```
 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
26clude <stdio.h>
27clude <stdlib.h>
28clude <ctype.h>
29clude <unistd.h>
30clude <limits.h>
31clude <string.h>
32clude <time.h>
33clude <sys/time.h>
34clude "freertos/FreeRTOS.h"
35clude "freertos/task.h"
36clude "freertos/event_groups.h"
37clude "freertos/queue.h"
38clude "math.h"
39clude "esp_system.h"
40clude "esp_wifi.h"
41clude "esp_event_loop.h"
42clude "esp_log.h"
43clude "esp_vfs_fat.h"
44clude "driver/sdmmc_host.h"
45clude "driver/timer.h"
46clude "driver/gpio.h"
47clude "driver/adc.h"
48clude "esp_adc_cal.h"
49clude "esp_sntp.h"
50clude "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
26clude <stdio.h>
27clude <stdlib.h>
28clude <ctype.h>
29clude <unistd.h>
30clude <limits.h>
31clude <string.h>
32clude <time.h>
33clude <sys/time.h>
34clude "freertos/FreeRTOS.h"
35clude "freertos/task.h"
36clude "freertos/event_groups.h"
37clude "freertos/queue.h"
38clude "math.h"
39clude "esp_system.h"
40clude "esp_wifi.h"
41clude "esp_event_loop.h"
42clude "esp_log.h"
43clude "esp_vfs_fat.h"
44clude "driver/sdmmc_host.h"
45clude "driver/timer.h"
46clude "driver/gpio.h"
47clude "driver/adc.h"
48clude "esp_adc_cal.h"
49clude "esp_sntp.h"
50clude "nvs.h"
```

```
clude "nvs_flash.h"

clude <time.h>
clude "esp_sntp.h"

clude "aws_iot_config.h"
clude "aws_iot_log.h"
clude "aws_iot_version.h"
clude "aws_iot_mqtt_client_interfa
clude "cJSON.h"

clude "demo_config.h"

def CONFIG_IDF_TARGET_ESP32
fine CHIP_NAME "ESP32"
dif

def CONFIG_IDF_TARGET_ESP32S2BETA
fine CHIP_NAME "ESP32-S2 Beta"
dif

72122 ycc added the following bloc
clude <wifi_provisioning/manager.h
clude <wifi_provisioning/scheme_bl
clude "qrcode.h"

clude "app_priv.h"   //for app_dri
clude "board_esp32_devkitc.h"
cc 3-13-22 added include files
OpenSSL sockets transport implemen
clude "tls_freertos.h"

Clock for timer. */
clude "clock.h"

pthread include. */
clude <pthread.h>
include "semaphore.h"
clude <unistd.h>
clude "freertos/FreeRTOS.h"
clude "freertos/task.h"
clude "esp_pthread.h"

MQTT include. */
clude "core_mqtt.h"
clude "mqtt_subscription_manager.h

nclude backoff algorithm header fo
clude "backoff_algorithm.h"
```


```
clude "nvs_flash.h"

clude <time.h>
clude "esp_sntp.h"

clude "aws_iot_config.h"
clude "aws_iot_log.h"
clude "aws_iot_version.h"
clude "aws_iot_mqtt_client_interfa
clude "cJSON.h"

clude "demo_config.h"

def CONFIG_IDF_TARGET_ESP32
fine CHIP_NAME "ESP32"
dif

def CONFIG_IDF_TARGET_ESP32S2BETA
fine CHIP_NAME "ESP32-S2 Beta"
dif

72122 ycc added the following bloc
clude <wifi_provisioning/manager.h
clude <wifi_provisioning/scheme_bl
clude "qrcode.h"

clude "app_priv.h"   //for app_dri
clude "board_esp32_devkitc.h"
cc 3-13-22 added include files
OpenSSL sockets transport implemen
clude "tls_freertos.h"

Clock for timer. */
clude "clock.h"

pthread include. */
clude <pthread.h>
include "semaphore.h"
clude <unistd.h>
clude "freertos/FreeRTOS.h"
clude "freertos/task.h"
clude "esp_pthread.h"

MQTT include. */
clude "core_mqtt.h"
clude "mqtt_subscription_manager.h

nclude backoff algorithm header fo
clude "backoff_algorithm.h"
```

```
OTA Library include. */
clude "ota.h"
clude "ota_config.h"

OTA Library Interface include. */
clude "ota_os_freertos.h"
clude "ota_mqtt_interface.h"
clude "ota_pal.h"

deep sleep */
clude "soc/soc_caps.h"
clude "esp_sleep.h"
clude "driver/adc.h"
clude "driver/rtc_io.h"
clude "soc/rtc.h"
clude "esp32/ulp.h"
clude "soc/sens_periph.h"
clude "driver/touch_pad.h"



fine DEFAULT_WAKEUP_LEVEL    ESP_G
Include firmware version struct de
clude "ota_appversion32.h"

clude "demo_header.h"
cc 3-13-22 end
tic const char *TAG = "subpub";
tic void initialize_sntp(void);
The examples use simple WiFi confi
'make menuconfig'.

If you'd rather not, just change t
the config you want - ie #define E

fine CONFIG_ESP_MAXIMUM_RETRY   5
fine EXAMPLE_WIFI_SSID CONFIG_WIFI
fine EXAMPLE_WIFI_PASS CONFIG_WIFI
fine EXAMPLE_ESP_MAXIMUM_RETRY  CO

The event group allows multiple bi
- we are connected to the AP with
- we failed to connect after the m
fine WIFI_CONNECTED_BIT BIT0
fine WIFI_FAIL_BIT      BIT1


fine TIMER_DIVIDER         (80)  /
fine TIMER_SCALE           (TIMER_
fine DEFAULT_VREF    1100        /
```

```
OTA Library include. */
clude "ota.h"
clude "ota_config.h"

OTA Library Interface include. */
clude "ota_os_freertos.h"
clude "ota_mqtt_interface.h"
clude "ota_pal.h"

Include firmware version struct de
clude "ota_appversion32.h"

clude "demo_header.h"
cc 3-13-22 end
tic const char *TAG = "subpub";
tic void initialize_sntp(void);
The examples use simple WiFi confi
'make menuconfig'.

If you'd rather not, just change t
the config you want - ie #define E

fine CONFIG_ESP_MAXIMUM_RETRY   5
fine EXAMPLE_WIFI_SSID CONFIG_WIFI
fine EXAMPLE_WIFI_PASS CONFIG_WIFI
fine EXAMPLE_ESP_MAXIMUM_RETRY  CO

The event group allows multiple bi
- we are connected to the AP with
- we failed to connect after the m
fine WIFI_CONNECTED_BIT BIT0
fine WIFI_FAIL_BIT      BIT1


fine TIMER_DIVIDER         (80)  /
fine TIMER_SCALE           (TIMER_
fine DEFAULT_VREF    1100        /
```

```
fine NO_OF_SAMPLES   3          //

GPIO definition, in addtion GPIO 0
fine LOPlus  34
fine LOMinus 35
fine SDN     23
fine FR      21
fine DC      22

fine LED_RED     GPIO_NUM_16
fine LED_GREEN   GPIO_NUM_17
fine LED_BLUE    GPIO_NUM_18

fine DEEPSLEEP   GPIO_NUM_32


fine ECG_IDLE 0               // Id
fine ECG_ACQUIRING 1          // EC
fine ECG_RECORDING 2          // EC
fine ECG_SENDING_MQTT 3       // EC
fine ECG_FINISH 4             // da
fine ECG_ERROR_WIFI 5         // er
fine ECG_ERROR_MQTT 6         // er
fine ECG_OTA_UPDATE 7         // EC
fine ECG_SSID_RESET 8
fine ECG_ACQCOUNT 3000       //  co
fine ECG_RECCOUNT 9000      //  cou
fine ECG_MQTTCOUNT 2000      //  c

fine JOB_CHECK_STATE_NOT_CHECKED
fine JOB_CHECK_STATE_CHECKED_NO_UP

fine MQTT_PROCESS_LOOP_TIMEOUT_MS

deep sleep defines */
fine DEFAULT_WAKEUP_LEVEL    ESP_G
fine TOUCH_THRESH_NO_USE 0

edef struct {
 int timer_group;
 int timer_idx;
 int alarm_interval;
 bool auto_reload;
xample_timer_info_t;


@brief A sample structure to pass
```

```
fine NO_OF_SAMPLES   3          //

GPIO definition, in addtion GPIO 0
fine LOPlus  34
fine LOMinus 35
fine SDN     23
fine FR      21
fine DC      22

fine LED_RED     GPIO_NUM_16
fine LED_GREEN   GPIO_NUM_17
fine LED_BLUE    GPIO_NUM_18


fine ECG_IDLE 0               // Id
fine ECG_ACQUIRING 1          // EC
fine ECG_RECORDING 2          // EC
fine ECG_SENDING_MQTT 3       // EC
fine ECG_FINISH 4             // da
fine ECG_ERROR_WIFI 5         // er
fine ECG_ERROR_MQTT 6         // er
fine ECG_OTA_UPDATE 7         // EC
fine ECG_SSID_RESET 8
fine ECG_ACQCOUNT 3000       //  co
fine ECG_RECCOUNT 9000      //  cou
fine ECG_MQTTCOUNT 2000      //  c

fine JOB_CHECK_STATE_NOT_CHECKED
fine JOB_CHECK_STATE_CHECKED_NO_UP

fine MQTT_PROCESS_LOOP_TIMEOUT_MS


edef struct {
 int timer_group;
 int timer_idx;
 int alarm_interval;
 bool auto_reload;
xample_timer_info_t;


@brief A sample structure to pass
```

```
edef struct {
 example_timer_info_t info;
 uint64_t timer_counter_value;
xample_timer_event_t;

tic xQueueHandle s_timer_queue;

tic esp_adc_cal_characteristics_t

tic const adc_channel_t channel =
tic const adc_bits_width_t width =

tic const adc_atten_t atten = ADC_
tic const adc_unit_t unit = ADC_UN

tic short jobCheckState;

igned char blinkingPattern[9][10]
                           {0,0,0,0,
                           {0,1,0,1,
                           {1,1,1,1,
                           {0,0,0,0,
                           {1,0,1,0,
                           {0,0,1,1,
                           {1,1,0,0,
                           {1,0,1,0,
                           {1,0,1,0,


FreeRTOS event group to signal whe
tic EventGroupHandle_t wifi_event_
rt ecgState;
r macAddress[13];
rt nvsProvisionStatus = false;
r *private_key = NULL;
e_t private_key_len = 0;
r *certificate_pem = NULL;
e_t certificate_pem_len = 0;
ern int    aws_iot_demo_main();

The event group allows multiple bi
but we only care about one event -
to the AP with an IP? */
st int CONNECTED_BIT = BIT0;
st int WIFI_CONNECTED_EVENT = BIT0

72221 ycc added the following bloc
