
- ipaddr_addr

- IP4_ADDR
- ipaddr_t

ipaddr_t

typedef для `структуры ipaddr`.См. также:

- структура ip_addr

структура ping_option

Области, содержавшие в структуре:

- `количество uint32` – количество раз, чтобы передать звон
- `uint32 ip` – IP-адрес, который является целью звона
- uint32 coarse_time

- recv_function recv_function
- sent_function sent_function
- пустота *перемена;

Включает:

- ping.h Видят также:

- Запрос звона
- ping_start
- ping_regist_recv
- ping_regist_sent

структура ping_resp

Области, содержавшие в структуре:

- uint32 total_count
- uint32 resp_time
- uint32 seqno
- uint32 timeout_count
- байты uint32
- uint32 total_bytes
- uint32 total_time

- `sint8 ping_err` – признак того, произошла ли ошибка. Ценность 0 средс

тва никакая

ошибка.

Включает:

- ping.h Видят также:

- Запрос звона
- ping_start
- ping_regist_recv
- ping_regist_sent

структура mdns_info

- случайная работа *host_name

- случайная работа *server_name
- uint16 server_port
- `неподписанный длинный ipAddr` – Это должно быть предлагаемым IP-адресом.
- `случайная работа *txt_data[10]` – множество вариантов формы «`называет = стоимость`».

См. также:

- Системы доменных имен передачи

enum phy_mode

802,11 физических способа, которые будут использоваться или быть используемым.

- PHY_MODE_11B
- PHY_MODE_11G
- PHY_MODE_11N

GPIO_INT_TYPE

Это возможные спусковые механизмы для перерыва. Это - enum, определенный следующим образом:

- `GPIO_PIN_INTR_DISABLE` – Перерывы отключены.
- `GPIO_PIN_INTR_POSEDGE` – Перерыв на положительном переходе края.
- `GPIO_PIN_INTR_NEGEDGE` – Перерыв на отрицательном переходе края.
- `GPIO_PIN_INTR_ANYEDGE` – Перерыв на любом переходе края.
- `GPIO_PIN_INTR_LOLEVEL` – Перерыв, когда низкий.
- `GPIO_PIN_INTR_HILEVEL` – Перерыв, когда высокий.

Страница 419

См. также:

- gpio_pin_wakeup_enable

System_Event_t

Тип событий содержит:

- `событие uint32` – тип события, которое произошло. Может быть
  - `EVENT_STAMODE_CONNECTED (0)` – Мы успешно соединились с точкой доступа.
    - `uint8[32] event_info.connected.ssid` – SSID точки доступа.
    - uint8 ssid_len
    - uint8[6] bssid
    - `event_info.connected.channel` – канал раньше соединялся с точка доступа.
  - `EVENT_STAMODE_DISCONNECTED (1)`
    - `uint8[6] event_info.disconnected.bssid`
    - `uint8[32] event_info.disconnected.ssid`
    - `uint8 ssid_len`
    - `uint8 event_info.disconnected.reason` – причина - один из следующее:
- 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

- 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`

Это - Союз C, содержащий данные, которые доступны как функция типа событий.

  -

- Event_StaMode_Disconnected_t разъединен
- 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

См. также:

- Ошибка: Справочный источник, не найденный

коды ошибок espconn

Постоянный	Стоимость
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

СТАТУС

Это - enum, определенный следующим образом:

Имя Enum	Стоимость
Хорошо	0
ТЕРПЯТ НЕУДАЧУ	1
ОЖИДАНИЕ	2
ЗАНЯТЫЙ	3
ОТМЕНИТЬ	4

См. также:

Страница 422

- Ошибка: Справочный источник, не найденный

Справочные материалы

Есть богатство информации, доступной на ESP8266 от множества источников.

C ++ программирование

Простое определение класса

Типовой заголовок класса

```
#ifndef
MyClass
_h
#define
MyClass
_h
класс
MyCla
ss
{обще
ствен
ность
:
       MyClass ();
       статическая
       пустота
       myStaticFunc ();
       пустота myFunc
       ();
};
#endif
```

Типовой источник класса

```
#include
<MyClass.h>
MyClass::
MyClass ()
{
       //Кодекс конструктора здесь...
}
Последовательность
       MyClass::
       myStaticFunc ()
```

```
        {//Кодекс
        здесь...
}
недействительн
        ый
        MyClass
        ::
        myFunc
        ()
        {//Коде
        кс
        здесь..
        .
}
```

Функции лямбды

Современный C ++ ввел функции лямбды. Это C ++ языковые функции, которые не должны быть предварительно объявлены, но могут вместо этого быть объявлены «действующими». Функции не имеют никаких имен, связанных с ними, но иначе ведут себя точно так же, как другие функции.

