

```
1
2Copyright 2010-2015 Amazon.com, In
3Additions Copyright 2016 Espressif
4
5Licensed under the Apache License,
6You may not use this file except i
7A copy of the License is located a
8
9 http://aws.amazon.com/apache2.0
10
11or in the "license" file accompany
12on an "AS IS" BASIS, WITHOUT WARRA
13express or implied. See the Licens
14permissions and limitations under
15
16
17@file subscribe_publish_sample.c
18@brief simple MQTT publish and sub
19
20This example takes the parameters
21It subscribes and publishes to the
22
23Some setup is required. See exampl
24
25
26#include <stdio.h>
27#include <stdlib.h>
28#include <ctype.h>
29#include <unistd.h>
30#include <limits.h>
31#include <string.h>
32#include <time.h>
33#include <sys/time.h>
34#include "freertos/FreeRTOS.h"
35#include "freertos/task.h"
36#include "freertos/event_groups.h"
37#include "freertos/queue.h"
38#include "math.h"
39#include "esp_system.h"
40#include "esp_wifi.h"
41#include "esp_event_loop.h"
42#include "esp_log.h"
43#include "esp_vfs_fat.h"
44#include "driver/sdmmc_host.h"
45#include "driver/timer.h"
46#include "driver/gpio.h"
47#include "driver/adc.h"
48#include "esp_adc_cal.h"
49#include "esp_sntp.h"
50#include "nvs.h"
```

```
1
2Copyright 2010-2015 Amazon.com, In
3Additions Copyright 2016 Espressif
4
5Licensed under the Apache License,
6You may not use this file except i
7A copy of the License is located a
8
9 http://aws.amazon.com/apache2.0
10
11or in the "license" file accompany
12on an "AS IS" BASIS, WITHOUT WARRA
13express or implied. See the Licens
14permissions and limitations under
15
16
17@file subscribe_publish_sample.c
18@brief simple MQTT publish and sub
19
20This example takes the parameters
21It subscribes and publishes to the
22
23Some setup is required. See exampl
24
25
26#include <stdio.h>
27#include <stdlib.h>
28#include <ctype.h>
29#include <unistd.h>
30#include <limits.h>
31#include <string.h>
32#include <time.h>
33#include <sys/time.h>
34#include "freertos/FreeRTOS.h"
35#include "freertos/task.h"
36#include "freertos/event_groups.h"
37#include "freertos/queue.h"
38#include "math.h"
39#include "esp_system.h"
40#include "esp_wifi.h"
41#include "esp_event_loop.h"
42#include "esp_log.h"
43#include "esp_vfs_fat.h"
44#include "driver/sdmmc_host.h"
45#include "driver/timer.h"
46#include "driver/gpio.h"
47#include "driver/adc.h"
48#include "esp_adc_cal.h"
49#include "esp_sntp.h"
50#include "nvs.h"
```

```

51#include "nvs_flash.h"
52
53#include <time.h>
54#include "esp_sntp.h"
55
56#include "aws_iot_config.h"
57#include "aws_iot_log.h"
58#include "aws_iot_version.h"
59#include "aws_iot_mqtt_client_interfa
60#include "cJSON.h"
61
62#include "demo_config.h"
63
64def CONFIG_IDF_TARGET_ESP32
65fine CHIP_NAME "ESP32"
66dif
67
68def CONFIG_IDF_TARGET_ESP32S2BETA
69fine CHIP_NAME "ESP32-S2 Beta"
70dif
71
7272122 ycc added the following bloc
73#include <wifi_provisioning/manager.h
74#include <wifi_provisioning/scheme_bl
75#include "qrcode.h"
76
77#include "app_priv.h" //for app_dri
78#include "board_esp32_devkitc.h"
79cc 3-13-22 added include files
80OpenSSL sockets transport implemen
81#include "tls_freertos.h"
82
83Clock for timer. */
84#include "clock.h"
85
86pthread include. */
87#include <pthread.h>
88#include "semaphore.h"
89#include <unistd.h>
90#include "freertos/FreeRTOS.h"
91#include "freertos/task.h"
92#include "esp_pthread.h"
93
94MQTT include. */
95#include "core_mqtt.h"
96#include "mqtt_subscription_manager.h
97
98nclude backoff algorithm header fo
99#include "backoff_algorithm.h"
100

```

```

51#include "nvs_flash.h"
52
53#include <time.h>
54#include "esp_sntp.h"
55
56#include "aws_iot_config.h"
57#include "aws_iot_log.h"
58#include "aws_iot_version.h"
59#include "aws_iot_mqtt_client_interfa
60#include "cJSON.h"
61
62#include "demo_config.h"
63
64def CONFIG_IDF_TARGET_ESP32
65fine CHIP_NAME "ESP32"
66dif
67
68def CONFIG_IDF_TARGET_ESP32S2BETA
69fine CHIP_NAME "ESP32-S2 Beta"
70dif
71
7272122 ycc added the following bloc
73#include <wifi_provisioning/manager.h
74#include <wifi_provisioning/scheme_bl
75#include "qrcode.h"
76
77#include "app_priv.h" //for app_dri
78#include "board_esp32_devkitc.h"
79cc 3-13-22 added include files
80OpenSSL sockets transport implemen
81#include "tls_freertos.h"
82
83Clock for timer. */
84#include "clock.h"
85
86pthread include. */
87#include <pthread.h>
88#include "semaphore.h"
89#include <unistd.h>
90#include "freertos/FreeRTOS.h"
91#include "freertos/task.h"
92#include "esp_pthread.h"
93
94MQTT include. */
95#include "core_mqtt.h"
96#include "mqtt_subscription_manager.h
97
98nclude backoff algorithm header fo
99#include "backoff_algorithm.h"
100

```

```

101OTA Library include. */
102clude "ota.h"
103clude "ota_config.h"
104
105OTA Library Interface include. */
106clude "ota_os_freertos.h"
107clude "ota_mqtt_interface.h"
108clude "ota_pal.h"
109
110deep sleep */
111clude "soc/soc_caps.h"
112clude "esp_sleep.h"
113clude "driver/adc.h"
114clude "driver/rtc_io.h"
115clude "soc/rtc.h"
116clude "esp32/ulp.h"
117clude "soc/sens_periph.h"
118clude "driver/touch_pad.h"
119
120
121
122fine DEFAULT_WAKEUP_LEVEL    ESP_G
123Include firmware version struct de
124clude "ota_appversion32.h"
125
126clude "demo_header.h"
127cc 3-13-22 end
128tic const char *TAG = "subpub";
129tic void initialize_sntp(void);
130The examples use simple WiFi confi
131'make menuconfig'.
132
133If you'd rather not, just change t
134the config you want - ie #define E
135
136fine CONFIG_ESP_MAXIMUM_RETRY    5
137fine EXAMPLE_WIFI_SSID CONFIG_WIFI
138fine EXAMPLE_WIFI_PASS CONFIG_WIFI
139fine EXAMPLE_ESP_MAXIMUM_RETRY    CO
140
141The event group allows multiple bi
142- we are connected to the AP with
143- we failed to connect after the m
144fine WIFI_CONNECTED_BIT BIT0
145fine WIFI_FAIL_BIT    BIT1
146
147
148fine TIMER_DIVIDER    (80) /
149fine TIMER_SCALE    (TIMER_
150fine DEFAULT_VREF    1100 /

```

```

101OTA Library include. */
102clude "ota.h"
103clude "ota_config.h"
104
105OTA Library Interface include. */
106clude "ota_os_freertos.h"
107clude "ota_mqtt_interface.h"
108clude "ota_pal.h"
109
110Include firmware version struct de
111clude "ota_appversion32.h"
112
113clude "demo_header.h"
114cc 3-13-22 end
115tic const char *TAG = "subpub";
116tic void initialize_sntp(void);
117The examples use simple WiFi confi
118'make menuconfig'.
119
120If you'd rather not, just change t
121the config you want - ie #define E
122
123fine CONFIG_ESP_MAXIMUM_RETRY    5
124fine EXAMPLE_WIFI_SSID CONFIG_WIFI
125fine EXAMPLE_WIFI_PASS CONFIG_WIFI
126fine EXAMPLE_ESP_MAXIMUM_RETRY    CO
127
128The event group allows multiple bi
129- we are connected to the AP with
130- we failed to connect after the m
131fine WIFI_CONNECTED_BIT BIT0
132fine WIFI_FAIL_BIT    BIT1
133
134
135fine TIMER_DIVIDER    (80) /
136fine TIMER_SCALE    (TIMER_
137fine DEFAULT_VREF    1100 /

```

```

151 fine NO_OF_SAMPLES    3           // 138 fine NO_OF_SAMPLES    3           //
152                               139
153                               140
154 GPIO definition, in addtion GPIO 0 141 GPIO definition, in addtion GPIO 0
155 fine LOPlus    34
156 fine LOMinus   35
157 fine SDN       23
158 fine FR        21
159 fine DC        22
160
161 fine LED_RED    GPIO_NUM_16
162 fine LED_GREEN  GPIO_NUM_17
163 fine LED_BLUE   GPIO_NUM_18
164
165 fine DEEPSLEEP  GPIO_NUM_32
166
167
168 fine ECG_IDLE 0           // Id 153 fine ECG_IDLE 0           // Id
169 fine ECG_ACQUIRING 1       // EC 154 fine ECG_ACQUIRING 1       // EC
170 fine ECG_RECORDING 2       // EC 155 fine ECG_RECORDING 2       // EC
171 fine ECG_SENDING_MQTT 3     // EC 156 fine ECG_SENDING_MQTT 3     // EC
172 fine ECG_FINISH 4         // da 157 fine ECG_FINISH 4         // da
173 fine ECG_ERROR_WIFI 5      // er 158 fine ECG_ERROR_WIFI 5      // er
174 fine ECG_ERROR_MQTT 6      // er 159 fine ECG_ERROR_MQTT 6      // er
175 fine ECG_OTA_UPDATE 7      // EC 160 fine ECG_OTA_UPDATE 7      // EC
176 fine ECG_SSID_RESET 8
177 fine ECG_ACQCOUNT 3000    // co 161 fine ECG_SSID_RESET 8
178 fine ECG_RECCOUNT 9000     // cou 162 fine ECG_ACQCOUNT 3000    // co
179 fine ECG_MQTTCOUNT 2000   // c 163 fine ECG_RECCOUNT 9000     // cou
180                               164 fine ECG_MQTTCOUNT 2000   // c
181                               165
181 fine JOB_CHECK_STATE_NOT_CHECKED 166 fine JOB_CHECK_STATE_NOT_CHECKED
182 fine JOB_CHECK_STATE_CHECKED_NO_UP 167 fine JOB_CHECK_STATE_CHECKED_NO_UP
183
184 fine MQTT_PROCESS_LOOP_TIMEOUT_MS 168
185                               169 fine MQTT_PROCESS_LOOP_TIMEOUT_MS
186 deep sleep defines */
187 fine DEFAULT_WAKEUP_LEVEL    ESP_G 170
188 fine TOUCH_THRESH_NO_USE 0
189
190 #define struct {
191   int timer_group;
192   int timer_idx;
193   int alarm_interval;
194   bool auto_reload;
195   xample_timer_info_t;
196
197
198 @brief A sample structure to pass
199
200
171 #define struct {
172   int timer_group;
173   int timer_idx;
174   int alarm_interval;
175   bool auto_reload;
176   xample_timer_info_t;
177
178
179 @brief A sample structure to pass
180
181