fine PROV_QR_VERSION         "v1"
fine PROV_TRANSPORT_SOFTAP   "soft
fine PROV_TRANSPORT_BLE      "ble"
```

```
edef struct {
 example_timer_info_t info;
 uint64_t timer_counter_value;
xample_timer_event_t;

tic xQueueHandle s_timer_queue;

tic esp_adc_cal_characteristics_t

tic const adc_channel_t channel =
tic const adc_bits_width_t width =

tic const adc_atten_t atten = ADC_
tic const adc_unit_t unit = ADC_UN

tic short jobCheckState;

igned char blinkingPattern[9][10]
                           {0,0,0,0,
                           {0,1,0,1,
                           {1,1,1,1,
                           {0,0,0,0,
                           {1,0,1,0,
                           {0,0,1,1,
                           {1,1,0,0,
                           {1,0,1,0,
                           {1,0,1,0,


FreeRTOS event group to signal whe
tic EventGroupHandle_t wifi_event_
rt ecgState;
r macAddress[13];
rt nvsProvisionStatus = false;
r *private_key = NULL;
e_t private_key_len = 0;
r *certificate_pem = NULL;
e_t certificate_pem_len = 0;
ern int    aws_iot_demo_main();

The event group allows multiple bi
but we only care about one event -
to the AP with an IP? */
st int CONNECTED_BIT = BIT0;
st int WIFI_CONNECTED_EVENT = BIT0

72221 ycc added the following bloc
fine PROV_QR_VERSION         "v1"
fine PROV_TRANSPORT_SOFTAP   "soft
fine PROV_TRANSPORT_BLE      "ble"
```

```
fine QRCODE_BASE_URL         "http

CA Root certificate, device ("Thin
("Thing") key.

Example can be configured one of t

"Embedded Certs" are loaded from f

"Filesystem Certs" are loaded from

See example README for more detail

 defined(CONFIG_EXAMPLE_EMBEDDED_C

ern const uint8_t aws_root_ca_pem_
ern const uint8_t aws_root_ca_pem_
ern const uint8_t certificate_pem_
ern const uint8_t certificate_pem_
ern const uint8_t private_pem_key_
ern const uint8_t private_pem_key_

if defined(CONFIG_EXAMPLE_FILESYST

tic const char * DEVICE_CERTIFICAT
tic const char * DEVICE_PRIVATE_KE
tic const char * ROOT_CA_PATH = CO

se
ror "Invalid method for loading ce
dif

fine MQTT_RECV_POLLING_TIMEOUT_MS


@brief Default MQTT HOST URL is pu

r HostAddress[255] = AWS_IOT_MQTT_


@brief Default MQTT port is pulled

t32_t port = AWS_IOT_MQTT_PORT;

cc 3-13-22 added externs

edef xSemaphoreHandle osi_sem_t;


@brief Network connection context
```

```
fine QRCODE_BASE_URL         "http

CA Root certificate, device ("Thin
("Thing") key.

Example can be configured one of t

"Embedded Certs" are loaded from f

"Filesystem Certs" are loaded from

See example README for more detail

 defined(CONFIG_EXAMPLE_EMBEDDED_C

ern const uint8_t aws_root_ca_pem_
ern const uint8_t aws_root_ca_pem_
ern const uint8_t certificate_pem_
ern const uint8_t certificate_pem_
ern const uint8_t private_pem_key_
ern const uint8_t private_pem_key_

if defined(CONFIG_EXAMPLE_FILESYST

tic const char * DEVICE_CERTIFICAT
tic const char * DEVICE_PRIVATE_KE
tic const char * ROOT_CA_PATH = CO

se
ror "Invalid method for loading ce
dif

fine MQTT_RECV_POLLING_TIMEOUT_MS


@brief Default MQTT HOST URL is pu

r HostAddress[255] = AWS_IOT_MQTT_


@brief Default MQTT port is pulled

t32_t port = AWS_IOT_MQTT_PORT;

cc 3-13-22 added externs

edef xSemaphoreHandle osi_sem_t;


@brief Network connection context
```

```
301
302ern NetworkContext_t networkContex
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
316l 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 l
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( MQTTContex
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 networkContex
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
297l 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 l
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( MQTTContex
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
352ern void setOtaInterfaces();
353
354ern void otaAppCallback();
355
356ern void * otaThread();
357
358ern void provisionEventCallback();
359cc 3-13-22 end
360eep  sleep
361tic void calibrate_touch_pad(touch
362
363at w0=0.0, w1=0.0, w2=0.0, w3=0.0,
364at bpfX, bpfX1;
365igned short aData;
366 peakCounter=0;
367 detectedBeat = 0;
368at bpfS0=0, bpfS1=0, bpfS2=0, bpfS
369at mVariance=0, mAvg=100, m0=100,
370at pVariance=300000, pAvg=0, p0=10
371at variance=650;
372at nAvg=0, n0=0, n1=0, n2=0, n3=0,
373 hb0=0, hb1=0;
374at heartRate=0;
375at noiseFloor = 0;
376
377
378igned short *dataBuffer;
379
380d time_sync_notification_cb(struct
381
382 ESP_LOGI(TAG, "Notification of a
383
384
385tic 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
391def CONFIG_SNTP_TIME_SYNC_METHOD_S
392 sntp_set_sync_mode(SNTP_SYNC_MODE
393dif
394 sntp_init();
395
396
397
398tic void calibrate_touch_pad(touch
399
400 int avg = 0;
```

```
332
333ern void setOtaInterfaces();
334
335ern void otaAppCallback();
336
337ern void * otaThread();
338
339ern void provisionEventCallback();
340cc 3-13-22 end
341
342at w0=0.0, w1=0.0, w2=0.0, w3=0.0,
343at bpfX, bpfX1;
344igned short aData;
345 peakCounter=0;
346 detectedBeat = 0;
347at bpfS0=0, bpfS1=0, bpfS2=0, bpfS
348at mVariance=0, mAvg=100, m0=100,
349at pVariance=300000, pAvg=0, p0=10
350at variance=650;
351at nAvg=0, n0=0, n1=0, n2=0, n3=0,
352 hb0=0, hb1=0;
353at heartRate=0;
354at noiseFloor = 0;
355
356
357igned short *dataBuffer;
358
359d time_sync_notification_cb(struct
360
361 ESP_LOGI(TAG, "Notification of a
362
363
364tic 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
370def CONFIG_SNTP_TIME_SYNC_METHOD_S
371 sntp_set_sync_mode(SNTP_SYNC_MODE
372dif
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 }
418
419
420tic 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
436tic 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
448tic bool IRAM_ATTR timer_group_isr
449
450 BaseType_t high_task_awoken = pdF
```

```
376tic 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
392tic 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
404tic bool IRAM_ATTR timer_group_isr
405
406 BaseType_t high_task_awoken = pdF
```

```
example_timer_info_t *info = (exa
uint64_t timer_counter_value = ti
/* Prepare basic event data that
example_timer_event_t evt = {
    .info.timer_group = info->tim
    .info.timer_idx = info->timer
    .info.auto_reload = info->aut
    .info.alarm_interval = info->
    .timer_counter_value = timer_
};
if (!info->auto_reload) {
    timer_counter_value += info->
    timer_group_set_alarm_value_i
}
/* Now just send the event data b
xQueueSendFromISR(s_timer_queue,
return high_task_awoken == pdTRUE
@brief Initialize selected timer o
@param group Timer Group number, i
@param timer timer ID, index from
@param auto_reload whether auto-re
@param timer_interval_sec interval
tic void ad_tg_timer_init(int grou
/* Select and initialize basic pa
timer_config_t config = {
    .divider = TIMER_DIVIDER,
    .counter_dir = TIMER_COUNT_UP
    .counter_en = TIMER_PAUSE,
    .alarm_en = TIMER_ALARM_EN,
    .auto_reload = auto_reload,
}; // default clock source is APB
timer_init(group, timer, &config)
/* Timer's counter will initially
   Also, if auto_reload is set, t
timer_set_counter_value(group, ti
```

```
example_timer_info_t *info = (exa
uint64_t timer_counter_value = ti
/* Prepare basic event data that
example_timer_event_t evt = {
    .info.timer_group = info->tim
    .info.timer_idx = info->timer
    .info.auto_reload = info->aut
    .info.alarm_interval = info->
    .timer_counter_value = timer_
};
if (!info->auto_reload) {
    timer_counter_value += info->
    timer_group_set_alarm_value_i
}
/* Now just send the event data b
xQueueSendFromISR(s_timer_queue,
return high_task_awoken == pdTRUE
@brief Initialize selected timer o
@param group Timer Group number, i
@param timer timer ID, index from
@param auto_reload whether auto-re
@param timer_interval_sec interval
tic void ad_tg_timer_init(int grou
/* Select and initialize basic pa
timer_config_t config = {
    .divider = TIMER_DIVIDER,
    .counter_dir = TIMER_COUNT_UP
    .counter_en = TIMER_PAUSE,
    .alarm_en = TIMER_ALARM_EN,
    .auto_reload = auto_reload,
}; // default clock source is APB
timer_init(group, timer, &config)
/* Timer's counter will initially
   Also, if auto_reload is set, t
timer_set_counter_value(group, ti
```

