My Project

Generated by Doxygen 1.8.16

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

File Documentation

2.1 main/mesh_main.c File Reference

```
#include <string.h>
#include "esp_wifi.h"
#include "esp_system.h"
#include "esp_event_loop.h"
#include "esp_log.h"
#include "esp_mesh.h"
#include "esp_mesh_internal.h"
#include "nvs_flash.h"
#include "dwm1001.h"
#include "esp_nvs.h"
#include "esp_interrupt.h"
#include "triangulation.h"
#include "lwip/err.h"
#include "lwip/sockets.h"
#include "lwip/sys.h"
#include "lwip/netdb.h"
#include "lwip/dns.h"
#include "driver/i2c.h"
#include "ssd1366.h"
#include "font8x8_basic.h"
```

Macros

- #define RX_SIZE (1500)
- #define TX_SIZE (1460)
- #define MESH_SERVER_IP "192.168.0.103"
- #define MESH_PORT 3000
- #define SDA_PIN GPIO_NUM_21
- #define SCL_PIN GPIO_NUM_22
- #define tag "SSD1306"

Functions

- void i2c_master_init ()
- void ssd1306_init ()
- void task_ssd1306_display_text (const void *arg_text)
- void display_clear ()
- void disp_info ()
- void esp_mesh_p2p_tx_main (void *arg)
- void esp_mesh_p2p_rx_main (void *arg)
- esp_err_t esp_mesh_comm_p2p_start (void)
- void mesh_event_handler (mesh_event_t event)
- void app_main (void)

2.1.1 Macro Definition Documentation

2.1.1.1 MESH_PORT

```
#define MESH_PORT 3000
```

Definition at line 46 of file mesh_main.c.

2.1.1.2 MESH_SERVER_IP

```
#define MESH_SERVER_IP "192.168.0.103"
```

Definition at line 45 of file mesh_main.c.

2.1.1.3 RX_SIZE

```
#define RX_SIZE (1500)
```

Definition at line 43 of file mesh_main.c.

2.1.1.4 SCL_PIN

```
#define SCL_PIN GPIO_NUM_22
```

Definition at line 48 of file mesh_main.c.

2.1.1.5 SDA_PIN

```
#define SDA_PIN GPIO_NUM_21
```

Definition at line 47 of file mesh_main.c.

2.1.1.6 tag

```
#define tag "SSD1306"
```

Definition at line 49 of file mesh_main.c.

2.1.1.7 TX_SIZE

```
#define TX_SIZE (1460)
```

Definition at line 44 of file mesh_main.c.