См. также:

- Функции лямбды

Игнорирование предупреждений

Время от времени Ваш кодекс может выпустить предупреждения компиляции, что Вы хотите подавить. Один способ достигнуть этого с помощью C, собирают `#pragma` директиву.

Например:

```
#pragma GCC диагностический проигнорировал «-Wformat»
```


См. также:

- GCC диагностический Pragmas

Затмение

Хотя не технически история ESP8266, я чувствую, что понимание главных компонентов Затмения не причинит вреда.

См. также:

- Затмение ударило документацию

Расстройство ESPFS

ESPFS - библиотека, которая хранит «файлы» во вспышке ESP8266 и позволяет заявлению прочитать их. Это - часть проекта ESPHTTPD.

EspFsInit

```
EspFsInitResult espFsInit (случайная работа *flashAddress)
```

Инициализируйте окружающую среду, указывающую туда, где данные о файле могут быть найдены. Возвращение будет одним из:

- ESPFS_INIT_RESULT_OK
- ESPFS_INIT_RESULT_NO_IMAGE
- ESPFS_INIT_RESULT_BAD_ALIGN

espFsOpen

```
EspFsFile *espFsOpen (случайная работа *имя файла)
```

Откройте файл, определенный именем файла, и возвратите структуру, которая является «ручкой» к файлу или ПУСТОМУ УКАЗАТЕЛЮ, если файл не может быть найден.

espFsClose

```
пустота espFsClose (EspFsFile *fileHandle)
```

Закройте файл, который был ранее открыт требованием к espFsOpen (). Нет далее читает, должен быть выполнен.

espFsFlags

```
интервал espFsFlags (EspFsFile *fileHandle)
```


espFsRead

```
интервал espFsRead (EspFsFile *fileHandle, случайная работа *буфер,
международная длина)
```

Читайте до байтов длины из файла и сохраните их в местоположении памяти, на которое указывает буфер. Фактическое число прочитанных байтов возвращено вызовом функции.

mkespfimage

Это не функция, а команда, которая строит двоичные данные файлов, которые будут помещены во флэш-память.

```
mkespfimage [-c компрессор] [-l compression_level]
```

- -c
  - 0 – Ни один
  - 1 – Heatshrink
- -l

1 ◦

Расстройство ESPHTTPD

Библиотека ESPHTTPD обеспечивает внедрение сервера HTTP, работающего на ESP8266. Чтобы использовать это, мы можем хотеть понять это лучше.

httpdInit

```
пустота httpdInit (HttpdBuiltInUrl *fixedUrls, международный порт)
```

Инициализируйте сервер HTTP, работающий в ESP. Параметр `порта` - номер порта, что ESP послушает на для поступающих запросов браузера. Номер порта по умолчанию, используемый браузерами, равняется 80.

`HttpdBuiltInUrl` - typedef, который предоставляет отображение URL, доступным на сервере HTTP. Области, содержавшие в:

- `случайная работа *URL` – URL, чтобы соответствовать.
- `cgiSendCallback cgiCb` – функция обратного вызова, чтобы звонить, когда был подобран.
- `пустота константы *cgiArg` – Параметры, чтобы пройти в функцию обратного вызова.

Жизненно важно, чтобы последний элемент во множестве имел, АННУЛИРУЕТ для всех признаков. Это служит отчетом завершения. Вот определение в качестве примера для минимального набора построенных в URL:

```
HttpdBuiltInUrl
      builtInUrls []
      = {{ПУСТОЙ
      УКАЗАТЕЛЬ,
      ПУСТОЙ
      УКАЗАТЕЛЬ,
      ПУСТОЙ
      УКАЗАТЕЛЬ}
};
```

`cgiSendCallback` - функция со следующей подписью:

```
интервал (* functionName) (HttpdConnData *connData)
```

Страница 426

Включает:

- httpd.h

httpdGetMimetype

```
случайная работа *httpdGetMimeType (случайная работа *URL)
```

Исследуйте `URL`, переданный в и смотря на его тип файла, определите тип ПАНТОМИМЫ данных. Если никакой тип файла не найден, то тип ПАНТОМИМЫ по умолчанию - «`текст/HTML`».