```
/* Configure the alarm value and
timer_set_alarm_value(group, time
timer_enable_intr(group, timer);

example_timer_info_t *timer_info
timer_info->timer_group = group;
timer_info->timer_idx = timer;
timer_info->auto_reload = auto_re
timer_info->alarm_interval = time
timer_isr_callback_add(group, tim

timer_start(group, timer);


tic int s_retry_num = 0;
tic void event_handler(void* arg,
                            int32


if (event_base == WIFI_PROV_EVENT
    switch (event_id) {
        case WIFI_PROV_START:
            ESP_LOGI(TAG, "Provis
            break;
        case WIFI_PROV_CRED_RECV:
            wifi_sta_config_t *wi
            ESP_LOGI(TAG, "Receiv
                     "\n\tSSID
                     (const char
                     (const char
            break;
        }
        case WIFI_PROV_CRED_FAIL:
            wifi_prov_sta_fail_re
            ESP_LOGE(TAG, "Provis
                     "\n\tPlease
                     (*reason ==
                     "Wi-Fi stati
            break;
        }
        case WIFI_PROV_CRED_SUCCE
            ESP_LOGI(TAG, "Provis
            break;
        case WIFI_PROV_END:
            /* De-initialize mana
            wifi_prov_mgr_deinit(
            break;
        default:
            break;
    }
```

```
/* Configure the alarm value and
timer_set_alarm_value(group, time
timer_enable_intr(group, timer);

example_timer_info_t *timer_info
timer_info->timer_group = group;
timer_info->timer_idx = timer;
timer_info->auto_reload = auto_re
timer_info->alarm_interval = time
timer_isr_callback_add(group, tim

timer_start(group, timer);


tic int s_retry_num = 0;
tic void event_handler(void* arg,
                            int32


if (event_base == WIFI_PROV_EVENT
    switch (event_id) {
        case WIFI_PROV_START:
            ESP_LOGI(TAG, "Provis
            break;
        case WIFI_PROV_CRED_RECV:
            wifi_sta_config_t *wi
            ESP_LOGI(TAG, "Receiv
                     "\n\tSSID
                     (const char
                     (const char
            break;
        }
        case WIFI_PROV_CRED_FAIL:
            wifi_prov_sta_fail_re
            ESP_LOGE(TAG, "Provis
                     "\n\tPlease
                     (*reason ==
                     "Wi-Fi stati
            break;
        }
        case WIFI_PROV_CRED_SUCCE
            ESP_LOGI(TAG, "Provis
            break;
        case WIFI_PROV_END:
            /* De-initialize mana
            wifi_prov_mgr_deinit(
            break;
        default:
            break;
    }
```

```
} else if (event_base == WIFI_EVE
    esp_wifi_connect();
    ESP_LOGI(TAG, "Received WIFI_
} else if (event_base == IP_EVENT
    ip_event_got_ip_t* event = (i
    ESP_LOGI(TAG, "Connected with
    s_retry_num = 0;
    gpio_set_level(LED_RED, 1);
    ecgState = ECG_IDLE;
    /* Signal main application to
    xEventGroupSetBits(wifi_event
} else if (event_base == WIFI_EVE
            if (s_retry_num < EXA
                esp_wifi_connect(
                xEventGroupClearB
                s_retry_num++;
                gpio_set_level(LE
                ecgState = ECG_ER
                ESP_LOGI(TAG, "re
            } else {
                xEventGroupSetBit
            }
            ESP_LOGI(TAG,"connect
        }


tic void get_device_service_name(c

uint8_t eth_mac[6];

const char *ssid_prefix = "PROV_"
esp_wifi_get_mac(WIFI_IF_STA, eth
snprintf(service_name, max, "%s%0
         ssid_prefix, eth_mac[3],


Handler for the optional provision
The data format can be chosen by a
Applications can choose to use oth

_err_t custom_prov_data_handler(ui


if (inbuf) {
    ESP_LOGI(TAG, "Received data:
}
char response[] = "SUCCESS";
*outbuf = (uint8_t *)strdup(respo
if (*outbuf == NULL) {
    ESP_LOGE(TAG, "System out of
```

```
} else if (event_base == WIFI_EVE
    esp_wifi_connect();
    ESP_LOGI(TAG, "Received WIFI_
} else if (event_base == IP_EVENT
    ip_event_got_ip_t* event = (i
    ESP_LOGI(TAG, "Connected with
    s_retry_num = 0;
    gpio_set_level(LED_RED, 1);
    ecgState = ECG_IDLE;
    /* Signal main application to
    xEventGroupSetBits(wifi_event
} else if (event_base == WIFI_EVE
            if (s_retry_num < EXA
                esp_wifi_connect(
                xEventGroupClearB
                s_retry_num++;
                gpio_set_level(LE
                ecgState = ECG_ER
                ESP_LOGI(TAG, "re
            } else {
                xEventGroupSetBit
            }
            ESP_LOGI(TAG,"connect
        }


tic void get_device_service_name(c

uint8_t eth_mac[6];

const char *ssid_prefix = "PROV_"
esp_wifi_get_mac(WIFI_IF_STA, eth
snprintf(service_name, max, "%s%0
         ssid_prefix, eth_mac[3],


Handler for the optional provision
The data format can be chosen by a
Applications can choose to use oth

_err_t custom_prov_data_handler(ui


if (inbuf) {
    ESP_LOGI(TAG, "Received data:
}
char response[] = "SUCCESS";
*outbuf = (uint8_t *)strdup(respo
if (*outbuf == NULL) {
    ESP_LOGE(TAG, "System out of
```

```
    return ESP_ERR_NO_MEM;
}
*outlen = strlen(response) + 1; /

return ESP_OK;


tic void wifi_prov_print_qr(const

if (!name || !transport) {
    ESP_LOGW(TAG, "Cannot generat
    return;
}
char payload[150] = {0};
if (pop) {
    snprintf(payload, sizeof(payl
                ",\"pop\":\"%s\",
                PROV_QR_VERSION,
} else {
    snprintf(payload, sizeof(payl
                ",\"transport\":\
                PROV_QR_VERSION,
}
def CONFIG_EXAMPLE_PROV_SHOW_QR
 ESP_LOGI(TAG, "Scan this QR code
 esp_qrcode_config_t cfg = ESP_QRC
 esp_qrcode_generate(&cfg, payload
dif // CONFIG_APP_WIFI_PROV_SHOW_Q
 ESP_LOGI(TAG, "If QR code is not


d iot_subscribe_callback_handler(A
                                 I
 ESP_LOGI(TAG, "Subscribe Callback



d disconnectCallbackHandler(AWS_Io
 ESP_LOGW(TAG, "MQTT Disconnect");
 IoT_Error_t rc = FAILURE;

 if(NULL == pClient) {
    return;
 }

 if(aws_iot_is_autoreconnect_enabl
    ESP_LOGI(TAG, "Auto Reconnect
 } else {
    ESP_LOGW(TAG, "Auto Reconnect
    rc = aws_iot_mqtt_attempt_rec
```

```
    return ESP_ERR_NO_MEM;
}
*outlen = strlen(response) + 1; /

return ESP_OK;


tic void wifi_prov_print_qr(const

if (!name || !transport) {
    ESP_LOGW(TAG, "Cannot generat
    return;
}
char payload[150] = {0};
if (pop) {
    snprintf(payload, sizeof(payl
                ",\"pop\":\"%s\",
                PROV_QR_VERSION,
} else {
    snprintf(payload, sizeof(payl
                ",\"transport\":\
                PROV_QR_VERSION,
}
def CONFIG_EXAMPLE_PROV_SHOW_QR
 ESP_LOGI(TAG, "Scan this QR code
 esp_qrcode_config_t cfg = ESP_QRC
 esp_qrcode_generate(&cfg, payload
dif // CONFIG_APP_WIFI_PROV_SHOW_Q
 ESP_LOGI(TAG, "If QR code is not


d iot_subscribe_callback_handler(A
                                 I
 ESP_LOGI(TAG, "Subscribe Callback



d disconnectCallbackHandler(AWS_Io
 ESP_LOGW(TAG, "MQTT Disconnect");
 IoT_Error_t rc = FAILURE;

 if(NULL == pClient) {
    return;
 }

 if(aws_iot_is_autoreconnect_enabl
    ESP_LOGI(TAG, "Auto Reconnect
 } else {
    ESP_LOGW(TAG, "Auto Reconnect
    rc = aws_iot_mqtt_attempt_rec
```

```
651             if(NETWORK_RECONNECTE
652         ESP_LOGW(TAG, "Manual Rec
653     } else {
654         ESP_LOGW(TAG, "Manual Rec
655     }
656 }
657
658
659
660d aws_iot_task(void *param) {
661 char cPayload[100];
662
663 short ecgHandsOn;
664 unsigned int ecgAcqCounter;
665 unsigned int ecgRecCounter;
666 unsigned int ecgMqttCounter;
667 unsigned short *headBuffer;
668 unsigned char *headTxBuffer;
669 //unsigned char *mqttTxBuffer;
670 unsigned short sequenceTimer;
671 unsigned short oldSequenceTimer;
672 unsigned short ledSelect;
673
674 char strftime_buf[64]; // for snt
675
676
677 //ycc 031221 end
678 ecgState = (short) ECG_IDLE;
679 ecgHandsOn = 0;
680 ecgAcqCounter = 0;
681 ecgRecCounter = 0;
682 ecgMqttCounter = 0;
683 sequenceTimer = 0;
684 oldSequenceTimer=0;
685 ledSelect = LED_GREEN;
686 jobCheckState = JOB_CHECK_STATE_N
687
688 AWS_IoT_Client client;
689
690 headTxBuffer = (unsigned char *)c
691 if(headTxBuffer == NULL)
692     {
693         ESP_LOGE(TAG, "Failed to
694         abort();
695     }
696
697 headBuffer = (char *)headTxBuffer
698 dataBuffer = headBuffer;