```

```

201 edef struct {
202   example_timer_info_t info;
203   uint64_t timer_counter_value;
204   xample_timer_event_t;
205
206   tic xQueueHandle s_timer_queue;
207
208   tic esp_adc_cal_characteristics_t
209
210   tic const adc_channel_t channel =
211   tic const adc_bits_width_t width =
212
213   tic const adc_atten_t atten = ADC_
214   tic const adc_unit_t unit = ADC_UN
215
216   tic short jobCheckState;
217
218   igned char blinkingPattern[9][10]
219       {0,0,0,0,
220       {0,1,0,1,
221       {1,1,1,1,
222       {0,0,0,0,
223       {1,0,1,0,
224       {0,0,1,1,
225       {1,1,0,0,
226       {1,0,1,0,
227       {1,0,1,0,
228
229
230   FreeRTOS event group to signal whe
231   tic EventGroupHandle_t wifi_event_
232   rt ecgState;
233   r macAddress[13];
234   rt nvsProvisionStatus = false;
235   r *private_key = NULL;
236   e_t private_key_len = 0;
237   r *certificate_pem = NULL;
238   e_t certificate_pem_len = 0;
239   ern int     aws_iot_demo_main();
240
241   The event group allows multiple bi
242   but we only care about one event -
243   to the AP with an IP? */
244   st int CONNECTED_BIT = BIT0;
245   st int WIFI_CONNECTED_EVENT = BIT0
246
247   72221 ycc added the following bloc
248   fine PROV_QR_VERSION      "v1"
249   fine PROV_TRANSPORT_SOFTAP  "soft
250   fine PROV_TRANSPORT_BLE     "ble"

```

```

182 edef struct {
183   example_timer_info_t info;
184   uint64_t timer_counter_value;
185   xample_timer_event_t;
186
187   tic xQueueHandle s_timer_queue;
188
189   tic esp_adc_cal_characteristics_t
190
191   tic const adc_channel_t channel =
192   tic const adc_bits_width_t width =
193
194   tic const adc_atten_t atten = ADC_
195   tic const adc_unit_t unit = ADC_UN
196
197   tic short jobCheckState;
198
199   igned char blinkingPattern[9][10]
200       {0,0,0,0,
201       {0,1,0,1,
202       {1,1,1,1,
203       {0,0,0,0,
204       {1,0,1,0,
205       {0,0,1,1,
206       {1,1,0,0,
207       {1,0,1,0,
208       {1,0,1,0,
209
210
211   FreeRTOS event group to signal whe
212   tic EventGroupHandle_t wifi_event_
213   rt ecgState;
214   r macAddress[13];
215   rt nvsProvisionStatus = false;
216   r *private_key = NULL;
217   e_t private_key_len = 0;
218   r *certificate_pem = NULL;
219   e_t certificate_pem_len = 0;
220   ern int     aws_iot_demo_main();
221
222   The event group allows multiple bi
223   but we only care about one event -
224   to the AP with an IP? */
225   st int CONNECTED_BIT = BIT0;
226   st int WIFI_CONNECTED_EVENT = BIT0
227
228   72221 ycc added the following bloc
229   fine PROV_QR_VERSION      "v1"
230   fine PROV_TRANSPORT_SOFTAP  "soft
231   fine PROV_TRANSPORT_BLE     "ble"

```

```

251 fine QRCODE_BASE_URL          "http
252
253 CA Root certificate, device ("Thin
254 ("Thing") key.
255
256 Example can be configured one of t
257
258 "Embedded Certs" are loaded from f
259
260 "Filesystem Certs" are loaded from
261
262 See example README for more detail
263
264 #defined(CONFIG_EXAMPLE_EMBEDDED_C
265
266 #ern const uint8_t aws_root_ca_pem_
267 #ern const uint8_t aws_root_ca_pem_
268 #ern const uint8_t certificate_pem_
269 #ern const uint8_t certificate_pem_
270 #ern const uint8_t private_pem_key_
271 #ern const uint8_t private_pem_key_
272
273 if defined(CONFIG_EXAMPLE_FILESYST
274
275 #tic const char * DEVICE_CERTIFICAT
276 #tic const char * DEVICE_PRIVATE_KE
277 #tic const char * ROOT_CA_PATH = CO
278
279 se
280 #ror "Invalid method for loading ce
281 #dif
282
283 fine MQTT_RECV_POLLING_TIMEOUT_MS
284
285
286 @brief Default MQTT HOST URL is pu
287
288 #r HostAddress[255] = AWS_IOT_MQTT_
289
290
291 @brief Default MQTT port is pulled
292
293 #t32_t port = AWS_IOT_MQTT_PORT;
294
295 cc 3-13-22 added externs
296
297 #edef xSemaphoreHandle osi_sem_t;
298
299
300 @brief Network connection context

```

```

232 fine QRCODE_BASE_URL          "http
233
234 CA Root certificate, device ("Thin
235 ("Thing") key.
236
237 Example can be configured one of t
238
239 "Embedded Certs" are loaded from f
240
241 "Filesystem Certs" are loaded from
242
243 See example README for more detail
244
245 #defined(CONFIG_EXAMPLE_EMBEDDED_C
246
247 #ern const uint8_t aws_root_ca_pem_
248 #ern const uint8_t aws_root_ca_pem_
249 #ern const uint8_t certificate_pem_
250 #ern const uint8_t certificate_pem_
251 #ern const uint8_t private_pem_key_
252 #ern const uint8_t private_pem_key_
253
254 if defined(CONFIG_EXAMPLE_FILESYST
255
256 #tic const char * DEVICE_CERTIFICAT
257 #tic const char * DEVICE_PRIVATE_KE
258 #tic const char * ROOT_CA_PATH = CO
259
260 se
261 #ror "Invalid method for loading ce
262 #dif
263
264 fine MQTT_RECV_POLLING_TIMEOUT_MS
265
266
267 @brief Default MQTT HOST URL is pu
268
269 #r HostAddress[255] = AWS_IOT_MQTT_
270
271
272 @brief Default MQTT port is pulled
273
274 #t32_t port = AWS_IOT_MQTT_PORT;
275
276 cc 3-13-22 added externs
277
278 #edef xSemaphoreHandle osi_sem_t;
279
280
281 @brief Network connection context

```

```

301
302ern NetworkContext_t networkContext
303
304
305@brief MQTT connection context use
306
307ern MQTTContext_t mqttContext;
308
309ern OtaAppBuffer_t otaBuffer;
310
311ern char *registrationBuff;
312
313
314@brief Keep a flag for indicating
315
3161 mqttSessionEstablished;
317
318
319@brief Mutex for synchronizing cor
320
321read_mutex_t mqttMutex;
322
323
324@brief Semaphore for synchronizing
325
326_sem_t bufferSemaphore;
327
328cc 071122 add this to be removed 1
329_handle_t fleet_prov_handle;
330ern int osi_sem_new(osi_sem_t *sem
331
332ern int osi_sem_free(osi_sem_t *se
333
334ern int osi_sem_take(osi_sem_t *se
335
336ern void osi_sem_give(osi_sem_t *s
337
338ern int initializeMqtt( MQTTContext
339
340ern int startOTADemo( void );
341
342ern void disconnect( void );
343
344ern int establishConnection(void);
345
346ern int mqttPublish();
347
348ern int mqttSubscribe();
349
350ern int mqttPublishNoMutex();

```

```

282
283ern NetworkContext_t networkContext
284
285
286@brief MQTT connection context use
287
288ern MQTTContext_t mqttContext;
289
290ern OtaAppBuffer_t otaBuffer;
291
292ern char *registrationBuff;
293
294
295@brief Keep a flag for indicating
296
2971 mqttSessionEstablished;
298
299
300@brief Mutex for synchronizing cor
301
302read_mutex_t mqttMutex;
303
304
305@brief Semaphore for synchronizing
306
307_sem_t bufferSemaphore;
308
309cc 071122 add this to be removed 1
310_handle_t fleet_prov_handle;
311ern int osi_sem_new(osi_sem_t *sem
312
313ern int osi_sem_free(osi_sem_t *se
314
315ern int osi_sem_take(osi_sem_t *se
316
317ern void osi_sem_give(osi_sem_t *s
318
319ern int initializeMqtt( MQTTContext
320
321ern int startOTADemo( void );
322
323ern void disconnect( void );
324
325ern int establishConnection(void);
326
327ern int mqttPublish();
328
329ern int mqttSubscribe();
330
331ern int mqttPublishNoMutex();

```



```

351
352 ern void setOtaInterfaces();
353
354 ern void otaAppCallback();
355
356 ern void * otaThread();
357
358 ern void provisionEventCallback();
359 cc 3-13-22 end
360 leep sleep
361 tic void calibrate_touch_pad(touch
362
363 at w0=0.0, w1=0.0, w2=0.0, w3=0.0,
364 at bpfX, bpfX1;
365 igned short aData;
366 peakCounter=0;
367 detectedBeat = 0;
368 at bpfS0=0, bpfS1=0, bpfS2=0, bpfS
369 at mVariance=0, mAvg=100, m0=100,
370 at pVariance=300000, pAvg=0, p0=10
371 at variance=650;
372 at nAvg=0, n0=0, n1=0, n2=0, n3=0,
373 hb0=0, hb1=0;
374 at heartRate=0;
375 at noiseFloor = 0;
376
377
378 igned short *dataBuffer;
379
380 d time_sync_notification_cb(struct
381
382 ESP_LOGI(TAG, "Notification of a
383
384
385 tic void initialize_sntp(void)
386
387 ESP_LOGI(TAG, "Initializing SNTP"
388 sntp_setoperatingmode(SNTP_OPMODE
389 sntp_setservername(0, "pool.ntp.o
390 sntp_set_time_sync_notification_c
391 def CONFIG_SNTP_TIME_SYNC_METHOD_S
392 sntp_set_sync_mode(SNTP_SYNC_MODE
393 dif
394 sntp_init();
395
396
397
398 tic void calibrate_touch_pad(touch
399
400 int avg = 0;

```

```

332
333 ern void setOtaInterfaces();
334
335 ern void otaAppCallback();
336
337 ern void * otaThread();
338
339 ern void provisionEventCallback();
340 cc 3-13-22 end
341
342 at w0=0.0, w1=0.0, w2=0.0, w3=0.0,
343 at bpfX, bpfX1;
344 igned short aData;
345 peakCounter=0;
346 detectedBeat = 0;
347 at bpfS0=0, bpfS1=0, bpfS2=0, bpfS
348 at mVariance=0, mAvg=100, m0=100,
349 at pVariance=300000, pAvg=0, p0=10
350 at variance=650;
351 at nAvg=0, n0=0, n1=0, n2=0, n3=0,
352 hb0=0, hb1=0;
353 at heartRate=0;
354 at noiseFloor = 0;
355
356
357 igned short *dataBuffer;
358
359 d time_sync_notification_cb(struct
360
361 ESP_LOGI(TAG, "Notification of a
362
363
364 tic void initialize_sntp(void)
365
366 ESP_LOGI(TAG, "Initializing SNTP"
367 sntp_setoperatingmode(SNTP_OPMODE
368 sntp_setservername(0, "pool.ntp.o
369 sntp_set_time_sync_notification_c
370 def CONFIG_SNTP_TIME_SYNC_METHOD_S
371 sntp_set_sync_mode(SNTP_SYNC_MODE
372 dif
373 sntp_init();
374
375

```



```

401 const size_t calibration_count =
402 for (int i = 0; i < calibration_c
403     uint16_t val;
404     touch_pad_read(pad, &val);
405     avg += val;
406 }
407 avg /= calibration_count;
408 const int min_reading = 300;
409 if (avg < min_reading) {
410     printf("Touch pad #%d average
411           "Not using for deep sl
412     touch_pad_config(pad, 0);
413 } else {
414     int threshold = avg - 100;
415     printf("Touch pad #%d average
416     touch_pad_config(pad, thresho
417 }

```

```

420 tic void check_efuse(void)
421
422 //Check if TP is burned into eFus
423 if (esp_adc_cal_check_efuse(ESP_A
424     //printf("eFuse Two Point: Su
425 } else {
426     //printf("eFuse Two Point: NO
427 }
428 //Check Vref is burned into eFuse
429 if (esp_adc_cal_check_efuse(ESP_A
430     //printf("eFuse Vref: Support
431 } else {
432     //printf("eFuse Vref: NOT sup
433 }
434
435
436 tic void print_char_val_type(esp_a
437
438 if (val_type == ESP_ADC_CAL_VAL_E
439     printf("Characterized using T
440 } else if (val_type == ESP_ADC_CA
441     printf("Characterized using e
442 } else {
443     printf("Characterized using D
444 }
445 */
446
447
448 tic bool IRAM_ATTR timer_group_isr
449
450 BaseType_t high_task_awoken = pdF