Включает:

- httpd.h

httpdUrlDecode

```
интервал httpdUrlDecode (случайная работа *val, интервал valLen, случайная
работа *мочит, интервал retLen),
```

Расшифруйте URL согласно правилам расшифровки URL. Закодированный URL поставляется в `val` с длиной `valLen` байтов. Получающаяся расшифрованная последовательность URL будет сохранена в, `мочат` с максимальной длиной `retLen`. Фактическая длина возвращена самим вызовом функции.

Включает:

- httpd.h

httpdStartResponse

```
пустота httpdStartResponse (HttpdConnData *ведут, международный кодекс),
```

Начните посылать данным об ответе вниз соединение по протоколу TCP к браузеру. `Кодовое обозначение` - основной код ответа браузера.

Включает:

- httpd.h

httpdSend

```
интервал httpdSend (HttpdConnData *ведут, случайная работа константы
*данные, интервал len),
```

Пошлите данные в браузер посредством соединения по протоколу TCP. Данными снабжают, поскольку `данные` и `len` параметры - число байтов, чтобы написать. Если `len ==-1`, то данными, как предполагается, является ПУСТОЙ УКАЗАТЕЛЬ, закончил последовательность.

Включает:

- httpd.h

httpdRedirect

```
пустота httpdRedirect (HttpdConnData *ведут, случайная работа *newUrl),
```

Пошлите инструкцию по перенаправлению HTTP в браузер. `newUrl` - URL, который мы хотим, чтобы браузер использовал.

Включает:

- httpd.h

httpdHeader

```
пустота httpdHeader (HttpdConnData *ведут, случайная работа константы
*область, случайная работа константы *val),
```

Пошлите заголовок HTTP. Название заголовка поставляется в `полевом` параметре и его стоимости, поставляемой в `val` параметре.

Включает:

- httpd.h

httpdGetHeader

```
интервал httpdGetHeader (HttpdConnData *ведут, случайная работа *заголовок,
случайная работа *, мочит, интервал retLen),
```

Ищите браузер поставлял заголовок данных, ища заголовок, который соответствует параметру `заголовка`. Если найдено, возвратитесь, стоимость заголовка в буфере, на который указывают, `мочат`, который должен быть, по крайней мере, `retLen` байтами долго.

Включает:

- httpd.h

httpdFindArg

```
интервал httpdFindArg (случайная работа *линия, случайная работа *аргумент,
случайная работа *любитель, интервал buffLen)
```

Учитывая линию текста, ищите параметр формы «name=value» в линии. Если имя соответствует нашему переданному на имя, то возвратите стоимость.

Включает:

- httpd.h

httpdEndHeaders

```
пустота httpdEndHeaders (HttpdConnData *ведут),
```

Завершите продукцию заголовков к

потоку продукции. Включает:

- httpd.h

Makefiles

Книги были написаны на языке и использовании Makefiles, и наша цель не состоит в том, чтобы попытаться переписать те книги. Скорее вот справочник мошенников по началу понять, как прочитать их.

У общего правила в сделать файле есть форма:

```
цель:
	p
	r
	e
	r
	e
	q
	s
	…
	r
	e
	c
	e
	i
	p
	e
```

.
.
.

Переменные определены в форме:

```
name=value
```

Мы можем использовать ценность переменной или с

$ (имя) или с $ {имя}. Другая форма определения:

```
имя: =value
```

Здесь, стоимость заперта к ее стоимости во время определения и не будет рекурсивно расширена.

Некоторые переменные хорошо определили значения:

Переменная	**Значение**
Копия:	C команда компилятора
AR	Archiver командуют
LD	Команда компоновщика
OBJCOPY	Команда копии объекта
OBJDUMP	Команда свалки объекта

Мы можем использовать ценность ранее определенной переменной в других переменных определениях. Например:

```
XTENSA_TOOLS_ROOT? = c:/Espressif/xtensa-lx106-elf/bin
CC           : = $ (XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-gcc
```

определяет компилятор C как абсолютный путь на основе ценности предыдущей переменной. Специальные расширения:

- `$` - название цели
- `$ <`-первый prereq

Комментарии - линии, которые начинаются с «#» характера.