699
700 //mqttTxBuffer = (unsigned char *
```

```
607             if(NETWORK_RECONNECTE
608         ESP_LOGW(TAG, "Manual Rec
609     } else {
610         ESP_LOGW(TAG, "Manual Rec
611     }
612 }
613
614
615
616d aws_iot_task(void *param) {
617 char cPayload[100];
618
619 short ecgHandsOn;
620 unsigned int ecgAcqCounter;
621 unsigned int ecgRecCounter;
622 unsigned int ecgMqttCounter;
623 unsigned short *headBuffer;
624 unsigned char *headTxBuffer;
625 //unsigned char *mqttTxBuffer;
626 unsigned short sequenceTimer;
627 unsigned short oldSequenceTimer;
628 unsigned short ledSelect;
629
630 char strftime_buf[64]; // for snt
631
632
633 //ycc 031221 end
634 ecgState = (short) ECG_IDLE;
635 ecgHandsOn = 0;
636 ecgAcqCounter = 0;
637 ecgRecCounter = 0;
638 ecgMqttCounter = 0;
639 sequenceTimer = 0;
640 oldSequenceTimer=0;
641 ledSelect = LED_GREEN;
642 jobCheckState = JOB_CHECK_STATE_N
643
644 AWS_IoT_Client client;
645
646 headTxBuffer = (unsigned char *)c
647 if(headTxBuffer == NULL)
648     {
649         ESP_LOGE(TAG, "Failed to
650         abort();
651     }
652
653 headBuffer = (char *)headTxBuffer
654 dataBuffer = headBuffer;
655
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 debouceCounter = 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 debouceCounter = 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 (sntp_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
766le (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 (sntp_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
721le (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                                                     
802         gpio_set_level(ledSelect,                   
803                                                     
804     }                                               
805     counter3++;                                     
806     jobCheckCounter++;                              
807     if (counter >300)                               
808         {                                           
809         //printf("---------------                   
810         counter = 0; //this count                   
811         //ycc 032122                                
812                                                     
813         //uint32_t adc_reading =                    
814         //Multisampling                             
815         //for (int i = 0; i < NO_                   
816         //    adc_reading += adc1                   
817                                                     
818         //adc_reading /= NO_OF_SA                   
819         //Convert adc_reading to                    
820         //uint32_t voltage = esp_                   
821         //printf("Raw: %d\tVoltag                   
822         //ESP_LOGI(TAG, "%f,%f,%f                   
823         //if(ecgState != ECG_IDLE                   
824         uint32_t fs = xPortGetFre                   
825         ESP_LOGI(TAG, "ecg state                    
826                                                     
827         time(&now);                                 
828         //localtime_r(&now, &time                   
829         struct tm *tm_struct = lo                   
830         strftime(strftime_buf, si                   
831         ESP_LOGI(TAG, "%s=  %d",                    
832         ESP_LOGI(TAG, "%f,%f,%f,%                   
833         if((tm_struct->tm_hour==0                   
834             jobCheckState = JOB_C                   
835         // if time is past midnig                   
836         if((tm_struct->tm_hour ==                   
837             gpio_set_level(LED_GR                   
838             gpio_set_level(LED_BL                   
839             gpio_set_level(LED_RE                   
840             aws_iot_demo_main(0,                    
841             jobCheckState =  JOB_                   
842             jobCheckCounter = 0;                    
843             gpio_set_level(LED_GR                   
844             gpio_set_level(LED_BL                   
845             gpio_set_level(LED_RE                   
846             }                                       
847             */                                      
848         }                                           
849                                                     
850     if(((gpio_get_level((gpio_num                   
```

```
756                                                     
757         gpio_set_level(ledSelect,                   
758                                                     
759     }                                               
760     counter3++;                                     
761     jobCheckCounter++;                              
762     if (counter >300)                               
763         {                                           
764         //printf("---------------                   
765         counter = 0; //this count                   
766         //ycc 032122                                
767                                                     
768         //uint32_t adc_reading =                    
769         //Multisampling                             
770         //for (int i = 0; i < NO_                   
771         //    adc_reading += adc1                   
772                                                     
773         //adc_reading /= NO_OF_SA                   
774         //Convert adc_reading to                    
775         //uint32_t voltage = esp_                   
776         //printf("Raw: %d\tVoltag                   
777         //ESP_LOGI(TAG, "%f,%f,%f                   
778         //if(ecgState != ECG_IDLE                   
779         uint32_t fs = xPortGetFre                   
780         ESP_LOGI(TAG, "ecg state                    
781                                                     
782         time(&now);                                 
783         //localtime_r(&now, &time                   
784         struct tm *tm_struct = lo                   
785         strftime(strftime_buf, si                   
786         ESP_LOGI(TAG, "%s=  %d",                    
787         ESP_LOGI(TAG, "%f,%f,%f,%                   
788         if((tm_struct->tm_hour==0                   
789             jobCheckState = JOB_C                   
790         // if time is past midnig                   
791         if((tm_struct->tm_hour ==                   
792             gpio_set_level(LED_GR                   
793             gpio_set_level(LED_BL                   
794             gpio_set_level(LED_RE                   
795             aws_iot_demo_main(0,                    
796             jobCheckState =  JOB_                   
797             jobCheckCounter = 0;                    
798             gpio_set_level(LED_GR                   
799             gpio_set_level(LED_BL                   
800             gpio_set_level(LED_RE                   
801             }                                       
802             */                                      
803         }                                           
804                                                     
805     if(((gpio_get_level((gpio_num                   
```

```
851         {
852         pVariance=300000; pAvg=30
853         variance = 1000;
854         //counter = 0;
855         //counter3=0;
856         ecgHandsOn = 0;
857
858         //if((gpio_get_level((gpi
859         //    {
860         //        ecgState = ECG_
861         //    }
862         }
863     else
864         {
865         //ycc 022022
866         if(debouceCounter<=100){
867             debouceCounter++;
868         }
869         else {
870             debouceCounter = 0;
871         }
872         ecgHandsOn = 1;
873         gpio_set_level(LED_RED, 1
874         // send the value of anal
875         //Data = 0;
876         //for (int i = 0; i < NO_
877         aData = adc1_get_raw((adc
878         //aData = aData/NO_OF_SAM
879         if((aData<4090) && (aData
880             bpfX = (float)aData;
881         /*
882         w0 = 3.336612*w1 -4.22598
883         bpfX1 = 0.036575*(w0 - 2.