```

```

376 tic void check_efuse(void)
377
378 //Check if TP is burned into eFus
379 if (esp_adc_cal_check_efuse(ESP_A
380     //printf("eFuse Two Point: Su
381 } else {
382     //printf("eFuse Two Point: NO
383 }
384 //Check Vref is burned into eFuse
385 if (esp_adc_cal_check_efuse(ESP_A
386     //printf("eFuse Vref: Support
387 } else {
388     //printf("eFuse Vref: NOT sup
389 }
390
391
392 tic void print_char_val_type(esp_a
393
394 if (val_type == ESP_ADC_CAL_VAL_E
395     printf("Characterized using T
396 } else if (val_type == ESP_ADC_CA
397     printf("Characterized using e
398 } else {
399     printf("Characterized using D
400 }
401 */
402
403
404 tic bool IRAM_ATTR timer_group_isr
405
406 BaseType_t high_task_awoken = pdF

```

```

451 example_timer_info_t *info = (exa
452
453 uint64_t timer_counter_value = ti
454
455 /* Prepare basic event data that
456
457 example_timer_event_t evt = {
458     .info.timer_group = info->tim
459     .info.timer_idx = info->timer
460     .info.auto_reload = info->aut
461     .info.alarm_interval = info->
462     .timer_counter_value = timer_
463 };
464
465
466 if (!info->auto_reload) {
467     timer_counter_value += info->
468     timer_group_set_alarm_value_i
469 }
470
471 /* Now just send the event data b
472 xQueueSendFromISR(s_timer_queue,
473
474 return high_task_awoken == pdTRUE
475
476
477
478 @brief Initialize selected timer o
479
480 @param group Timer Group number, i
481 @param timer timer ID, index from
482 @param auto_reload whether auto-re
483 @param timer_interval_sec interval
484
485 tic void ad_tg_timer_init(int grou
486
487 /* Select and initialize basic pa
488 timer_config_t config = {
489     .divider = TIMER_DIVIDER,
490     .counter_dir = TIMER_COUNT_UP
491     .counter_en = TIMER_PAUSE,
492     .alarm_en = TIMER_ALARM_EN,
493     .auto_reload = auto_reload,
494 }; // default clock source is APB
495 timer_init(group, timer, &config)
496
497 /* Timer's counter will initially
498     Also, if auto_reload is set, t
499 timer_set_counter_value(group, ti
500
407 example_timer_info_t *info = (exa
408
409 uint64_t timer_counter_value = ti
410
411 /* Prepare basic event data that
412
413 example_timer_event_t evt = {
414     .info.timer_group = info->tim
415     .info.timer_idx = info->timer
416     .info.auto_reload = info->aut
417     .info.alarm_interval = info->
418     .timer_counter_value = timer_
419 };
420
421
422 if (!info->auto_reload) {
423     timer_counter_value += info->
424     timer_group_set_alarm_value_i
425 }
426
427 /* Now just send the event data b
428 xQueueSendFromISR(s_timer_queue,
429
430 return high_task_awoken == pdTRUE
431
432
433
434 @brief Initialize selected timer o
435
436 @param group Timer Group number, i
437 @param timer timer ID, index from
438 @param auto_reload whether auto-re
439 @param timer_interval_sec interval
440
441 tic void ad_tg_timer_init(int grou
442
443 /* Select and initialize basic pa
444 timer_config_t config = {
445     .divider = TIMER_DIVIDER,
446     .counter_dir = TIMER_COUNT_UP
447     .counter_en = TIMER_PAUSE,
448     .alarm_en = TIMER_ALARM_EN,
449     .auto_reload = auto_reload,
450 }; // default clock source is APB
451 timer_init(group, timer, &config)
452
453 /* Timer's counter will initially
454     Also, if auto_reload is set, t
455 timer_set_counter_value(group, ti
456

```

```

501 /* Configure the alarm value and
502 timer_set_alarm_value(group, time
503 timer_enable_intr(group, timer);
504
505 example_timer_info_t *timer_info
506 timer_info->timer_group = group;
507 timer_info->timer_idx = timer;
508 timer_info->auto_reload = auto_re
509 timer_info->alarm_interval = time
510 timer_isr_callback_add(group, tim
511
512 timer_start(group, timer);
513
514
515 tic int s_retry_num = 0;
516 tic void event_handler(void* arg,
517                        int32
518
519
520 if (event_base == WIFI_PROV_EVENT
521     switch (event_id) {
522         case WIFI_PROV_START:
523             ESP_LOGI(TAG, "Provis
524             break;
525         case WIFI_PROV_CRED_RECV:
526             wifi_sta_config_t *wi
527             ESP_LOGI(TAG, "Receiv
528                 "\n\tSSID
529                 (const char
530                 (const char
531             break;
532     }
533     case WIFI_PROV_CRED_FAIL:
534         wifi_prov_sta_fail_re
535         ESP_LOGE(TAG, "Provis
536             "\n\tPlease
537             (*reason ==
538             "Wi-Fi stati
539         break;
540     }
541     case WIFI_PROV_CRED_SUCCE
542         ESP_LOGI(TAG, "Provis
543         break;
544     case WIFI_PROV_END:
545         /* De-initialize mana
546         wifi_prov_mgr_deinit(
547         break;
548     default:
549         break;
550 }

457 /* Configure the alarm value and
458 timer_set_alarm_value(group, time
459 timer_enable_intr(group, timer);
460
461 example_timer_info_t *timer_info
462 timer_info->timer_group = group;
463 timer_info->timer_idx = timer;
464 timer_info->auto_reload = auto_re
465 timer_info->alarm_interval = time
466 timer_isr_callback_add(group, tim
467
468 timer_start(group, timer);
469
470
471 tic int s_retry_num = 0;
472 tic void event_handler(void* arg,
473                        int32
474
475
476 if (event_base == WIFI_PROV_EVENT
477     switch (event_id) {
478         case WIFI_PROV_START:
479             ESP_LOGI(TAG, "Provis
480             break;
481         case WIFI_PROV_CRED_RECV:
482             wifi_sta_config_t *wi
483             ESP_LOGI(TAG, "Receiv
484                 "\n\tSSID
485                 (const char
486                 (const char
487             break;
488     }
489     case WIFI_PROV_CRED_FAIL:
490         wifi_prov_sta_fail_re
491         ESP_LOGE(TAG, "Provis
492             "\n\tPlease
493             (*reason ==
494             "Wi-Fi stati
495         break;
496     }
497     case WIFI_PROV_CRED_SUCCE
498         ESP_LOGI(TAG, "Provis
499         break;
500     case WIFI_PROV_END:
501         /* De-initialize mana
502         wifi_prov_mgr_deinit(
503         break;
504     default:
505         break;
506 }

```

```

551 } else if (event_base == WIFI_EVE 507 } else if (event_base == WIFI_EVE
552     esp_wifi_connect();           508     esp_wifi_connect();
553     ESP_LOGI(TAG, "Received WIFI_ 509     ESP_LOGI(TAG, "Received WIFI_
554 } else if (event_base == IP_EVENT 510 } else if (event_base == IP_EVENT
555     ip_event_got_ip_t* event = (i 511     ip_event_got_ip_t* event = (i
556     ESP_LOGI(TAG, "Connected with 512     ESP_LOGI(TAG, "Connected with
557     s_retry_num = 0;              513     s_retry_num = 0;
558     gpio_set_level(LED_RED, 1);    514     gpio_set_level(LED_RED, 1);
559     ecgState = ECG_IDLE;           515     ecgState = ECG_IDLE;
560     /* Signal main application to 516     /* Signal main application to
561     xEventGroupSetBits(wifi_event 517     xEventGroupSetBits(wifi_event
562 } else if (event_base == WIFI_EVE 518 } else if (event_base == WIFI_EVE
563     if (s_retry_num < EXA 519     if (s_retry_num < EXA
564         esp_wifi_connect( 520         esp_wifi_connect(
565         xEventGroupClearB 521         xEventGroupClearB
566         s_retry_num++;      522         s_retry_num++;
567         gpio_set_level(LE 523         gpio_set_level(LE
568         ecgState = ECG_ER 524         ecgState = ECG_ER
569         ESP_LOGI(TAG, "re 525         ESP_LOGI(TAG, "re
570     } else {                      526     } else {
571         xEventGroupSetBit 527         xEventGroupSetBit
572     }                             528     }
573     ESP_LOGI(TAG, "connect 529     ESP_LOGI(TAG, "connect
574 }                               530 }
575                               531
576                               532
577tic void get_device_service_name(c 533tic void get_device_service_name(c
578                               534
579 uint8_t eth_mac[6];            535 uint8_t eth_mac[6];
580                               536
581 const char *ssid_prefix = "PROV_" 537 const char *ssid_prefix = "PROV_"
582 esp_wifi_get_mac(WIFI_IF_STA, eth 538 esp_wifi_get_mac(WIFI_IF_STA, eth
583 snprintf(service_name, max, "%s%0 539 snprintf(service_name, max, "%s%0
584     ssid_prefix, eth_mac[3],      540     ssid_prefix, eth_mac[3],
585                               541
586                               542
587Handler for the optional provision 543Handler for the optional provision
588The data format can be chosen by a 544The data format can be chosen by a
589Applications can choose to use oth 545Applications can choose to use oth
590                               546
591_err_t custom_prov_data_handler(ui 547_err_t custom_prov_data_handler(ui
592                               548
593                               549
594 if (inbuf) {                    550 if (inbuf) {
595     ESP_LOGI(TAG, "Received data: 551     ESP_LOGI(TAG, "Received data:
596 }                               552 }
597 char response[] = "SUCCESS";     553 char response[] = "SUCCESS";
598 *outbuf = (uint8_t *)strdup(respo 554 *outbuf = (uint8_t *)strdup(respo
599 if (*outbuf == NULL) {          555 if (*outbuf == NULL) {
600     ESP_LOGE(TAG, "System out of 556     ESP_LOGE(TAG, "System out of :