Групповые символы:

- `*` - Все знаки
- ? - Один характер
- `[…]` - Ряд знаков

Сделайте может быть призван, рекурсивно используя

```
сделайте-C <directoryName>
```

Предположите, что мы хотели построить список исходных файлов,

называя справочники, и список исходных файлов тогда становится всеми «.c» файлами в тех справочниках? Как мы можем достигнуть этого?

```
SRC_DIR = dir1 dir2
SRC: = $ (foreach sdir, $ (SRC_DIR), $ ($
группового символа (sdir)/*.c)) OBJ: = $ (patsubst
%.c, $ (BUILD_BASE) / %.o, $ (SRC))
```

Загадка

Вообразите структуру каталогов с

```
a
       a1.c
       a2.c
b
       b1.c
       b2.c
```

цель состоит в том, чтобы собрать их к

```
построить
       a
              a1.o
              a2.o
       b
              b1.o
              b2.o
```

Мы знаем, как собрать x.c → x.o

```
MODULES=
a b
BUILD_BA
SE=build
$ BUILD_DIRS= (addprefix $ (BUILD_BASE)/, $
(МОДУЛИ)) $ SRC= (foreach директор, $
(МОДУЛИ), $ ($ группового символа (директор)/
*.c))
# Замените весь x.c x.o
OBJS= $ (patsubst %.c,
%.o, $ (SRC))
все:
       $ эха (OBJS)
       $ эха ($ группового символа (OBJS)/*.c)
       $ эха (foreach директор, $ (OBJS), $ ($
       группового символа (директор)/*.c))
       повторяют «SRC»: $ (SRC)
тест: $ checkdirs (OBJS)
       повторите «Собранный» $ (SRC)
```

Страница 430

```
.c.o:
       повторите «Компилирование $ (basename $ <)»
```

```
	$ (CC)-c $ <-o строят / $ (addsuffix.o, $ (basename $ <))

checkdirs: $ (BUILD_DIRS)

$
	(
	B
	U
	I
	L
	D
	_
	D
	I
	R
	S
	)
	:
	m
	k
	d
	i
	r
	-
	p
	$
чисто:
	комната-f $ (BUILD_DIRS)
```

У Makefiles также есть интересные команды:

- `$ (раковина <команда раковины>)` - Пробег раковина командует
- `$ (информация «текст»)`, $ (ошибка «текст»), $ (предупреждение «текста») – Производят продукцию от, делают

См. также:

- GNU делает
- Шпаргалка Makefile

Форумы

Есть несколько превосходных мест, чтобы задать вопросы, ответить на другие народные вопросы и читать о вопросах и ответах прошлого.

- Espressif ESP8266 BBS – смягченный форум, которым управляет Espressif. Основной источник для загрузок SDK и источник большой части основных материалов.
- Форум Сообщества ESP8266 – Ряд форумов, посвященных ESP8266, бежит за и пользовательским

сообществом ESP8266.

- Форум Сообщества ESP32 – общественный форум ESP32, где **все** обсуждения ESP32 происходят.

Справочные документы

Espressif распределяет PDF и электронные таблицы Excel, содержащие основную информацию о ESP8266. Они могут быть загружены свободно с сети.

- ESP8266 Техническая Ссылка – v1.2
- Часто задаваемые вопросы ESP8266
- ESP8266 SDK, Начинающий Руководство – v2.3
- Ссылка API ESP8266 Non-OS SDK – v2.0
- Спецификация ESP-WROOM-32 – 2016-08-22
- ESP32 техническое справочное руководство – 2016-09-23
- ESP32_RTOS_SDK

Старые документы

- Спецификация v4.3 0A-ESP8266
- Руководство пользователя Аппаратных средств 0B-ESP8266 v1.1
- Спецификация v0.3 0C-ESP8266 WROOM WiFi Модуля
- Выпуск 2014-11-15 списка булавки 0D-ESP8266