884         w4 = w3;
885         w3 = w2;
886         w2 = w1;
887         w1 = w0;
888         */
889         w0 = 3.269793*w1 -4.16941
890         bpfX1 = 0.082619*(w0 - 2.
891         w4 = w3;
892         w3 = w2;
893         w2 = w1;
894         w1 = w0;
895         w0 = 3.000162*w1 -3.29408
896         bpfX1 = 0.065274*(w0 - 2.
897         w4 = w3;
898         w3 = w2;
899         w2 = w1;
900         w1 = w0;
```

```
806         {
807         pVariance=300000; pAvg=30
808         variance = 1000;
809         //counter = 0;
810         //counter3=0;
811         ecgHandsOn = 0;
812
813         //if((gpio_get_level((gpi
814         //    {
815         //        ecgState = ECG_
816         //    }
817         }
818     else
819         {
820         //ycc 022022
821         if(debouceCounter<=100){
822             debouceCounter++;
823         }
824         else {
825             debouceCounter = 0;
826         }
827         ecgHandsOn = 1;
828         gpio_set_level(LED_RED, 1
829         // send the value of anal
830         //Data = 0;
831         //for (int i = 0; i < NO_
832         aData = adc1_get_raw((adc
833         //aData = aData/NO_OF_SAM
834         if((aData<4090) && (aData
835             bpfX = (float)aData;
836         /*
837         w0 = 3.336612*w1 -4.22598
838         bpfX1 = 0.036575*(w0 - 2.
839         w4 = w3;
840         w3 = w2;
841         w2 = w1;
842         w1 = w0;
843         */
844         w0 = 3.269793*w1 -4.16941
845         bpfX1 = 0.082619*(w0 - 2.
846         w4 = w3;
847         w3 = w2;
848         w2 = w1;
849         w1 = w0;
850         w0 = 3.000162*w1 -3.29408
851         bpfX1 = 0.065274*(w0 - 2.
852         w4 = w3;
853         w3 = w2;
854         w2 = w1;
855         w1 = w0;
```

```
901         //end-band pass filter
902
903         bpfS0=bpfS1;
904         bpfS1=bpfS2;
905         bpfS2=bpfS3;
906         bpfS3=bpfS4;
907         bpfS4=bpfS5;
908         bpfS5=bpfS6;
909         bpfS6=bpfX1;
910
911         if((bpfS3>(mAvg+noiseFloo
912             {
913             //heart beat detected
914
915             heartRate = 30000/pea
916
917             m0=m1;
918             m1=m2;
919             m2=m3;
920             m3=m4;
921             //if(bpfS3> 40 && bpf
922             m4=bpfS3;
923             mAvg = (m0+m1+m2+m3+m
924             //mVariance = (m0-mAv
925             p0=p1;
926             p1=p2;
927             p2=p3;
928             p3=p4;
929             p4=peakCounter;
930
931             pAvg = (p0+p1+p2+p3+p
932             pVariance = ((p0-pAvg
933             variance = sqrt(pVari
934
935             //digitalWrite(LEDPin
936             detectedBeat++;
937             bpftmp = bpfS3;
938             peakCounter = 0;
939             }
940         else
941             {
942             if(peakCounter> 5000)
943                {
944                 peakCounter = 0;
945                 detectedBeat--;
946                 }
947
948             peakCounter++;
949             if((bpfS3>bpfS2) && (
950                 {
```

```
856         //end-band pass filter
857
858         bpfS0=bpfS1;
859         bpfS1=bpfS2;
860         bpfS2=bpfS3;
861         bpfS3=bpfS4;
862         bpfS4=bpfS5;
863         bpfS5=bpfS6;
864         bpfS6=bpfX1;
865
866         if((bpfS3>(mAvg+noiseFloo
867             {
868             //heart beat detected
869
870             heartRate = 30000/pea
871
872             m0=m1;
873             m1=m2;
874             m2=m3;
875             m3=m4;
876             //if(bpfS3> 40 && bpf
877             m4=bpfS3;
878             mAvg = (m0+m1+m2+m3+m
879             //mVariance = (m0-mAv
880             p0=p1;
881             p1=p2;
882             p2=p3;
883             p3=p4;
884             p4=peakCounter;
885
886             pAvg = (p0+p1+p2+p3+p
887             pVariance = ((p0-pAvg
888             variance = sqrt(pVari
889
890             //digitalWrite(LEDPin
891             detectedBeat++;
892             bpftmp = bpfS3;
893             peakCounter = 0;
894             }
895         else
896             {
897             if(peakCounter> 5000)
898                {
899                 peakCounter = 0;
900                 detectedBeat--;
901                 }
902
903             peakCounter++;
904             if((bpfS3>bpfS2) && (
905                 {
```

```
                                                    noiseFloor = (noi
                                                    if(peakCounter>50
                                                    }
                                                bpftmp = 0;
                                                } //end if-else

                                            }//end if-else
                                        //state machine

                                        switch(ecgState)
                                            {
                                            case ECG_IDLE:
                                                if(ecgHandsOn == 1){
                                                    //printf("IDLE mo
                                                    ecgState = ECG_AC
                                                    //start ACQ timer
                                                    ecgAcqCounter = 0
                                                    ecgRecCounter = 0
                                                    ecgMqttCounter =
                                                    dataBuffer = head
                                                    deepSleepCounter
                                                }
                                                else {
                                                    deepSleepCounter-
                                                    if(deepSleepCount
                                                         //deep sleep
                                                        // Initialize
                                                        // The defaul

                                                        ESP_ERROR_CHE
                                                        // If use tou

                                                        touch_pad_set
                                                        // Set refere
                                                        // In this ca
                                                        // The low re
                                                        // The larger

                                                        touch_pad_set

                                                        //init RTC IO
                                                        // touch pad

                                                        touch_pad_con

                                                        calibrate_tou

                                                        printf("Enabl
                                                        esp_sleep_ena
```

```
                    esp_sleep_pd_

                    // Isolate GP
                    // which have
                    // to minimiz
                    rtc_gpio_isol
                    gpio_set_leve
                    printf("Enter
                    esp_deep_slee
                }
            }
            break;
        case ECG_ACQUIRING:
            ecgAcqCounter++;
            if(ecgHandsOn == 0)
                {
                //hands off go ba
                printf("ACQUIRING
                ecgState = ECG_ID
                ecgAcqCounter = 0
                }
            else if (ecgAcqCounte
                {//can't recogniz
                printf("ACQUIRING
                ecgState = ECG_RE
                //start recording
                ecgRecCounter = 0
                //set up pointer
                }
            else if (variance < 1
                {
                //printf("varianc
                ecgState = ECG_RE
                //start recording
                ecgRecCounter = 0
                //ecg signal reco
                //set up pointer
                }
            break;
        case ECG_RECORDING:
                //dataBuffer[ecgR
                *dataBuffer++ = a
                ecgRecCounter++;
                if(ecgHandsOn ==
                    {   //hands o
                        //printf(
                        if (ecgRe
                            { //S
                                /
                                e
```

```
            }
            break;
        case ECG_ACQUIRING:
            ecgAcqCounter++;
            if(ecgHandsOn == 0)
                {
                //hands off go ba
                printf("ACQUIRING
                ecgState = ECG_ID
                ecgAcqCounter = 0
                }
            else if (ecgAcqCounte
                {//can't recogniz
                printf("ACQUIRING
                ecgState = ECG_RE
                //start recording
                ecgRecCounter = 0
                //set up pointer
                }
            else if (variance < 1
                {
                //printf("varianc
                ecgState = ECG_RE
                //start recording
                ecgRecCounter = 0
                //ecg signal reco
                //set up pointer
                }
            break;
        case ECG_RECORDING:
                //dataBuffer[ecgR
                *dataBuffer++ = a
                ecgRecCounter++;
                if(ecgHandsOn ==
                    {   //hands o
                        //printf(
                        if (ecgRe
                            { //S
                                /
                                e
```

```
                                    /
                                    e
                                }
                            else
                            //hands off g
                                {
                                ecgState
                                //printf(
                                }