```

```

601     return ESP_ERR_NO_MEM;
602 }
603 *outlen = strlen(response) + 1; /
604
605 return ESP_OK;
606
607
608 static void wifi_prov_print_qr(const
609
610 if (!name || !transport) {
611     ESP_LOGW(TAG, "Cannot generat
612     return;
613 }
614 char payload[150] = {0};
615 if (pop) {
616     snprintf(payload, sizeof(payload),
617              "\",\"pop\\\":\\\"%s\\\",
618              PROV_QR_VERSION,
619 } else {
620     snprintf(payload, sizeof(payload),
621              "\",\"transport\\\":\\
622              PROV_QR_VERSION,
623 }
624 #define CONFIG_EXAMPLE_PROV_SHOW_QR
625 ESP_LOGI(TAG, "Scan this QR code
626 esp_qrcode_config_t cfg = ESP_QRC
627 esp_qrcode_generate(&cfg, payload
628 #if CONFIG_APP_WIFI_PROV_SHOW_Q
629 ESP_LOGI(TAG, "If QR code is not
630
631
632 #if IOT_SUBSCRIBE_CALLBACK_HANDLER
633     I
634     ESP_LOGI(TAG, "Subscribe Callback
635
636
637
638 #if AWS_IOT_DISCONNECT_CALLBACK_HANDLER
639     ESP_LOGW(TAG, "MQTT Disconnect");
640     IoT_Error_t rc = FAILURE;
641
642     if(NULL == pClient) {
643         return;
644     }
645
646     if(aws_iot_is_autoreconnect_enabl
647         ESP_LOGI(TAG, "Auto Reconnect
648 } else {
649     ESP_LOGW(TAG, "Auto Reconnect
650     rc = aws_iot_mqtt_attempt_rec
557     return ESP_ERR_NO_MEM;
558 }
559 *outlen = strlen(response) + 1; /
560
561 return ESP_OK;
562
563
564 static void wifi_prov_print_qr(const
565
566 if (!name || !transport) {
567     ESP_LOGW(TAG, "Cannot generat
568     return;
569 }
570 char payload[150] = {0};
571 if (pop) {
572     snprintf(payload, sizeof(payload),
573              "\",\"pop\\\":\\\"%s\\\",
574              PROV_QR_VERSION,
575 } else {
576     snprintf(payload, sizeof(payload),
577              "\",\"transport\\\":\\
578              PROV_QR_VERSION,
579 }
580 #define CONFIG_EXAMPLE_PROV_SHOW_QR
581 ESP_LOGI(TAG, "Scan this QR code
582 esp_qrcode_config_t cfg = ESP_QRC
583 esp_qrcode_generate(&cfg, payload
584 #if CONFIG_APP_WIFI_PROV_SHOW_Q
585 ESP_LOGI(TAG, "If QR code is not
586
587
588 #if IOT_SUBSCRIBE_CALLBACK_HANDLER
589     I
590     ESP_LOGI(TAG, "Subscribe Callback
591
592
593
594 #if AWS_IOT_DISCONNECT_CALLBACK_HANDLER
595     ESP_LOGW(TAG, "MQTT Disconnect");
596     IoT_Error_t rc = FAILURE;
597
598     if(NULL == pClient) {
599         return;
600     }
601
602     if(aws_iot_is_autoreconnect_enabl
603         ESP_LOGI(TAG, "Auto Reconnect
604 } else {
605     ESP_LOGW(TAG, "Auto Reconnect
606     rc = aws_iot_mqtt_attempt_rec

```

```

651         if(NETWORK_RECONNECTE 607         if(NETWORK_RECONNECTE
652             ESP_LOGW(TAG, "Manual Rec 608             ESP_LOGW(TAG, "Manual Rec
653     } else { 609     } else {
654         ESP_LOGW(TAG, "Manual Rec 610         ESP_LOGW(TAG, "Manual Rec
655     } 611     }
656 } 612 }
657 613
658 614
659 615
660 d aws_iot_task(void *param) { 616 d aws_iot_task(void *param) {
661     char cPayload[100]; 617     char cPayload[100];
662 618
663     short ecgHandsOn; 619     short ecgHandsOn;
664     unsigned int ecgAcqCounter; 620     unsigned int ecgAcqCounter;
665     unsigned int ecgRecCounter; 621     unsigned int ecgRecCounter;
666     unsigned int ecgMqttCounter; 622     unsigned int ecgMqttCounter;
667     unsigned short *headBuffer; 623     unsigned short *headBuffer;
668     unsigned char *headTxBuffer; 624     unsigned char *headTxBuffer;
669     //unsigned char *mqttTxBuffer; 625     //unsigned char *mqttTxBuffer;
670     unsigned short sequenceTimer; 626     unsigned short sequenceTimer;
671     unsigned short oldSequenceTimer; 627     unsigned short oldSequenceTimer;
672     unsigned short ledSelect; 628     unsigned short ledSelect;
673 629
674     char strftime_buf[64]; // for snt 630     char strftime_buf[64]; // for snt
675 631
676 632
677     //ycc 031221 end 633     //ycc 031221 end
678     ecgState = (short) ECG_IDLE; 634     ecgState = (short) ECG_IDLE;
679     ecgHandsOn = 0; 635     ecgHandsOn = 0;
680     ecgAcqCounter = 0; 636     ecgAcqCounter = 0;
681     ecgRecCounter = 0; 637     ecgRecCounter = 0;
682     ecgMqttCounter = 0; 638     ecgMqttCounter = 0;
683     sequenceTimer = 0; 639     sequenceTimer = 0;
684     oldSequenceTimer=0; 640     oldSequenceTimer=0;
685     ledSelect = LED_GREEN; 641     ledSelect = LED_GREEN;
686     jobCheckState = JOB_CHECK_STATE_N 642     jobCheckState = JOB_CHECK_STATE_N
687 643
688     AWS_IoT_Client client; 644     AWS_IoT_Client client;
689 645
690     headTxBuffer = (unsigned char *)c 646     headTxBuffer = (unsigned char *)c
691     if(headTxBuffer == NULL) 647     if(headTxBuffer == NULL)
692     { 648     {
693         ESP_LOGE(TAG, "Failed to 649         ESP_LOGE(TAG, "Failed to
694         abort(); 650         abort();
695     } 651     }
696 652
697     headBuffer = (char *)headTxBuffer 653     headBuffer = (char *)headTxBuffer
698     dataBuffer = headBuffer; 654     dataBuffer = headBuffer;
699 655
700     //mqttTxBuffer = (unsigned char * 656     //mqttTxBuffer = (unsigned char *

```



```

701 //if(mqttTxBuffer == NULL)
702 //    {
703 //        ESP_LOGE(TAG, "Failed t
704 //        abort();
705 //    }
706
707
708 uint8_t brd_mac[6];
709 char topic_name[16];
710 const char *topic_prefix = "ecg/"
711 esp_wifi_get_mac(WIFI_IF_STA, brd
712 snprintf(topic_name, 18, "%s%02X%
713         topic_prefix, brd_mac[0]
714
715 ESP_LOGI(TAG, "calloc return %x",
716
717 int32_t i = 0;
718
719 IoT_Error_t rc = FAILURE;
720
721
722
723
724 /* Wait for WiFi to show as conne
725 int bits = xEventGroupWaitBits(wi
726         false, true,
727
728 if (!(bits & CONNECTED_BIT)) {
729     ESP_LOGE(TAG, "timeout bits=%
730     gpio_set_level(LED_RED, 0);
731     ecgState = ECG_ERROR_WIFI;
732 }
733
734 int counter = 0;
735 //int counter2=0;
736 int counter3 = 0;
737 unsigned long deepSleepCounter =1
738 int debounceCounter = 101;
739 int jobCheckCounter = 0;
740 //turn off all LEDs so that the L
741 gpio_set_level(LED_GREEN, 1);
742 gpio_set_level(LED_BLUE, 1);
743 gpio_set_level(LED_RED, 1);
744
745
746 initialize_sntp();
747
748 // wait for time to be set
749
750 time_t now = 0;

```

```

657 //if(mqttTxBuffer == NULL)
658 //    {
659 //        ESP_LOGE(TAG, "Failed t
660 //        abort();
661 //    }
662
663
664 uint8_t brd_mac[6];
665 char topic_name[16];
666 const char *topic_prefix = "ecg/"
667 esp_wifi_get_mac(WIFI_IF_STA, brd
668 snprintf(topic_name, 18, "%s%02X%
669         topic_prefix, brd_mac[0]
670
671 ESP_LOGI(TAG, "calloc return %x",
672
673 int32_t i = 0;
674
675 IoT_Error_t rc = FAILURE;
676
677
678
679
680 /* Wait for WiFi to show as conne
681 int bits = xEventGroupWaitBits(wi
682         false, true,
683
684 if (!(bits & CONNECTED_BIT)) {
685     ESP_LOGE(TAG, "timeout bits=%
686     gpio_set_level(LED_RED, 0);
687     ecgState = ECG_ERROR_WIFI;
688 }
689
690 int counter = 0;
691 //int counter2=0;
692 int counter3 = 0;
693 int debounceCounter = 101;
694 int jobCheckCounter = 0;
695 //turn off all LEDs so that the L
696 gpio_set_level(LED_GREEN, 1);
697 gpio_set_level(LED_BLUE, 1);
698 gpio_set_level(LED_RED, 1);
699
700
701 initialize_sntp();
702
703 // wait for time to be set
704
705 time_t now = 0;

```



```

751 struct tm timeinfo = { 0 };
752 int retry = 0;
753 const int retry_count = 10;
754 while (snrt_get_sync_status() ==
755         ESP_LOGI(TAG, "Waiting for sy
756         vTaskDelay(2000 / portTICK_PE
757 }
758 time(&now);
759 localtime_r(&now, &timeinfo);
760 setenv("TZ", "EST5EDT,M3.2.0/2,M1
761 tzset();
762 struct tm *tm_struct = localtime_
763 strftime(strftime_buf, sizeof(str
764 ESP_LOGI(TAG, "%s= %d", strftime
765
766 le (1) {
767     example_timer_event_t evt;
768     xQueueReceive(s_timer_queue,
769
770     counter++;
771     oldSequenceTimer = sequenceTi
772     sequenceTimer = (unsigned sho
773     //select LED color based on t
774     if(sequenceTimer!= oldSequenc
775         ledSelect = LED_GREEN;
776         switch(ecgState){
777             case ECG_IDLE:
778                 ledSelect = LED_G
779                 break;
780             case ECG_ACQUIRING:
781             case ECG_RECORDING:
782             case ECG_SENDING_MQTT
783                 ledSelect = LED_B
784                 break;
785             case ECG_SSID_RESET:
786                 ledSelect= LED_R
787                 break;
788             case ECG_FINISH:
789                 ledSelect = LED_G
790                 break;
791             case ECG_ERROR_WIFI:
792             case ECG_ERROR_MQTT:
793             default:
794                 ledSelect = LED_R
795                 break;
796
797     }
798     gpio_set_level(LED_GREEN,
799     gpio_set_level(LED_BLUE,
800     gpio_set_level(LED_RED, 1

```

```

706 struct tm timeinfo = { 0 };
707 int retry = 0;
708 const int retry_count = 10;
709 while (snrt_get_sync_status() ==
710         ESP_LOGI(TAG, "Waiting for sy
711         vTaskDelay(2000 / portTICK_PE
712 }
713 time(&now);
714 localtime_r(&now, &timeinfo);
715 setenv("TZ", "EST5EDT,M3.2.0/2,M1
716 tzset();
717 struct tm *tm_struct = localtime_
718 strftime(strftime_buf, sizeof(str
719 ESP_LOGI(TAG, "%s= %d", strftime
720
721 le (1) {
722     example_timer_event_t evt;
723     xQueueReceive(s_timer_queue,
724
725     counter++;
726     oldSequenceTimer = sequenceTi
727     sequenceTimer = (unsigned sho
728     //select LED color based on t
729     if(sequenceTimer!= oldSequenc
730         ledSelect = LED_GREEN;
731         switch(ecgState){
732             case ECG_IDLE:
733                 ledSelect = LED_G
734                 break;
735             case ECG_ACQUIRING:
736             case ECG_RECORDING:
737             case ECG_SENDING_MQTT
738                 ledSelect = LED_B
739                 break;
740             case ECG_SSID_RESET:
741                 ledSelect = LED_R
742                 break;
743             case ECG_FINISH:
744                 ledSelect = LED_G
745                 break;
746             case ECG_ERROR_WIFI:
747             case ECG_ERROR_MQTT:
748             default:
749                 ledSelect = LED_R
750                 break;
751
752     }
753     gpio_set_level(LED_GREEN,
754     gpio_set_level(LED_BLUE,
755     gpio_set_level(LED_RED, 1