- Руководство пользователя V1.4 2A-ESP8266 IOT SDK
- Образец 2B-ESP8266 SDK IOT V1.3
- Руководство по программированию 2C-ESP8266 SDK V1.5
- 4A-ESP8266 В наборе команд V1.5
- 4B-ESP8266 В примерах команды V1.3
- 4C-ESP8266 В примере модернизации
- GPIO v0.5 Интерфейса 8A-ESP8266
- 8B-ESP8266 соединяют выпуск 2014-11-15 регистров GPIO
- I2C v1.0 Интерфейса 8C-ESP8266
- PWM v1.1 Интерфейса 8D-ESP8266
- UART v0.2 Интерфейса 8E-ESP8266
- Интерфейс 8F-ESP8266 Регистры UART v0.1
- Интерфейс 8G-ESP8266 Инфракрасное Дистанционное управление v0.3
- 8-й ESP8266 Интерфейс ода SDIO SPI M v0.1-2
- Передача SPI-Wi-Fi Интерфейса 8I-ESP8266 1 – прерывает метод v0.1
- Передача SPI-Wi-Fi Интерфейса 8J-ESP8266 2 – прерывает метод v1.0
- Введение Наркомана 8K-ESP8266 v0.3
- 8L-ESP8266 соединяют выпуск 2014-11-18 регистров SPI
- Таймер интерфейса 8M-ESP8266 регистрирует выпуск 2014-11-18
- Ссылка 8N-ESP8266 SPI v1.0
- 8O-ESP8266 SPI Overlap & Display Application Guide v0.1
- Описание v1.0 8P-ESP8266 I2S Модуля
- Мультиустройство API v1.0 Хозяина 8Q-ESP8266 HSPI
- Введение таймера 9A-ESP8266 FRC (еще изданный)
- Описание v1.0 Функции режима ожидания 9B-ESP8266
- Руководство по программированию 20A-ESP8266 RTOS SDK V1. 3.0
- Ссылка 20B-ESP8266 RTOS SDK API V1.3.0
- Руководство пользователя сетки 30A-ESP8266 V1.0
- Вспышка 99A-ESP8266 Операция RW v0.2
- Таймер 99B-ESP8266 (еще изданный)
- 99C-ESP8266 Модернизация ОТЫ v1.6

Вот подобные справочные документы для ESP32

- Ссылка API ESP32 RTOS SDK v1.1.0

GitHub

Есть много общедоступных проектов, разработанных сверху и вокруг ESP8266, который может быть найден на GitHub. Вот список связей с некоторыми из этих проектов, которые очень хорошо стоят взглянуть:

- EspressifApp
- jantje/arduion-eclipse-plugin
- eriksl/esp8266-universal-io-bridge
- CHERTS/esp8266-devkit
- Проект ESPHTTPD
  - Spritetm/esphttpd
  - Spritetm/libesphttpd

GitHub быстрые обманы

Работая с общедоступными проектами, есть времена, когда мы хотели бы выполнить некоторые задачи, которые включают несколько команд. Здесь мы пытаемся захватить некоторые более интересные, которые время от времени используются в проектах ESP8266.

Страница 432

```
мерзавец отдаленный-v
отдаленный мерзавец
добавляет по
разведке и добыче
нефти и газа <URL>
усилие мерзавца
вверх по течению
разведка и добыча нефти и газа/владелец слияния мерзавца
```

См. также:

- Простой справочник по подцепляет на вилку GitHub и Мерзавца

SDK

Software Development Kit (SDK) издан Espressif и обязан создавать базирующиеся приложения C. Это содержит жизненную документацию в форме PDF, которые, кажется, не доступны в другом месте.

- ESP8266 SDK v1.4.0

esp-

open-

sdk
cygwin
бизон
пакето
в
с
о
г
н
и
т
e
t
e
x
i
n
f
o
w
g
e
t
,
и
с
п
р
а
в
л
я

ю
т
l
i
b
t
o
o
l
,
а
в
т
о
д
е
л
а
ю
т
gettext-devl

Выполните клона мерзавца - рекурсивный <Repo> из раковины.