                            }
                        else if (ecgRecCo
                            {//finish tim
                                //printf(
                                ecgState
                                //ecgRecC
                                ecgMqttCo
                                dataBuffe
                            }

                    break;
                case ECG_SENDING_MQTT:
                    //AWS_IoT_Client clie
                    rc = FAILURE;
                    gpio_set_level(LED_BL
                    //ycc 3-13-22 added t
                    /* Return error statu
                    int returnStatus = EX


                    if( returnStatus == E
                    {
                        /* Start OTA demo
                        //returnStatus =
                        //ycc 031422
                        uint32_t free_hea
                        free_heap_size =
                        min_free_heap_siz
                        //printf("\n free
                        uint32_t fs = xPo
                        ESP_LOGI(TAG, "Se
                        if( mqttSessionEs
                            int ret = est
                            //printf("est
                            //printf("Pub
                        }
                        if( mqttSessionEs

                            unsigned shor
                            char topic_na
                            const char *t
```

```
                                    /
                                    e
                                }
                            else
                            //hands off g
                                {
                                ecgState
                                //printf(
                                }
                            }
                        else if (ecgRecCo
                            {//finish tim
                                //printf(
                                ecgState
                                //ecgRecC
                                ecgMqttCo
                                dataBuffe
                            }

                    break;
                case ECG_SENDING_MQTT:
                    //AWS_IoT_Client clie
                    rc = FAILURE;
                    gpio_set_level(LED_BL
                    //ycc 3-13-22 added t
                    /* Return error statu
                    int returnStatus = EX


                    if( returnStatus == E
                    {
                        /* Start OTA demo
                        //returnStatus =
                        //ycc 031422
                        uint32_t free_hea
                        free_heap_size =
                        min_free_heap_siz
                        //printf("\n free
                        uint32_t fs = xPo
                        ESP_LOGI(TAG, "Se
                        if( mqttSessionEs
                            int ret = est
                            //printf("est
                            //printf("Pub
                        }
                        if( mqttSessionEs

                            unsigned shor
                            char topic_na
                            const char *t
```

```
1101                    uint8_t brd_m
1102                    esp_wifi_get_
1103                    snprintf(topi
1104                        topic_pre
1105                    const char *T
1106                    const int TOP
1107                    mqttPublishNo
1108
1109
1110
1111
1112
1113                }
1114
1115
1116
1117            }
1118
1119            //ycc end 3-13-22
1120            //ycc 031422 end
1121            disconnect();
1122            ecgState =  ECG_FINIS
1123            ecgRecCounter = 0;
1124            jobCheckCounter = 0;
1125
1126            ecgMqttCounter++;
1127            break;
1128        case ECG_FINISH:
1129            if(ecgHandsOn == 0){
1130                ecgState = ECG_ID
1131                jobCheckCounter =
1132                break;
1133            }
1134            else {
1135
1136                    if(jobCheckCo
1137                        ecgState
1138                        break;
1139                    }
1140                    else{
1141                        gpio_set_
1142                        gpio_set_
1143                        gpio_set_
1144
1145                        //ESP_LOG
1146                        // ycc 03
1147                        //int ret
1148                        //ESP_LOG
1149                        //if(ret=
1150                        //    job
```

```
1017                    uint8_t brd_m
1018                    esp_wifi_get_
1019                    snprintf(topi
1020                        topic_pre
1021                    const char *T
1022                    const int TOP
1023                    mqttPublishNo
1024
1025
1026
1027
1028
1029                }
1030
1031
1032
1033            }
1034
1035            //ycc end 3-13-22
1036            //ycc 031422 end
1037            disconnect();
1038            ecgState =  ECG_FINIS
1039            ecgRecCounter = 0;
1040            jobCheckCounter = 0;
1041
1042            ecgMqttCounter++;
1043            break;
1044        case ECG_FINISH:
1045            if(ecgHandsOn == 0){
1046                ecgState = ECG_ID
1047                jobCheckCounter =
1048                break;
1049            }
1050            else {
1051
1052                    if(jobCheckCo
1053                        ecgState
1054                        break;
1055                    }
1056                    else{
1057                        gpio_set_
1058                        gpio_set_
1059                        gpio_set_
1060
1061                        //ESP_LOG
1062                        // ycc 03
1063                        //int ret
1064                        //ESP_LOG
1065                        //if(ret=
1066                        //    job
```

```
1151                        jobCheckC
1152                        ecgState
1153                        gpio_set_
1154                        gpio_set_
1155                        gpio_set_
1156
1157                        break;
1158                    }
1159
1160            }
1161            break;
1162        case ECG_OTA_UPDATE:
1163            ecgState = ECG_OTA_UP
1164            break;
1165        case ECG_SSID_RESET:
1166            if(counter3 <3000)
1167                break;
1168            else
1169                {
1170                ecgState = ECG_ID
1171                counter3 = 0;
1172                }
1173            break;
1174        default:
1175            //printf("Default mov
1176            ecgState = ECG_IDLE;
1177            break;
1178        }
1179
1180 }//end while
1181
1182 ESP_LOGE(TAG, "An error occurred
1183 abort();
1184
1185
1186
1187d app_main()
1188
1189 esp_chip_info_t chip_info;
1190 esp_chip_info(&chip_info);
1191 /*
1192 printf("This is %s chip with %d C
1193         CHIP_NAME,
1194         chip_info.cores,
1195         (chip_info.features & CHI
1196         (chip_info.features & CHI
1197 */
1198 //printf("silicon revision %d, ",
1199
1200 //printf("%dMB %s flash\n", spi_f
```

```
1067                        jobCheckC
1068                        ecgState
1069                        gpio_set_
1070                        gpio_set_
1071                        gpio_set_
1072
1073                        break;
1074                    }
1075
1076            }
1077            break;
1078        case ECG_OTA_UPDATE:
1079            ecgState = ECG_OTA_UP
1080            break;
1081        case ECG_SSID_RESET:
1082            if(counter3 <3000)
1083                break;
1084            else
1085                {
1086                ecgState = ECG_ID
1087                counter3 = 0;
1088                }
1089            break;
1090        default:
1091            //printf("Default mov
1092            ecgState = ECG_IDLE;
1093            break;
1094        }
1095
1096 }//end while
1097
1098 ESP_LOGE(TAG, "An error occurred
1099 abort();
1100
1101
1102
1103d app_main()
1104
1105 esp_chip_info_t chip_info;
1106 esp_chip_info(&chip_info);
1107 /*
1108 printf("This is %s chip with %d C
1109         CHIP_NAME,
1110         chip_info.cores,
1111         (chip_info.features & CHI
1112         (chip_info.features & CHI
1113 */
1114 //printf("silicon revision %d, ",
1115
1116 //printf("%dMB %s flash\n", spi_f
```

```
1201 //        (chip_info.features & C
1202
1203 //072521 ycc added the following
1204 app_driver_init();
1205
1206 //printf("Configured WiFi SSID is
1207 // Initialize NVS.
1208 esp_err_t err = nvs_flash_init();
1209 if (err == ESP_ERR_NVS_NO_FREE_PA
1210     ESP_ERROR_CHECK(nvs_flash_era
1211     err = nvs_flash_init();
1212 }
1213 ESP_ERROR_CHECK( err );
1214
1215
1216 s_timer_queue = xQueueCreate(10,
1217 //GPIO setup
1218 gpio_reset_pin(32);
1219 touch_pad_set_fsm_mode(TOUCH_FSM_
1220 gpio_set_direction(LOPlus, GPIO_M
1221 gpio_set_direction(LOMinus, GPIO_
1222 gpio_set_direction(SDN, GPIO_MODE
1223 gpio_set_direction(FR, GPIO_MODE_
1224 gpio_set_direction(DC, GPIO_MODE_
1225 gpio_set_direction(LED_GREEN, GPI
1226 gpio_set_direction(LED_BLUE, GPIO
1227 gpio_set_direction(LED_RED, GPIO_
1228
1229 gpio_set_direction(DEEPSLEEP, GPI
1230
1231 gpio_set_level(LED_GREEN, 1);
1232 gpio_set_level(LED_BLUE, 1);
1233 gpio_set_level(LED_RED, 1);
1234 gpio_set_level(SDN, 1);
1235 gpio_set_level(FR, 1);
1236 gpio_set_level(DC, 0);
1237
1238 //gpio_set_level(SDN, 0);
1239 //ESP_ERROR_CHECK(esp_sleep_disab
1240 //ESP_ERROR_CHECK(touch_pad_deini
1241
1242 ad_tg_timer_init(TIMER_GROUP_0, T
1243 /*example_tg_timer_init(TIMER_GRO
1244 //Check if Two Point or Vref are
1245 check_efuse();
1246
1247 //Configure ADC
1248 adc1_config_width(width);
1249 adc1_config_channel_atten(channel
1250
```

```
1117 //        (chip_info.features & C
1118
1119 //072521 ycc added the following
1120 app_driver_init();
1121
1122 //printf("Configured WiFi SSID is
1123 // Initialize NVS.