```

801		756	
802	gpio_set_level(ledSelect,	757	gpio_set_level(ledSelect,
803		758	
804	}	759	}
805	counter3++;	760	counter3++;
806	jobCheckCounter++;	761	jobCheckCounter++;
807	if (counter >300)	762	if (counter >300)
808	{	763	{
809	//printf("-----	764	//printf("-----
810	counter = 0; //this count	765	counter = 0; //this count
811	//ycc 032122	766	//ycc 032122
812		767	
813	//uint32_t adc_reading =	768	//uint32_t adc_reading =
814	//Multisampling	769	//Multisampling
815	//for (int i = 0; i < NO_	770	//for (int i = 0; i < NO_
816	// adc_reading += adc1	771	// adc_reading += adc1
817		772	
818	//adc_reading /= NO_OF_SA	773	//adc_reading /= NO_OF_SA
819	//Convert adc_reading to	774	//Convert adc_reading to
820	//uint32_t voltage = esp_	775	//uint32_t voltage = esp_
821	//printf("Raw: %d\tVolutag	776	//printf("Raw: %d\tVolutag
822	//ESP_LOGI(TAG, "%f,%f,%f	777	//ESP_LOGI(TAG, "%f,%f,%f
823	//if(ecgState != ECG_IDLE	778	//if(ecgState != ECG_IDLE
824	uint32_t fs = xPortGetFre	779	uint32_t fs = xPortGetFre
825	ESP_LOGI(TAG, "ecg state	780	ESP_LOGI(TAG, "ecg state
826		781	
827	time(&now);	782	time(&now);
828	//localtime_r(&now, &time	783	//localtime_r(&now, &time
829	struct tm *tm_struct = lo	784	struct tm *tm_struct = lo
830	strftime(strftime_buf, si	785	strftime(strftime_buf, si
831	ESP_LOGI(TAG, "%s= %d",	786	ESP_LOGI(TAG, "%s= %d",
832	ESP_LOGI(TAG, "%f,%f,%f,%	787	ESP_LOGI(TAG, "%f,%f,%f,%
833	if((tm_struct->tm_hour==0	788	if((tm_struct->tm_hour==0
834	jobCheckState = JOB_C	789	jobCheckState = JOB_C
835	// if time is past midnig	790	// if time is past midnig
836	if((tm_struct->tm_hour ==	791	if((tm_struct->tm_hour ==
837	gpio_set_level(LED_GR	792	gpio_set_level(LED_GR
838	gpio_set_level(LED_BL	793	gpio_set_level(LED_BL
839	gpio_set_level(LED_RE	794	gpio_set_level(LED_RE
840	aws_iot_demo_main(0,	795	aws_iot_demo_main(0,
841	jobCheckState = JOB_	796	jobCheckState = JOB_
842	jobCheckCounter = 0;	797	jobCheckCounter = 0;
843	gpio_set_level(LED_GR	798	gpio_set_level(LED_GR
844	gpio_set_level(LED_BL	799	gpio_set_level(LED_BL
845	gpio_set_level(LED_RE	800	gpio_set_level(LED_RE
846	}	801	}
847	*/	802	*/
848	}	803	}
849		804	
850	if(((gpio_get_level((gpio_num	805	if(((gpio_get_level((gpio_num

851	{	806	{
852	pVariance=300000; pAvg=30	807	pVariance=300000; pAvg=30
853	variance = 1000;	808	variance = 1000;
854	//counter = 0;	809	//counter = 0;
855	//counter3=0;	810	//counter3=0;
856	ecgHandsOn = 0;	811	ecgHandsOn = 0;
857		812	
858	//if((gpio_get_level((gpi	813	//if((gpio_get_level((gpi
859	// {	814	// {
860	// ecgState = ECG_	815	// ecgState = ECG_
861	// }	816	// }
862	}	817	}
863	else	818	else
864	{	819	{
865	//ycc 022022	820	//ycc 022022
866	if(debounceCounter<=100){	821	if(debounceCounter<=100){
867	debounceCounter++;	822	debounceCounter++;
868	}	823	}
869	else {	824	else {
870	debounceCounter = 0;	825	debounceCounter = 0;
871	}	826	}
872	ecgHandsOn = 1;	827	ecgHandsOn = 1;
873	gpio_set_level(LED_RED, 1	828	gpio_set_level(LED_RED, 1
874	// send the value of anal	829	// send the value of anal
875	//Data = 0;	830	//Data = 0;
876	//for (int i = 0; i < NO_	831	//for (int i = 0; i < NO_
877	aData = adc1_get_raw((adc	832	aData = adc1_get_raw((adc
878	//aData = aData/NO_OF_SAM	833	//aData = aData/NO_OF_SAM
879	if((aData<4090) && (aData	834	if((aData<4090) && (aData
880	bpfx = (float)aData;	835	bpfx = (float)aData;
881	/*	836	/*
882	w0 = 3.336612*w1 -4.22598	837	w0 = 3.336612*w1 -4.22598
883	bpfx1 = 0.036575*(w0 - 2.	838	bpfx1 = 0.036575*(w0 - 2.
884	w4 = w3;	839	w4 = w3;
885	w3 = w2;	840	w3 = w2;
886	w2 = w1;	841	w2 = w1;
887	w1 = w0;	842	w1 = w0;
888	*/	843	*/
889	w0 = 3.269793*w1 -4.16941	844	w0 = 3.269793*w1 -4.16941
890	bpfx1 = 0.082619*(w0 - 2.	845	bpfx1 = 0.082619*(w0 - 2.
891	w4 = w3;	846	w4 = w3;
892	w3 = w2;	847	w3 = w2;
893	w2 = w1;	848	w2 = w1;
894	w1 = w0;	849	w1 = w0;
895	w0 = 3.000162*w1 -3.29408	850	w0 = 3.000162*w1 -3.29408
896	bpfx1 = 0.065274*(w0 - 2.	851	bpfx1 = 0.065274*(w0 - 2.
897	w4 = w3;	852	w4 = w3;
898	w3 = w2;	853	w3 = w2;
899	w2 = w1;	854	w2 = w1;
900	w1 = w0;	855	w1 = w0;

901	//end-band pass filter	856	//end-band pass filter
902		857	
903	bpfs0=bpfs1;	858	bpfs0=bpfs1;
904	bpfs1=bpfs2;	859	bpfs1=bpfs2;
905	bpfs2=bpfs3;	860	bpfs2=bpfs3;
906	bpfs3=bpfs4;	861	bpfs3=bpfs4;
907	bpfs4=bpfs5;	862	bpfs4=bpfs5;
908	bpfs5=bpfs6;	863	bpfs5=bpfs6;
909	bpfs6=bpfX1;	864	bpfs6=bpfX1;
910		865	
911	if (bpfs3>(mAvg+noiseFloo	866	if (bpfs3>(mAvg+noiseFloo
912	{	867	{
913	//heart beat detected	868	//heart beat detected
914		869	
915	heartRate = 30000/pea	870	heartRate = 30000/pea
916		871	
917	m0=m1;	872	m0=m1;
918	m1=m2;	873	m1=m2;
919	m2=m3;	874	m2=m3;
920	m3=m4;	875	m3=m4;
921	//if(bpfs3> 40 && bpf	876	//if(bpfs3> 40 && bpf
922	m4=bpfs3;	877	m4=bpfs3;
923	mAvg = (m0+m1+m2+m3+m	878	mAvg = (m0+m1+m2+m3+m
924	//mVariance = (m0-mAv	879	//mVariance = (m0-mAv
925	p0=p1;	880	p0=p1;
926	p1=p2;	881	p1=p2;
927	p2=p3;	882	p2=p3;
928	p3=p4;	883	p3=p4;
929	p4=peakCounter;	884	p4=peakCounter;
930		885	
931	pAvg = (p0+p1+p2+p3+p	886	pAvg = (p0+p1+p2+p3+p
932	pVariance = ((p0-pAvg	887	pVariance = ((p0-pAvg
933	variance = sqrt(pVari	888	variance = sqrt(pVari
934		889	
935	//digitalWrite(LEDPin	890	//digitalWrite(LEDPin
936	detectedBeat++;	891	detectedBeat++;
937	bpftmp = bpfs3;	892	bpftmp = bpfs3;
938	peakCounter = 0;	893	peakCounter = 0;
939	}	894	}
940	else	895	else
941	{	896	{
942	if(peakCounter> 5000)	897	if(peakCounter> 5000)
943	{	898	{
944	peakCounter = 0;	899	peakCounter = 0;
945	detectedBeat--;	900	detectedBeat--;
946	}	901	}
947		902	
948	peakCounter++;	903	peakCounter++;
949	if ((bpfs3>bpfs2) && (904	if ((bpfs3>bpfs2) && (
950	{	905	{

951		906	
952	noiseFloor = (noi	907	noiseFloor = (noi
953	if(peakCounter>50	908	if(peakCounter>50
954	}	909	}
955	bpftmp = 0;	910	bpftmp = 0;
956	} //end if-else	911	} //end if-else
957		912	
958	}//end if-else	913	}//end if-else
959	//state machine	914	//state machine
960		915	
961	switch(ecgState)	916	switch(ecgState)
962	{	917	{
963	case ECG_IDLE:	918	case ECG_IDLE:
964	if(ecgHandsOn == 1){	919	if(ecgHandsOn == 1){
965	//printf("IDLE mo	920	//printf("IDLE mo
966	ecgState = ECG_AC	921	ecgState = ECG_AC
967	//start ACQ timer	922	//start ACQ timer
968	ecgAcqCounter = 0	923	ecgAcqCounter = 0
969	ecgRecCounter = 0	924	ecgRecCounter = 0
970	ecgMqttCounter =	925	ecgMqttCounter =
971	dataBuffer = head	926	dataBuffer = head
972	deepSleepCounter		
973	}		
974	else {		
975	deepSleepCounter-		
976	if(deepSleepCount		
977	//deep sleep		
978	// Initialize		
979	// The defaul		
980			
981	ESP_ERROR_CHE		
982	// If use tou		
983			
984	touch_pad_set		
985	// Set refere		
986	// In this ca		
987	// The low re		
988	// The larger		
989			
990	touch_pad_set		
991			
992	//init RTC IO		
993	// touch pad		
994			
995	touch_pad_con		
996			
997	calibrate_tou		
998			
999	printf("Enabl		
1000	esp_sleep_ena		

1001	esp_sleep_pd_		
1002			
1003	// Isolate GP		
1004	// which have		
1005	// to minimiz		
1006	rtc_gpio_isol		
1007	gpio_set_leve		
1008	printf("Enter		
1009	esp_deep_slee		
1010	}		
1011	}	927	}
1012	break;	928	break;
1013	case ECG_ACQUIRING:	929	case ECG_ACQUIRING:
1014	ecgAcqCounter++;	930	ecgAcqCounter++;
1015	if(ecgHandsOn == 0)	931	if(ecgHandsOn == 0)
1016	{	932	{
1017	//hands off go ba	933	//hands off go ba
1018	printf("ACQUIRING	934	printf("ACQUIRING
1019	ecgState = ECG_ID	935	ecgState = ECG_ID
1020	ecgAcqCounter = 0	936	ecgAcqCounter = 0
1021	}	937	}
1022	else if (ecgAcqCounte	938	else if (ecgAcqCounte
1023	{//can't recogniz	939	{//can't recogniz
1024	printf("ACQUIRING	940	printf("ACQUIRING
1025	ecgState = ECG_RE	941	ecgState = ECG_RE
1026	//start recording	942	//start recording
1027	ecgRecCounter = 0	943	ecgRecCounter = 0
1028	//set up pointer	944	//set up pointer
1029	}	945	}
1030	else if (variance < 1	946	else if (variance < 1
1031	{	947	{
1032	//printf("varianc	948	//printf("varianc
1033	ecgState = ECG_RE	949	ecgState = ECG_RE
1034	//start recording	950	//start recording
1035	ecgRecCounter = 0	951	ecgRecCounter = 0
1036	//ecg signal reco	952	//ecg signal reco
1037	//set up pointer	953	//set up pointer
1038	}	954	}
1039	break;	955	break;
1040	case ECG_RECORDING:	956	case ECG_RECORDING:
1041	//dataBuffer[ecgR	957	//dataBuffer[ecgR
1042	*dataBuffer++ = a	958	*dataBuffer++ = a
1043	ecgRecCounter++;	959	ecgRecCounter++;
1044	if(ecgHandsOn ==	960	if(ecgHandsOn ==
1045	{ //hands o	961	{ //hands o
1046	//printf(962	//printf(
1047	if (ecgRe	963	if (ecgRe
1048	{ //S	964	{ //S
1049	/	965	/
1050	e	966	e

```

1051                                / 967                                /
1052                                e 968                                e
1053                                } 969                                }
1054                                else 970                                else
1055                                //hands off g 971                                //hands off g
1056                                { 972                                {
1057                                ecgState 973                                ecgState
1058                                //printf( 974                                //printf(
1059                                } 975                                }
1060                                } 976                                }
1061                                else if (ecgRecCo 977                                else if (ecgRecCo
1062                                { //finish tim 978                                { //finish tim
1063                                //printf( 979                                //printf(
1064                                ecgState 980                                ecgState
1065                                //ecgRecC 981                                //ecgRecC
1066                                ecgMqttCo 982                                ecgMqttCo
1067                                dataBuffe 983                                dataBuffe
1068                                } 984                                }
1069                                985
1070                                break; 986                                break;
1071                                case ECG_SENDING_MQTT: 987                                case ECG_SENDING_MQTT:
1072                                //AWS_IoT_Client clie 988                                //AWS_IoT_Client clie
1073                                rc = FAILURE; 989                                rc = FAILURE;
1074                                gpio_set_level(LED_BL 990                                gpio_set_level(LED_BL
1075                                //ycc 3-13-22 added t 991                                //ycc 3-13-22 added t
1076                                /* Return error statu 992                                /* Return error statu
1077                                int returnStatus = EX 993                                int returnStatus = EX
1078                                994
1079                                995
1080                                if( returnStatus == E 996                                if( returnStatus == E
1081                                { 997                                {
1082                                /* Start OTA demo 998                                /* Start OTA demo
1083                                //returnStatus = 999                                //returnStatus =
1084                                //ycc 031422 1000                                //ycc 031422
1085                                uint32_t free_he 1001                                uint32_t free_he
1086                                free_heap_size = 1002                                free_heap_size =
1087                                min_free_heap_siz 1003                                min_free_heap_siz
1088                                //printf("\n free 1004                                //printf("\n free
1089                                uint32_t fs = xPo 1005                                uint32_t fs = xPo
1090                                ESP_LOGI(TAG, "Se 1006                                ESP_LOGI(TAG, "Se
1091                                if( mqttSessionEs 1007                                if( mqttSessionEs
1092                                int ret = est 1008                                int ret = est
1093                                //printf("est 1009                                //printf("est
1094                                //printf("Pub 1010                                //printf("Pub
1095                                } 1011                                }
1096                                if( mqttSessionEs 1012                                if( mqttSessionEs
1097                                1013
1098                                unsigned shor 1014                                unsigned shor
1099                                char topic_na 1015                                char topic_na
1100                                const char *t 1016                                const char *t