Сравнения одноплатного компьютера

На рынке есть много одноплатных компьютеров. Хотя ESP8266 обычно не считают одним из них, много людей использует его как таковой. Давайте поднимем стол и противопоставим ESP8266 против этих компьютеров:

Устройство	Центральный процессор	RAM	Вспышка	Wi-Fi	GPIO	OS	Стоимость
ESP8266	80 МГц	80K	512K	Y	9	FreeRTOS	4$
ESP32	160 МГц	512K	Вар	Y	?	FreeRTOS	??
Ардуино	20 МГц	2K	32K	N	?	N/A	2$
Ноль пи	1 ГГц	512 МБ	SD	N	?	Linux	5$
Омега	400 МГц	64 МБ	16 МБ	Y	18	Linux	19$
Омега 2							
C.H.I.P.	1 ГГц	512 МБ	4 ГБ	Y	8 +	Linux	9$

Герои

В пользовательском сообществе ESP8266 есть люди, которых я рассматриваю, чтобы раздвинуть границы знания далее или разработал инструменты, которые существенно улучшают работу с устройствами. Я хочу занять некоторое время и вызвать этих хороших людей, без которых все наши путешествия ESP8266 были бы более трудными:

Макс Филиппов - jcmvbkbc - компилятор GCC для Xtensa

- Веб-сайт: GitHub –https://github.com/jcmvbkbc

Компилятор для C на основе GCC, который собирает к набору из двух предметов Xtensa для высвечивания. Сомнительно, что любая полезная работа могла быть выполнена без этого вклада.

Иван Грохотков - igrr - Ардуино IDE для развития ESP8266

- Веб-сайт: GitHub –esp8266/Arduino

Внедрение технологии, которая позволяет разрабатывать приложения ESP8266, используя IDE Ардуино, а также библиотеки, которые наносят на карту функции Ардуино к эквивалентам ESP8266 или около эквивалентов.

jantje - Затмение Ардуино

- Веб-сайт:Затмение Ардуино
- Веб-сайт: GitHub –jantje/arduino-eclipse-plugin

Хотя не технически для просто ESP8266, проект Затмения Ардуино важен. Это обеспечивает способность создать приложения Ардуино, используя Затмение, у которого есть превосходящая среда разработки для более опытных программистов. Объедините это с проектом Arduino ESP8266, и у нас есть фантастическая окружающая среда в нашем распоряжении.

Ричард Слоан - Владелец Сообщества ESP8266

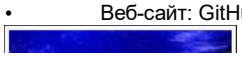
- Веб-сайт:http://www.esp8266.com/index.php

Несомненно любой, кто касается ESP8266, должен посетить этот общественный веб-сайт. Форум нашел, что там чрезвычайно богато

знанием и в большой степени торговавший. Одинаково все встречены люди, плохо знакомые с ESP8266 и экспертами. Просто приезжайте и присоединитесь к забаве. Ричард - также владелецthemindfactory.com.

Михаил Григорев - ЧЕРТЫ - Затмение для развития ESP8266

- Веб-сайт:Проект Неофициальное Средство разработки для Espressif ESP8266
- Веб-сайт: GitHub –CHERTS/esp8266-devkit



Необычно хорошо полированный набор артефактов и инструкций для строительства заявлений ESP8266 C в рамках среды разработки Затмения.

Mmiscool - Основной переводчик

- Веб-сайт:http://www.esp8266basic.com
- Веб-сайт:форумы

Основной переводчик/окружающая среда для написания применений в Основном языке программирования. Автор посвящает много времени и энергии в длительное развитие и предоставляет очень быстрый ответ на вопросы пользователя.

Области к исследованию

- Таймеры аппаратных средств …, когда они становятся названными?
- Если я определяю функции в библиотеке, названной libcommon.a,

что добавлено к собранному применению, когда я связываюсь с этой библиотекой? Это - все в библиотеке или просто файлах объекта, на которые ссылаются?

- Какова карта/расположение памяти ESP8266?
- Сколько RAM установлено и доступно для использования?
- Зарегистрируйте информацию, содержавшую здесь …http://bbs.espressif.com/viewtopic.php?p=3066#p3066
- Что такое SSDP и как делает связанный с библиотеками SSDP?
- Улей устройства исследования -http://devicehive.com/
- Документ используя Визуальный Микро отладчик с Визуальной Студией.http://www.visualmicro.com/
- Управление электропитанием
- Поддержка MQTT
- Исследуйте семантику wifi_station_connect (), когда мы будем уже связаны.
- Ведите Ардуино как раба ESP8266.