1124 esp_err_t err = nvs_flash_init();
1125 if (err == ESP_ERR_NVS_NO_FREE_PA
1126     ESP_ERROR_CHECK(nvs_flash_era
1127     err = nvs_flash_init();
1128 }
1129 ESP_ERROR_CHECK( err );
1130
1131
1132 s_timer_queue = xQueueCreate(10,
1133 //GPIO setup
1134 gpio_set_direction(LOPlus, GPIO_M
1135 gpio_set_direction(LOMinus, GPIO_
1136 gpio_set_direction(SDN, GPIO_MODE
1137 gpio_set_direction(FR, GPIO_MODE_
1138 gpio_set_direction(DC, GPIO_MODE_
1139 gpio_set_direction(LED_GREEN, GPI
1140 gpio_set_direction(LED_BLUE, GPIO
1141 gpio_set_direction(LED_RED, GPIO_
1142 gpio_set_level(LED_GREEN, 1);
1143 gpio_set_level(LED_BLUE, 1);
1144 gpio_set_level(LED_RED, 1);
1145 gpio_set_level(SDN, 1);
1146 gpio_set_level(FR, 1);
1147 gpio_set_level(DC, 0);
1148
1149 //gpio_set_level(SDN, 0);
1150
1151 ad_tg_timer_init(TIMER_GROUP_0, T
1152 /*example_tg_timer_init(TIMER_GRO
1153 //Check if Two Point or Vref are
1154 check_efuse();
1155
1156 //Configure ADC
1157 adc1_config_width(width);
1158 adc1_config_channel_atten(channel
1159
```

```
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
```

```
ESP_ERROR_CHECK(wifi_prov_mgr_is_
                                    
/* If device is not yet provision
if (!provisioned) {                 
    ESP_LOGI(TAG, "Starting provi
    gpio_set_level(LED_RED, 0); /
    /* What is the Device Service
     * This translates to :       
     *     - Wi-Fi SSID when sche
     *     - device name when sch
     */                            
    char service_name[12];         
    get_device_service_name(servi
    /* What is the security level
     *      - WIFI_PROV_SECURITY_
     *      - WIFI_PROV_SECURITY_
     *          using X25519 key 
     *          for encryption/de
     */                            
    wifi_prov_security_t security
                                    
    /* Do we want a proof-of-poss
     *      - this should be a st
     *      - NULL if not used    
     */                            
    const char *pop = "abcd1234";
                                    
    /* What is the service key (c
     * This translates to :       
     *     - Wi-Fi password when  
     *     - simply ignored when  
     */                            
    const char *service_key = NUL
                                    
                                    
    /* This step is only useful w
     * set a custom 128 bit UUID 
     * and will correspond to the
     * endpoints as GATT characte
     * formed using the primary s
     * 12th and 13th bytes (assum
     * applications must identify
     * Description descriptor (0x
     * endpoint name of the chara
    uint8_t custom_service_uuid[]
        /* LSB <----------------- 
         * ----------------------
        0xb4, 0xdf, 0x5a, 0x1c, 0
        0xea, 0x4a, 0x82, 0x03, 0
    };                              
```

```
ESP_ERROR_CHECK(wifi_prov_mgr_is_
                                    
/* If device is not yet provision
if (!provisioned) {                 
    ESP_LOGI(TAG, "Starting provi
    gpio_set_level(LED_RED, 0); /
    /* What is the Device Service
     * This translates to :       
     *     - Wi-Fi SSID when sche
     *     - device name when sch
     */                            
    char service_name[12];         
    get_device_service_name(servi
    /* What is the security level
     *      - WIFI_PROV_SECURITY_
     *      - WIFI_PROV_SECURITY_
     *          using X25519 key 
     *          for encryption/de
     */                            
    wifi_prov_security_t security
                                    
    /* Do we want a proof-of-poss
     *      - this should be a st
     *      - NULL if not used    
     */                            
    const char *pop = "abcd1234";
                                    
    /* What is the service key (c
     * This translates to :       
     *     - Wi-Fi password when  
     *     - simply ignored when  
     */                            
    const char *service_key = NUL
                                    
                                    
    /* This step is only useful w
     * set a custom 128 bit UUID 
     * and will correspond to the
     * endpoints as GATT characte
     * formed using the primary s
     * 12th and 13th bytes (assum
     * applications must identify
     * Description descriptor (0x
     * endpoint name of the chara
    uint8_t custom_service_uuid[]
        /* LSB <----------------- 
         * ----------------------
        0xb4, 0xdf, 0x5a, 0x1c, 0
        0xea, 0x4a, 0x82, 0x03, 0
    };                              
```

```
    wifi_prov_scheme_ble_set_serv
                                    
                                    
    /* An optional endpoint that 
     * get some additional custom 
     * The endpoint name can be a 
     * This call must be made bef 
     */                            
    wifi_prov_mgr_endpoint_create 
    /* Start provisioning service 
    ESP_ERROR_CHECK(wifi_prov_mgr 
                                    
    /* The handler for the option 
     * This call must be made aft 
     * has already been created a 
     */                            
    wifi_prov_mgr_endpoint_regist 
                                    
                                    
    wifi_prov_print_qr(service_na 
    /* Uncomment the following to 
     * the resources of the manag 
     * by the default event loop  
     wifi_prov_mgr_wait();         
     wifi_prov_mgr_deinit();       
    /* Print QR code for provisio 
                                    
    //wifi_prov_print_qr(service_ 
} else {                            
     ESP_LOGI(TAG, "Already provi 
                                    
    /* We don't need the manager  
     * so let's release it's reso 
    wifi_prov_mgr_deinit();       
    ESP_ERROR_CHECK( esp_wifi_set 
    ESP_ERROR_CHECK( esp_wifi_sta 
                                    
}                                   
/* ycc 030922 added to Wait for W 
int bits = xEventGroupWaitBits(wi 
                      false, true,  
                                    
if (!(bits & CONNECTED_BIT)) {      
    ESP_LOGE(TAG, "timeout bits=% 
    gpio_set_level(LED_RED, 0);   
}                                   
                                    
//ycc 051722 get board mac addres 
//get mac address                   
uint8_t brd_mac[6];                 
```

```
    wifi_prov_scheme_ble_set_serv
                                    
                                    
    /* An optional endpoint that 
     * get some additional custom 
     * The endpoint name can be a 
     * This call must be made bef 
     */                            
    wifi_prov_mgr_endpoint_create 
    /* Start provisioning service 
    ESP_ERROR_CHECK(wifi_prov_mgr 
                                    
    /* The handler for the option 
     * This call must be made aft 
     * has already been created a 
     */                            
    wifi_prov_mgr_endpoint_regist 
                                    
                                    
    wifi_prov_print_qr(service_na 
    /* Uncomment the following to 
     * the resources of the manag 
     * by the default event loop  
     wifi_prov_mgr_wait();         
     wifi_prov_mgr_deinit();       
    /* Print QR code for provisio 
                                    
    //wifi_prov_print_qr(service_ 
} else {                            
     ESP_LOGI(TAG, "Already provi 
                                    
    /* We don't need the manager  
     * so let's release it's reso 
    wifi_prov_mgr_deinit();       
    ESP_ERROR_CHECK( esp_wifi_set 
    ESP_ERROR_CHECK( esp_wifi_sta 
                                    
}                                   
/* ycc 030922 added to Wait for W 
int bits = xEventGroupWaitBits(wi 
                      false, true,  
                                    
if (!(bits & CONNECTED_BIT)) {      
    ESP_LOGE(TAG, "timeout bits=% 
    gpio_set_level(LED_RED, 0);   
}                                   
                                    
//ycc 051722 get board mac addres 
//get mac address                   
uint8_t brd_mac[6];                 
```

```
esp_wifi_get_mac(WIFI_IF_STA, brd
snprintf(macAddress, 13, "%02X%02
//printf("macAddress is %s, lengt
//ycc 3-15-22 initialize mqtt and
/* Return error status. */
int returnStatus = EXIT_SUCCESS;
/* Semaphore initialization flag.
bool bufferSemInitialized = false
bool mqttMutexInitialized = false

mqttSessionEstablished = false;

/* Initialize semaphore for buffe
if( osi_sem_new( &bufferSemaphore
{
    LogError( ( "Failed to initia
                ",errno=%s",
                strerror( errno )

    returnStatus = EXIT_FAILURE;
}
else
{
    bufferSemInitialized = true;
}

/* Initialize mutex for coreMQTT
if( pthread_mutex_init( &mqttMute
{
    LogError( ( "Failed to initia
                ",errno=%s",
                strerror( errno )

    returnStatus = EXIT_FAILURE;
}
else
{
    mqttMutexInitialized = true;
}

if( returnStatus == EXIT_SUCCESS
{
    /* Initialize MQTT library. I
        * done only once in this
    returnStatus = initializeMqtt
}
//aws_iot_task(&aws_iot_task);
//xTaskCreatePinnedToCore(&aws_io

//ycc 070522 check nvs to see if
```

```
esp_wifi_get_mac(WIFI_IF_STA, brd
snprintf(macAddress, 13, "%02X%02
//printf("macAddress is %s, lengt
//ycc 3-15-22 initialize mqtt and
/* Return error status. */
int returnStatus = EXIT_SUCCESS;
/* Semaphore initialization flag.