```


subscribe_publish_sample.c x 2

1101	uint8_t brd_m	1017	uint8_t brd_m
1102	esp_wifi_get_	1018	esp_wifi_get_
1103	snprintf(topi	1019	snprintf(topi
1104	topic_pre	1020	topic_pre
1105	const char *T	1021	const char *T
1106	const int TOP	1022	const int TOP
1107	mqttPublishNo	1023	mqttPublishNo
1108		1024	
1109		1025	
1110		1026	
1111		1027	
1112		1028	
1113	}	1029	}
1114		1030	
1115		1031	
1116		1032	
1117	}	1033	}
1118		1034	
1119	//ycc end 3-13-22	1035	//ycc end 3-13-22
1120	//ycc 031422 end	1036	//ycc 031422 end
1121	disconnect();	1037	disconnect();
1122	ecgState = ECG_FINIS	1038	ecgState = ECG_FINIS
1123	ecgRecCounter = 0;	1039	ecgRecCounter = 0;
1124	jobCheckCounter = 0;	1040	jobCheckCounter = 0;
1125		1041	
1126	ecgMqttCounter++;	1042	ecgMqttCounter++;
1127	break;	1043	break;
1128	case ECG_FINISH:	1044	case ECG_FINISH:
1129	if(ecgHandsOn == 0){	1045	if(ecgHandsOn == 0){
1130	ecgState = ECG_ID	1046	ecgState = ECG_ID
1131	jobCheckCounter =	1047	jobCheckCounter =
1132	break;	1048	break;
1133	}	1049	}
1134	else {	1050	else {
1135		1051	
1136	if(jobCheckCo	1052	if(jobCheckCo
1137	ecgState	1053	ecgState
1138	break;	1054	break;
1139	}	1055	}
1140	else{	1056	else{
1141	gpio_set_	1057	gpio_set_
1142	gpio_set_	1058	gpio_set_
1143	gpio_set_	1059	gpio_set_
1144		1060	
1145	//ESP_LOG	1061	//ESP_LOG
1146	// ycc 03	1062	// ycc 03
1147	//int ret	1063	//int ret
1148	//ESP_LOG	1064	//ESP_LOG
1149	//if(ret=	1065	//if(ret=
1150	// job	1066	// job

```

1151             jobCheckC 1067             jobCheckC
1152             ecgState 1068             ecgState
1153             gpio_set_ 1069             gpio_set_
1154             gpio_set_ 1070             gpio_set_
1155             gpio_set_ 1071             gpio_set_
1156                                     1072
1157             break; 1073             break;
1158         } 1074         }
1159                                     1075
1160     } 1076     }
1161     break; 1077     break;
1162     case ECG_OTA_UPDATE: 1078     case ECG_OTA_UPDATE:
1163         ecgState = ECG_OTA_UP 1079         ecgState = ECG_OTA_UP
1164         break; 1080         break;
1165     case ECG_SSID_RESET: 1081     case ECG_SSID_RESET:
1166         if(counter3 < 3000) 1082         if(counter3 < 3000)
1167             break; 1083             break;
1168         else 1084         else
1169             { 1085             {
1170                 ecgState = ECG_ID 1086                 ecgState = ECG_ID
1171                 counter3 = 0; 1087                 counter3 = 0;
1172             } 1088             }
1173             break; 1089             break;
1174     default: 1090     default:
1175         //printf("Default mov 1091         //printf("Default mov
1176         ecgState = ECG_IDLE; 1092         ecgState = ECG_IDLE;
1177         break; 1093         break;
1178     } 1094     }
1179                                     1095
1180 } //end while 1096 } //end while
1181                                     1097
1182 ESP_LOGE(TAG, "An error occurred 1098 ESP_LOGE(TAG, "An error occurred
1183 abort(); 1099 abort();
1184                                     1100
1185                                     1101
1186                                     1102
1187 d app_main() 1103 d app_main()
1188                                     1104
1189 esp_chip_info_t chip_info; 1105 esp_chip_info_t chip_info;
1190 esp_chip_info(&chip_info); 1106 esp_chip_info(&chip_info);
1191 /* 1107 /*
1192 printf("This is %s chip with %d C 1108 printf("This is %s chip with %d C
1193     CHIP_NAME, 1109     CHIP_NAME,
1194     chip_info.cores, 1110     chip_info.cores,
1195     (chip_info.features & CHI 1111     (chip_info.features & CHI
1196     (chip_info.features & CHI 1112     (chip_info.features & CHI
1197 */ 1113 */
1198 //printf("silicon revision %d, ", 1114 //printf("silicon revision %d, ",
1199                                     1115
1200 //printf("%dMB %s flash\n", spi_f 1116 //printf("%dMB %s flash\n", spi_f

```

```

1201 //      (chip_info.features & C 1117 //      (chip_info.features & C
1202                                     1118
1203 //072521 ycc added the following 1119 //072521 ycc added the following
1204 app_driver_init();                1120 app_driver_init();
1205                                     1121
1206 //printf("Configured WiFi SSID is 1122 //printf("Configured WiFi SSID is
1207 // Initialize NVS.                 1123 // Initialize NVS.
1208 esp_err_t err = nvs_flash_init(); 1124 esp_err_t err = nvs_flash_init();
1209 if (err == ESP_ERR_NVS_NO_FREE_PA 1125 if (err == ESP_ERR_NVS_NO_FREE_PA
1210     ESP_ERROR_CHECK(nvs_flash_era 1126     ESP_ERROR_CHECK(nvs_flash_era
1211     err = nvs_flash_init();        1127     err = nvs_flash_init();
1212 }                                  1128 }
1213 ESP_ERROR_CHECK( err );           1129 ESP_ERROR_CHECK( err );
1214                                     1130
1215                                     1131
1216 s_timer_queue = xQueueCreate(10, 1132 s_timer_queue = xQueueCreate(10,
1217 //GPIO setup                       1133 //GPIO setup
1218 gpio_reset_pin(32);                1134
1219 touch_pad_set_fsm_mode(TOUCH_FSM_ 1135
1220 gpio_set_direction(LOPlus, GPIO_M 1136 gpio_set_direction(LOPlus, GPIO_M
1221 gpio_set_direction(LOMinus, GPIO_ 1137 gpio_set_direction(LOMinus, GPIO_
1222 gpio_set_direction(SDN, GPIO_MODE 1138 gpio_set_direction(SDN, GPIO_MODE
1223 gpio_set_direction(FR, GPIO_MODE_ 1139 gpio_set_direction(FR, GPIO_MODE_
1224 gpio_set_direction(DC, GPIO_MODE_ 1140 gpio_set_direction(DC, GPIO_MODE_
1225 gpio_set_direction(LED_GREEN, GPI 1141 gpio_set_direction(LED_GREEN, GPI
1226 gpio_set_direction(LED_BLUE, GPIO 1142 gpio_set_direction(LED_BLUE, GPIO
1227 gpio_set_direction(LED_RED, GPIO_ 1143 gpio_set_direction(LED_RED, GPIO_
1228                                     1144
1229 gpio_set_direction(DEEPSLEEP, GPI 1145
1230                                     1146
1231 gpio_set_level(LED_GREEN, 1);       1147 gpio_set_level(LED_GREEN, 1);
1232 gpio_set_level(LED_BLUE, 1);       1148 gpio_set_level(LED_BLUE, 1);
1233 gpio_set_level(LED_RED, 1);        1149 gpio_set_level(LED_RED, 1);
1234 gpio_set_level(SDN, 1);            1150 gpio_set_level(SDN, 1);
1235 gpio_set_level(FR, 1);             1151 gpio_set_level(FR, 1);
1236 gpio_set_level(DC, 0);             1152 gpio_set_level(DC, 0);
1237                                     1153
1238 //gpio_set_level(SDN, 0);          1154 //gpio_set_level(SDN, 0);
1239 //ESP_ERROR_CHECK(esp_sleep_disab 1155
1240 //ESP_ERROR_CHECK(touch_pad_deini 1156
1241                                     1157
1242 ad_tg_timer_init(TIMER_GROUP_0, T 1158 ad_tg_timer_init(TIMER_GROUP_0, T
1243 /*example_tg_timer_init(TIMER_GRO 1159 /*example_tg_timer_init(TIMER_GRO
1244 //Check if Two Point or Vref are 1160 //Check if Two Point or Vref are
1245 check_efuse();                    1161 check_efuse();
1246                                     1162
1247 //Configure ADC                    1163 //Configure ADC
1248 adcl_config_width(width);          1164 adcl_config_width(width);
1249 adcl_config_channel_atten(channel 1165 adcl_config_channel_atten(channel
1250                                     1166

```

```

1251                                     1160
1252 //Characterize ADC                    1161 //Characterize ADC
1253 adc_chars = calloc(1, sizeof(esp_ 1162 adc_chars = calloc(1, sizeof(esp_
1254 esp_adc_cal_value_t val_type = es 1163 esp_adc_cal_value_t val_type = es
1255 print_char_val_type(val_type);      1164 print_char_val_type(val_type);
1256                                     1165
1257                                     1166
1258 /* 072221 ycc */                     1167 /* 072221 ycc */
1259 tcpip_adapter_init(); // move fro 1168 tcpip_adapter_init(); // move fro
1260                                     1169
1261 ESP_ERROR_CHECK(esp_event_loop_cr 1170 ESP_ERROR_CHECK(esp_event_loop_cr
1262 wifi_event_group = xEventGroupCre 1171 wifi_event_group = xEventGroupCre
1263                                     1172
1264 * Register our event handler for W 1173 * Register our event handler for W
1265 ESP_ERROR_CHECK(esp_event_handler 1174 ESP_ERROR_CHECK(esp_event_handler
1266 ESP_ERROR_CHECK(esp_event_handler 1175 ESP_ERROR_CHECK(esp_event_handler
1267 ESP_ERROR_CHECK(esp_event_handler 1176 ESP_ERROR_CHECK(esp_event_handler
1268                                     1177
1269 //072321 ycc commented out the fo 1178 //072321 ycc commented out the fo
1270 //ESP_ERROR_CHECK( esp_event_loop 1179 //ESP_ERROR_CHECK( esp_event_loop
1271 wifi_init_config_t cfg = WIFI_INI 1180 wifi_init_config_t cfg = WIFI_INI
1272 ESP_ERROR_CHECK( esp_wifi_init(&c 1181 ESP_ERROR_CHECK( esp_wifi_init(&c
1273                                     1182
1274                                     1183
1275 //initialise_wifi();                 1184 //initialise_wifi();
1276 /* 072221 ycc add Configuration f 1185 /* 072221 ycc add Configuration f
1277 wifi_prov_mgr_config_t config = { 1186 wifi_prov_mgr_config_t config = {
1278     /* What is the Provisioning S 1187     /* What is the Provisioning S
1279     * wifi_prov_scheme_softap or 1188     * wifi_prov_scheme_softap or
1280     .scheme = wifi_prov_scheme_bl 1189     .scheme = wifi_prov_scheme_bl
1281                                     1190
1282     /* Any default scheme specifi 1191     /* Any default scheme specifi
1283     * like to choose. Since our 1192     * like to choose. Since our
1284     * neither BT nor BLE, we can 1193     * neither BT nor BLE, we can
1285     * memory once provisioning i 1194     * memory once provisioning i
1286     * (in case when device is al 1195     * (in case when device is al
1287     * appropriate scheme specifi 1196     * appropriate scheme specifi
1288     * to take care of this autom 1197     * to take care of this autom
1289     * WIFI_PROV_EVENT_HANDLER_NO 1198     * WIFI_PROV_EVENT_HANDLER_NO
1290     .scheme_event_handler = WIFI_ 1199     .scheme_event_handler = WIFI_
1291                                     1200
1292                                     1201
1293 };                                   1202 };
1294                                     1203
1295 /* Initialize provisioning manage 1204 /* Initialize provisioning manage
1296 * configuration parameters set a 1205 * configuration parameters set a
1297 ESP_ERROR_CHECK(wifi_prov_mgr_ini 1206 ESP_ERROR_CHECK(wifi_prov_mgr_ini
1298                                     1207
1299 bool provisioned = false;           1208 bool provisioned = false;
1300 /* Let's find out if the device i 1209 /* Let's find out if the device i

```

```

1301 ESP_ERROR_CHECK(wifi_prov_mgr_is_ 1210 ESP_ERROR_CHECK(wifi_prov_mgr_is_
1302                                1211
1303 /* If device is not yet provision 1212 /* If device is not yet provision
1304 if (!provisioned) { 1213 if (!provisioned) {
1305     ESP_LOGI(TAG, "Starting provi 1214     ESP_LOGI(TAG, "Starting provi
1306     gpio_set_level(LED_RED, 0); / 1215     gpio_set_level(LED_RED, 0); /
1307     /* What is the Device Service 1216     /* What is the Device Service
1308     * This translates to : 1217     * This translates to :
1309     *     - Wi-Fi SSID when sche 1218     *     - Wi-Fi SSID when sche
1310     *     - device name when sch 1219     *     - device name when sch
1311     */ 1220     */
1312     char service_name[12]; 1221     char service_name[12];
1313     get_device_service_name(servi 1222     get_device_service_name(servi
1314     /* What is the security level 1223     /* What is the security level
1315     *     - WIFI_PROV_SECURITY_ 1224     *     - WIFI_PROV_SECURITY_
1316     *     - WIFI_PROV_SECURITY_ 1225     *     - WIFI_PROV_SECURITY_
1317     *         using X25519 key 1226     *         using X25519 key
1318     *         for encryption/de 1227     *         for encryption/de
1319     */ 1228     */
1320     wifi_prov_security_t security 1229     wifi_prov_security_t security
1321                                1230
1322     /* Do we want a proof-of-poss 1231     /* Do we want a proof-of-poss
1323     *     - this should be a st 1232     *     - this should be a st
1324     *     - NULL if not used 1233     *     - NULL if not used
1325     */ 1234     */
1326     const char *pop = "abcd1234"; 1235     const char *pop = "abcd1234";
1327                                1236
1328     /* What is the service key (c 1237     /* What is the service key (c
1329     * This translates to : 1238     * This translates to :
1330     *     - Wi-Fi password when 1239     *     - Wi-Fi password when
1331     *     - simply ignored when 1240     *     - simply ignored when
1332     */ 1241     */
1333     const char *service_key = NUL 1242     const char *service_key = NUL
1334                                1243
1335                                1244
1336     /* This step is only useful w 1245     /* This step is only useful w
1337     * set a custom 128 bit UUID 1246     * set a custom 128 bit UUID
1338     * and will correspond to the 1247     * and will correspond to the
1339     * endpoints as GATT characte 1248     * endpoints as GATT characte
1340     * formed using the primary s 1249     * formed using the primary s
1341     * 12th and 13th bytes (assum 1250     * 12th and 13th bytes (assum
1342     * applications must identify 1251     * applications must identify
1343     * Description descriptor (0x 1252     * Description descriptor (0x
1344     * endpoint name of the chara 1253     * endpoint name of the chara
1345     uint8_t custom_service_uuid[] 1254     uint8_t custom_service_uuid[]
1346     /* LSB <----- 1255     /* LSB <-----
1347     * ----- 1256     * -----
1348     0xb4, 0xdf, 0x5a, 0x1c, 0 1257     0xb4, 0xdf, 0x5a, 0x1c, 0
1349     0xea, 0x4a, 0x82, 0x03, 0 1258     0xea, 0x4a, 0x82, 0x03, 0
1350 }; 1259 };

```

```

1351 wifi_prov_scheme_ble_set_serv 1260 wifi_prov_scheme_ble_set_serv
1352                               1261
1353                               1262
1354 /* An optional endpoint that 1263 /* An optional endpoint that
1355 * get some additional custom 1264 * get some additional custom
1356 * The endpoint name can be a 1265 * The endpoint name can be a
1357 * This call must be made bef 1266 * This call must be made bef
1358 */ 1267 */
1359 wifi_prov_mgr_endpoint_create 1268 wifi_prov_mgr_endpoint_create
1360 /* Start provisioning service 1269 /* Start provisioning service
1361 ESP_ERROR_CHECK(wifi_prov_mgr 1270 ESP_ERROR_CHECK(wifi_prov_mgr
1362                               1271
1363 /* The handler for the option 1272 /* The handler for the option
1364 * This call must be made aft 1273 * This call must be made aft
1365 * has already been created a 1274 * has already been created a
1366 */ 1275 */
1367 wifi_prov_mgr_endpoint_regist 1276 wifi_prov_mgr_endpoint_regist
1368                               1277
1369                               1278
1370 wifi_prov_print_qr(service_na 1279 wifi_prov_print_qr(service_na
1371 /* Uncomment the following to 1280 /* Uncomment the following to
1372 * the resources of the manag 1281 * the resources of the manag
1373 * by the default event loop 1282 * by the default event loop
1374 wifi_prov_mgr_wait(); 1283 wifi_prov_mgr_wait();
1375 wifi_prov_mgr_deinit(); 1284 wifi_prov_mgr_deinit();
1376 /* Print QR code for provisio 1285 /* Print QR code for provisio
1377                               1286
1378 //wifi_prov_print_qr(service_ 1287 //wifi_prov_print_qr(service_
1379 } else { 1288 } else {
1380     ESP_LOGI(TAG, "Already provi 1289 ESP_LOGI(TAG, "Already provi
1381                               1290
1382 /* We don't need the manager 1291 /* We don't need the manager
1383 * so let's release it's reso 1292 * so let's release it's reso
1384 wifi_prov_mgr_deinit(); 1293 wifi_prov_mgr_deinit();
1385 ESP_ERROR_CHECK( esp_wifi_set 1294 ESP_ERROR_CHECK( esp_wifi_set
1386 ESP_ERROR_CHECK( esp_wifi_sta 1295 ESP_ERROR_CHECK( esp_wifi_sta
1387                               1296
1388 } 1297 }
1389 /* ycc 030922 added to Wait for W 1298 /* ycc 030922 added to Wait for W
1390 int bits = xEventGroupWaitBits(wi 1299 int bits = xEventGroupWaitBits(wi
1391                               false, true, 1300                               false, true,
1392                               1301
1393 if (!(bits & CONNECTED_BIT)) { 1302 if (!(bits & CONNECTED_BIT)) {
1394     ESP_LOGE(TAG, "timeout bits=% 1303 ESP_LOGE(TAG, "timeout bits=%
1395     gpio_set_level(LED_RED, 0); 1304 gpio_set_level(LED_RED, 0);
1396 } 1305 }
1397                               1306
1398 //ycc 051722 get board mac addres 1307 //ycc 051722 get board mac addres
1399 //get mac address 1308 //get mac address
1400 uint8_t brd_mac[6]; 1309 uint8_t brd_mac[6];

```



```

1401 esp_wifi_get_mac(WIFI_IF_STA, brd 1310 esp_wifi_get_mac(WIFI_IF_STA, brd
1402 snprintf(macAddress, 13, "%02X%02 1311 snprintf(macAddress, 13, "%02X%02
1403 //printf("macAddress is %s, lengt 1312 //printf("macAddress is %s, lengt
1404 //ycc 3-15-22 initialize mqtt and 1313 //ycc 3-15-22 initialize mqtt and
1405 /* Return error status. */ 1314 /* Return error status. */
1406 int returnStatus = EXIT_SUCCESS; 1315 int returnStatus = EXIT_SUCCESS;
1407 /* Semaphore initialization flag. 1316 /* Semaphore initialization flag.
1408 bool bufferSemInitialized = false 1317 bool bufferSemInitialized = false
1409 bool mqttMutexInitialized = false 1318 bool mqttMutexInitialized = false
1410 1319
1411 mqttSessionEstablished = false; 1320 mqttSessionEstablished = false;
1412 1321
1413 /* Initialize semaphore for buffe 1322 /* Initialize semaphore for buffe
1414 if( osi_sem_new( &bufferSemaphore 1323 if( osi_sem_new( &bufferSemaphore
1415 { 1324 {
1416     LogError( ( "Failed to initia 1325     LogError( ( "Failed to initia
1417         ",errno=%s", 1326         ",errno=%s",
1418         strerror( errno ) 1327         strerror( errno )
1419 1328
1420     returnStatus = EXIT_FAILURE; 1329     returnStatus = EXIT_FAILURE;
1421 } 1330 }
1422 else 1331 else
1423 { 1332 {
1424     bufferSemInitialized = true; 1333     bufferSemInitialized = true;
1425 } 1334 }
1426 1335
1427 /* Initialize mutex for coreMQTT 1336 /* Initialize mutex for coreMQTT .
1428 if( pthread_mutex_init( &mqttMute 1337 if( pthread_mutex_init( &mqttMute
1429 { 1338 {
1430     LogError( ( "Failed to initia 1339     LogError( ( "Failed to initia
1431         ",errno=%s", 1340         ",errno=%s",
1432         strerror( errno ) 1341         strerror( errno )
1433 1342
1434     returnStatus = EXIT_FAILURE; 1343     returnStatus = EXIT_FAILURE;
1435 } 1344 }
1436 else 1345 else
1437 { 1346 {
1438     mqttMutexInitialized = true; 1347     mqttMutexInitialized = true;
1439 } 1348 }
1440 1349
1441 if( returnStatus == EXIT_SUCCESS 1350 if( returnStatus == EXIT_SUCCESS
1442 { 1351 {
1443     /* Initialize MQTT library. I 1352     /* Initialize MQTT library. I
1444     * done only once in this 1353     * done only once in this
1445     returnStatus = initializeMqtt 1354     returnStatus = initializeMqtt
1446 } 1355 }
1447 //aws_iot_task(&aws_iot_task); 1356 //aws_iot_task(&aws_iot_task);
1448 //xTaskCreatePinnedToCore(&aws_io 1357 //xTaskCreatePinnedToCore(&aws_io
1449 1358
1450 //ycc 070522 check nvs to see if 1359 //ycc 070522 check nvs to see if