bool bufferSemInitialized = false
bool mqttMutexInitialized = false

mqttSessionEstablished = false;

/* Initialize semaphore for buffe
if( osi_sem_new( &bufferSemaphore
{
    LogError( ( "Failed to initia
                ",errno=%s",
                strerror( errno )

    returnStatus = EXIT_FAILURE;
}
else
{
    bufferSemInitialized = true;
}

/* Initialize mutex for coreMQTT
if( pthread_mutex_init( &mqttMute
{
    LogError( ( "Failed to initia
                ",errno=%s",
                strerror( errno )

    returnStatus = EXIT_FAILURE;
}
else
{
    mqttMutexInitialized = true;
}

if( returnStatus == EXIT_SUCCESS
{
    /* Initialize MQTT library. I
        * done only once in this
    returnStatus = initializeMqtt
}
//aws_iot_task(&aws_iot_task);
//xTaskCreatePinnedToCore(&aws_io

//ycc 070522 check nvs to see if
```

```
printf("Opening Non-Volatile Stor
//nvs_handle_t fleet_prov_handle;
err = nvs_open("storage", NVS_REA
if (err != ESP_OK) {
    printf("Error (%s) opening NV
} else {
    printf("Done\n");

    // Read
    printf("Reading fleet prov ce
    //size_t private_key_len;
    err = nvs_get_str(fleet_prov_
    private_key = malloc(private_
    err = nvs_get_str(fleet_prov_
    switch (err) {
        case ESP_OK:
            printf("Done\n");
            //printf("private_key
            nvsProvisionStatus =
            break;
        case ESP_ERR_NVS_NOT_FOUN
            printf("The value is
            nvsProvisionStatus =
            break;
        default :
            printf("Error (%s) re
            nvsProvisionStatus =
    }
    //size_t certificate_pem_len;
    err = nvs_get_str(fleet_prov_
    certificate_pem = malloc(cert
    if(nvsProvisionStatus == true
        {
        err = nvs_get_str(fleet_p
        switch (err) {
            case ESP_OK:
                printf("Done\n");
                //printf("certifi
                nvsProvisionStatu
                break;
            case ESP_ERR_NVS_NOT_
                printf("The value
                nvsProvisionStatu
                break;
            default :
                printf("Error (%s
                nvsProvisionStatu
            }
        }
}
```

```
printf("Opening Non-Volatile Stor
//nvs_handle_t fleet_prov_handle;
err = nvs_open("storage", NVS_REA
if (err != ESP_OK) {
    printf("Error (%s) opening NV
} else {
    printf("Done\n");

    // Read
    printf("Reading fleet prov ce
    //size_t private_key_len;
    err = nvs_get_str(fleet_prov_
    private_key = malloc(private_
    err = nvs_get_str(fleet_prov_
    switch (err) {
        case ESP_OK:
            printf("Done\n");
            //printf("private_key
            nvsProvisionStatus =
            break;
        case ESP_ERR_NVS_NOT_FOUN
            printf("The value is
            nvsProvisionStatus =
            break;
        default :
            printf("Error (%s) re
            nvsProvisionStatus =
    }
    //size_t certificate_pem_len;
    err = nvs_get_str(fleet_prov_
    certificate_pem = malloc(cert
    if(nvsProvisionStatus == true
        {
        err = nvs_get_str(fleet_p
        switch (err) {
            case ESP_OK:
                printf("Done\n");
                //printf("certifi
                nvsProvisionStatu
                break;
            case ESP_ERR_NVS_NOT_
                printf("The value
                nvsProvisionStatu
                break;
            default :
                printf("Error (%s
                nvsProvisionStatu
            }
        }
}
```

```
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("lenghth of re       1457             printf("lenghth of re
1549             mqttReturn = mqttPubl       1458             mqttReturn = mqttPubl
1550             printf("mqttreturn 5=       1459             printf("mqttreturn 5=
```

```
1551            nvsProvisionStatus =
1552
1553
1554            MQTTStatus_t mqttStat
1555            {
1556                        {
1557                /* Acquire the mq
1558                if( pthread_mutex
1559                {
1560                    /* Loop to re
1561                    mqttStatus =
1562
1563                    pthread_mutex
1564                }
1565                else
1566                {
1567                    LogError( ( "
1568                                "
1569                                s
1570                }
1571
1572                if( mqttStatus ==
1573                {
1574                    /* Get OTA st
1575                    /* Delay if m
1576                    if( MQTT_PROC
1577                    {
1578                        vTaskDela
1579                    }
1580                    printf("Delay
1581                }
1582                else
1583                {
1584                    LogError( ( "
1585                                M
1586
1587                    /* Disconnect
1588                    disconnect();
1589
1590
1591                }
1592                // Read
1593                printf("Reading f
1594
1595                err = nvs_get_str
1596                private_key = mal
1597                err = nvs_get_str
1598                switch (err) {
1599                    case ESP_OK:
1600                        printf("D
```

```
1460            nvsProvisionStatus =
1461
1462
1463            MQTTStatus_t mqttStat
1464            {
1465                        {
1466                /* Acquire the mq
1467                if( pthread_mutex
1468                {
1469                    /* Loop to re
1470                    mqttStatus =
1471
1472                    pthread_mutex
1473                }
1474                else
1475                {
1476                    LogError( ( "
1477                                "
1478                                s
1479                }
1480
1481                if( mqttStatus ==
1482                {
1483                    /* Get OTA st
1484                    /* Delay if m
1485                    if( MQTT_PROC
1486                    {
1487                        vTaskDela
1488                    }
1489                    printf("Delay
1490                }
1491                else
1492                {
1493                    LogError( ( "
1494                                M
1495
1496                    /* Disconnect
1497                    disconnect();
1498
1499
1500                }
1501                // Read
1502                printf("Reading f
1503
1504                err = nvs_get_str
1505                private_key = mal
1506                err = nvs_get_str
1507                switch (err) {
1508                    case ESP_OK:
1509                        printf("D
```

```
                        //printf(
                        break;
                    case ESP_ERR_
                        printf("T
                        break;
                    default :
                        printf("E
                }

                err = nvs_get_str
                certificate_pem =
                err = nvs_get_str
                switch (err) {
                    case ESP_OK:
                        printf("D
                        //printf(
                        break;
                    case ESP_ERR_
                        printf("T
                        break;
                    default :
                        printf("E
                }
                disconnect();
                printf("reconnect
                //ycc 062222 for
                if( mqttSessionEs
                    int ret = est
                    printf("estab
                        //printf(
                }

            }
        }

    }
}
disconnect();


if( bufferSemInitialized == true
{
    /* Cleanup semaphore created
    if( osi_sem_free( &bufferSema
    {
        LogError( ( "Failed to de
                    ",errno=%s",
                    strerror( err
```

```
                        //printf(
                        break;
                    case ESP_ERR_
                        printf("T
                        break;
                    default :
                        printf("E
                }

                err = nvs_get_str
                certificate_pem =
                err = nvs_get_str
                switch (err) {
                    case ESP_OK:
                        printf("D
                        //printf(
                        break;
                    case ESP_ERR_
                        printf("T
                        break;
                    default :
                        printf("E
                }
                disconnect();
                printf("reconnect
                //ycc 062222 for
                if( mqttSessionEs
                    int ret = est
                    printf("estab
                        //printf(
                }

            }
        }

    }
}
disconnect();


if( bufferSemInitialized == true
{
    /* Cleanup semaphore created
    if( osi_sem_free( &bufferSema
    {
        LogError( ( "Failed to de
                    ",errno=%s",
                    strerror( err
```

```
1651            returnStatus = EXIT_FAILU
1652        }
1653    }
1654
1655    if( mqttMutexInitialized == true
1656    {
1657        /* Cleanup mutex created for
1658        if( pthread_mutex_destroy( &m
1659        {
1660            LogError( ( "Failed to de
1661                        ",errno=%s",
1662                        strerror( err
1663
1664            returnStatus = EXIT_FAILU
1665        }
1666    }
1667
1668
```

```
1560            returnStatus = EXIT_FAILU
1561        }
1562    }
1563
1564    if( mqttMutexInitialized == true
1565    {
1566        /* Cleanup mutex created for
1567        if( pthread_mutex_destroy( &m
1568        {
1569            LogError( ( "Failed to de
1570                        ",errno=%s",
1571                        strerror( err
1572
1573            returnStatus = EXIT_FAILU
1574        }
1575    }
1576
1577
```