```



```

1451 printf("Opening Non-Volatile Stor 1360 printf("Opening Non-Volatile Stor
1452 //nvs_handle_t fleet_prov_handle; 1361 //nvs_handle_t fleet_prov_handle;
1453 err = nvs_open("storage", NVS_REA 1362 err = nvs_open("storage", NVS_REA
1454 if (err != ESP_OK) { 1363 if (err != ESP_OK) {
1455     printf("Error (%s) opening NV 1364     printf("Error (%s) opening NV
1456 } else { 1365 } else {
1457     printf("Done\n"); 1366     printf("Done\n");
1458 1367
1459 // Read 1368 // Read
1460 printf("Reading fleet prov ce 1369 printf("Reading fleet prov ce
1461 //size_t private_key_len; 1370 //size_t private_key_len;
1462 err = nvs_get_str(fleet_prov_ 1371 err = nvs_get_str(fleet_prov_
1463 private_key = malloc(private_ 1372 private_key = malloc(private_
1464 err = nvs_get_str(fleet_prov_ 1373 err = nvs_get_str(fleet_prov_
1465 switch (err) { 1374 switch (err) {
1466     case ESP_OK: 1375     case ESP_OK:
1467         printf("Done\n"); 1376         printf("Done\n");
1468         //printf("private_key 1377         //printf("private_key
1469         nvsProvisionStatus = 1378         nvsProvisionStatus =
1470         break; 1379         break;
1471     case ESP_ERR_NVS_NOT_FOUN 1380     case ESP_ERR_NVS_NOT_FOUN
1472         printf("The value is 1381         printf("The value is
1473         nvsProvisionStatus = 1382         nvsProvisionStatus =
1474         break; 1383         break;
1475     default : 1384     default :
1476         printf("Error (%s) re 1385         printf("Error (%s) re
1477         nvsProvisionStatus = 1386         nvsProvisionStatus =
1478     } 1387     }
1479 //size_t certificate_pem_len; 1388 //size_t certificate_pem_len;
1480 err = nvs_get_str(fleet_prov_ 1389 err = nvs_get_str(fleet_prov_
1481 certificate_pem = malloc(cert 1390 certificate_pem = malloc(cert
1482 if(nvsProvisionStatus == true 1391 if(nvsProvisionStatus == true
1483 { 1392 {
1484     err = nvs_get_str(fleet_p 1393     err = nvs_get_str(fleet_p
1485     switch (err) { 1394     switch (err) {
1486         case ESP_OK: 1395         case ESP_OK:
1487             printf("Done\n"); 1396             printf("Done\n");
1488             //printf("certifi 1397             //printf("certifi
1489             nvsProvisionStatu 1398             nvsProvisionStatu
1490             break; 1399             break;
1491         case ESP_ERR_NVS_NOT_ 1400         case ESP_ERR_NVS_NOT_
1492             printf("The value 1401             printf("The value
1493             nvsProvisionStatu 1402             nvsProvisionStatu
1494             break; 1403             break;
1495         default : 1404         default :
1496             printf("Error (%s 1405             printf("Error (%s
1497             nvsProvisionStatu 1406             nvsProvisionStatu
1498         } 1407         }
1499     } 1408     }
1500 } 1409 }

```

```

1501 //ycc 090222                                1410 //ycc 090222
1502 if (nvsProvisionStatus == true) { 1411 if (nvsProvisionStatus == true) {
1503     //nvs has credentials, go ahe 1412     //nvs has credentials, go ahe
1504     typedef struct Data_t 1413     typedef struct Data_t
1505     { 1414     {
1506         uint32_t uData; 1415         uint32_t uData;
1507         char **id; 1416         char **id;
1508     } arg_params_t; 1417     } arg_params_t;
1509 1418
1510     arg_params_t data1 = {0, NULL 1419     arg_params_t data1 = {0, NULL
1511     uint32_t free_heap_size=0, mi 1420     uint32_t free_heap_size=0, mi
1512     free_heap_size = esp_get_free 1421     free_heap_size = esp_get_free
1513     min_free_heap_size = esp_get_ 1422     min_free_heap_size = esp_get_
1514     printf("\n free heap size = % 1423     printf("\n free heap size = %
1515 1424
1516     xTaskCreatePinnedToCore((Task 1425     xTaskCreatePinnedToCore((Task
1517     aws_iot_task(&aws_iot_task); 1426     aws_iot_task(&aws_iot_task);
1518     /* Disconnect from broker and 1427     /* Disconnect from broker and
1519 } 1428 }
1520 else 1429 else
1521 { 1430 {
1522     //ycc 062222 for fleet provis 1431     //ycc 062222 for fleet provis
1523     if( mqttSessionEstablished != 1432     if( mqttSessionEstablished !=
1524         int ret = establishConnec 1433         int ret = establishConnec
1525         printf("establish connect 1434         printf("establish connect
1526         //printf( 1435         //printf(
1527     } 1436     }
1528     if( mqttSessionEstablished == 1437     if( mqttSessionEstablished ==
1529 1438
1530         const char *topic = " 1439         const char *topic = "
1531         OtaMqttStatus_t mqttR 1440         OtaMqttStatus_t mqttR
1532         printf("mqttreturn 0= 1441         printf("mqttreturn 0=
1533         const char *topic1 = 1442         const char *topic1 =
1534         mqttReturn = mqttSubs 1443         mqttReturn = mqttSubs
1535         printf("mqttreturn 1= 1444         printf("mqttreturn 1=
1536         const char *topic2 = 1445         const char *topic2 =
1537         mqttReturn = mqttSubs 1446         mqttReturn = mqttSubs
1538         printf("mqttreturn 2= 1447         printf("mqttreturn 2=
1539         const char *topic3 = 1448         const char *topic3 =
1540         mqttReturn = mqttSubs 1449         mqttReturn = mqttSubs
1541         printf("mqttreturn 3= 1450         printf("mqttreturn 3=
1542         const char *topic4 = 1451         const char *topic4 =
1543         mqttReturn = mqttPubl 1452         mqttReturn = mqttPubl
1544         printf("mqttreturn 4= 1453         printf("mqttreturn 4=
1545         //vTaskDelay(300); 1454         //vTaskDelay(300);
1546         // registering with t 1455         // registering with t
1547         const char *topic5 = 1456         const char *topic5 =
1548         printf("lengthh of re 1457         printf("lengthh of re
1549         mqttReturn = mqttPubl 1458         mqttReturn = mqttPubl
1550         printf("mqttreturn 5= 1459         printf("mqttreturn 5=

```

subscribe_publish_sample.c x 2

1551	nvsProvisionStatus =	1460	nvsProvisionStatus =
1552		1461	
1553		1462	
1554	MQTTStatus_t mqttStat	1463	MQTTStatus_t mqttStat
1555	{	1464	{
1556		1465	
1557	/* Acquire the mq	1466	/* Acquire the mq
1558	if(pthread_mutex	1467	if(pthread_mutex
1559	{	1468	{
1560	/* Loop to re	1469	/* Loop to re
1561	mqttStatus =	1470	mqttStatus =
1562		1471	
1563	pthread_mutex	1472	pthread_mutex
1564	}	1473	}
1565	else	1474	else
1566	{	1475	{
1567	LogError(("	1476	LogError(("
1568	"	1477	"
1569	s	1478	s
1570	}	1479	}
1571		1480	
1572	if(mqttStatus ==	1481	if(mqttStatus ==
1573	{	1482	{
1574	/* Get OTA st	1483	/* Get OTA st
1575	/* Delay if m	1484	/* Delay if m
1576	if(MQTT_PROC	1485	if(MQTT_PROC
1577	{	1486	{
1578	vTaskDela	1487	vTaskDela
1579	}	1488	}
1580	printf("Delay	1489	printf("Delay
1581	}	1490	}
1582	else	1491	else
1583	{	1492	{
1584	LogError(("	1493	LogError(("
1585	M	1494	M
1586		1495	
1587	/* Disconnect	1496	/* Disconnect
1588	disconnect();	1497	disconnect();
1589		1498	
1590		1499	
1591	}	1500	}
1592	// Read	1501	// Read
1593	printf("Reading f	1502	printf("Reading f
1594		1503	
1595	err = nvs_get_str	1504	err = nvs_get_str
1596	private_key = mal	1505	private_key = mal
1597	err = nvs_get_str	1506	err = nvs_get_str
1598	switch (err) {	1507	switch (err) {
1599	case ESP_OK:	1508	case ESP_OK:
1600	printf("D	1509	printf("D

```

1601         //printf( 1510         //printf(
1602         break; 1511         break;
1603         case ESP_ERR_ 1512         case ESP_ERR_
1604         printf("T 1513         printf("T
1605         break; 1514         break;
1606         default : 1515         default :
1607         printf("E 1516         printf("E
1608     } 1517     }
1609     1518
1610     err = nvs_get_str 1519     err = nvs_get_str
1611     certificate_pem = 1520     certificate_pem =
1612     err = nvs_get_str 1521     err = nvs_get_str
1613     switch (err) { 1522     switch (err) {
1614         case ESP_OK: 1523         case ESP_OK:
1615         printf("D 1524         printf("D
1616         //printf( 1525         //printf(
1617         break; 1526         break;
1618         case ESP_ERR_ 1527         case ESP_ERR_
1619         printf("T 1528         printf("T
1620         break; 1529         break;
1621         default : 1530         default :
1622         printf("E 1531         printf("E
1623     } 1532     }
1624     disconnect(); 1533     disconnect();
1625     printf("reconnect 1534     printf("reconnect
1626     //ycc 062222 for 1535     //ycc 062222 for
1627     if( mqttSessionEs 1536     if( mqttSessionEs
1628         int ret = est 1537         int ret = est
1629         printf("estab 1538         printf("estab
1630         //printf( 1539         //printf(
1631     } 1540     }
1632     1541
1633     } 1542     }
1634     } 1543     }
1635     1544
1636     } 1545     }
1637 } 1546 }
1638 disconnect(); 1547 disconnect();
1639 1548
1640 1549
1641 1550
1642 if( bufferSemInitialized == true 1551 if( bufferSemInitialized == true
1643 { 1552 {
1644     /* Cleanup semaphore created 1553     /* Cleanup semaphore created
1645     if( osi_sem_free( &bufferSema 1554     if( osi_sem_free( &bufferSema
1646     { 1555     {
1647         LogError( ( "Failed to de 1556         LogError( ( "Failed to de
1648         ",errno=%s", 1557         ",errno=%s",
1649         strerror( err 1558         strerror( err
1650 1559

```

subscribe_publish_sample.c x 2

1651	returnStatus = EXIT_FAILU	1560	returnStatus = EXIT_FAILU
1652	}	1561	}
1653	}	1562	}
1654		1563	
1655	if(mqttMutexInitialized == true	1564	if(mqttMutexInitialized == true
1656	{	1565	{
1657	/* Cleanup mutex created for :	1566	/* Cleanup mutex created for :
1658	if(pthread_mutex_destroy(&m	1567	if(pthread_mutex_destroy(&m
1659	{	1568	{
1660	LogError(("Failed to de	1569	LogError(("Failed to de
1661	",errno=%s",	1570	",errno=%s",
1662	strerror(err	1571	strerror(err
1663		1572	
1664	returnStatus = EXIT_FAILU	1573	returnStatus = EXIT_FAILU
1665	}	1574	}
1666	}	1575	}
1667		1576	
1668		1577	