2.1.2 Function Documentation

2.1.2.1 app_main()

```
void app_main (
            void  )
```

Main Program start

Definition at line 575 of file mesh_main.c.

```
00576 {
00577
00578     /**************************************************************************/
00579     /* ---------- ----------       Global Initialization   ---------- ---------- */
00580     /**************************************************************************/
00581     loc.id = (uint16_t *)malloc(sizeof(uint16_t) * NODE_LIMIT);   // Allocate # of slots for distance
      id's
00582     loc.dist = (uint32_t *)malloc(sizeof(uint32_t) * NODE_LIMIT); // Allocate # of slots for distances
00583     loc.qf = (uint8_t *)malloc(sizeof(uint8_t) * NODE_LIMIT);     // Allocate # of slots for quality
      factor of distances
00584
00585     /**************************************************************************/
00586     /* ---------- ----------       SPI Initialization      ---------- ---------- */
00587     /**************************************************************************/
00588     /* --- SPI Initializaion Start --- */
00589     spi_init();
00590     // -- Add slight delay to let all systems comfortably boot up
00591     vTaskDelay(1000 / portTICK_PERIOD_MS);
00592
00593     /**************************************************************************/
00594     /* ---------- ----------       I2C Initialization      ---------- ---------- */
00595     /**************************************************************************/
00596
00597     i2c_master_init();
00598     ssd1306_init();
00599
00600     // Display information on OLED
00601     disp_info();
00602
00603     /**************************************************************************/
00604     /* ---------- ----------    Node Configuration Check   ---------- ---------- */
00605     /**************************************************************************/
00606     dwm_resp_cfg_get dwm1cfg, dwm2cfg;
00607     dwm_resp_err err1, err2;
00608     bool rst = false;
00609     // -- Check if designated Anchor DWM is configured as desired
00610     dwm2cfg = dwm_cfg_get(NODE_TYPE_ANC);
00611     if ((dwm2cfg.cfg.cfg_bytes[0] != DWM2_CFG_BYTE0) && (dwm2cfg.cfg.cfg_bytes[1] != DWM2_CFG_BYTE1))
00612     {
00613         uint8_t anccfg_arg = 0x9A, intcfg_arg = 0x02;
00614         err2 = dwm_anc_set(anccfg_arg, NODE_TYPE_ANC);
00615         err2 = dwm_rst(NODE_TYPE_ANC);
00616         rst = true;
```

```
00617     }
00618     // -- Check if designated Anchor DWM is configured as desired
00619     dwm1cfg = dwm_cfg_get(NODE_TYPE_TAG);
00620     if ((dwm1cfg.cfg.cfg_bytes[0] != DWM1_CFG_BYTE0) && (dwm1cfg.cfg.cfg_bytes[1] != DWM1_CFG_BYTE1))
00621     {
00622         uint8_t tagcfg_arg[] = {0x9A, 0x00}, intcfg_arg = 0x02;
00623         err1 = dwm_tag_set(tagcfg_arg, NODE_TYPE_TAG);
00624         err1 = dwm_int_cfg(intcfg_arg, NODE_TYPE_TAG); // Does not seem to work, will ignore
      interrupts for now
00625         err1 = dwm_rst(NODE_TYPE_TAG);
00626         rst = true;
00627     }
00628     // -- If a DWM module had to reconfigure, we wait a few moments for it to reboot & connect to
      other DWMs
00629     if (rst == true)
00630         vTaskDelay(5000 / portTICK_PERIOD_MS);
00631 
00632     /*****************************************************************************/
00633     /* ---------- ----------    Mesh Initializaion Start   ---------- ---------- */
00634     /*****************************************************************************/
00635     // ESP_ERROR_CHECK(mesh_light_init());
00636     ESP_ERROR_CHECK(nvs_flash_init());
00637     /*  tcpip initialization */
00638     tcpip_adapter_init();
00639     /* for mesh
00640      * stop DHCP server on softAP interface by default
00641      * stop DHCP client on station interface by default
00642      * */
00643     ESP_ERROR_CHECK(tcpip_adapter_dhcps_stop(TCPIP_ADAPTER_IF_AP));
00644     ESP_ERROR_CHECK(tcpip_adapter_dhcpc_stop(TCPIP_ADAPTER_IF_STA));
00645 #if 0
00646     /* static ip settings */
00647     tcpip_adapter_ip_info_t sta_ip;
00648     sta_ip.ip.addr = ipaddr_addr("192.168.1.102");
00649     sta_ip.gw.addr = ipaddr_addr("192.168.1.1");
00650     sta_ip.netmask.addr = ipaddr_addr("255.255.255.0");
00651     tcpip_adapter_set_ip_info(WIFI_IF_STA, &sta_ip);
00652 #endif
00653     /*  wifi initialization */
00654     ESP_ERROR_CHECK(esp_event_loop_init(NULL, NULL));
00655     wifi_init_config_t config = WIFI_INIT_CONFIG_DEFAULT();
00656     ESP_ERROR_CHECK(esp_wifi_init(&config));
00657     ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_FLASH));
00658     ESP_ERROR_CHECK(esp_wifi_start());
00659     /*  mesh initialization */
00660     ESP_ERROR_CHECK(esp_mesh_init());
00661     ESP_ERROR_CHECK(esp_mesh_set_max_layer(CONFIG_MESH_MAX_LAYER));
00662     ESP_ERROR_CHECK(esp_mesh_set_vote_percentage(1));
00663     ESP_ERROR_CHECK(esp_mesh_set_ap_assoc_expire(10));
00664 #ifdef MESH_FIX_ROOT
00665     ESP_ERROR_CHECK(esp_mesh_fix_root(1));
00666 #endif
00667     mesh_cfg_t cfg = MESH_INIT_CONFIG_DEFAULT();
00668     /* mesh ID */
00669     memcpy((uint8_t *)&cfg.mesh_id, MESH_ID, 6);
00670     /* mesh event callback */
00671     cfg.event_cb = &mesh_event_handler;
00672     /* router */
00673     cfg.channel = CONFIG_MESH_CHANNEL;
00674     cfg.router.ssid_len = strlen(CONFIG_MESH_ROUTER_SSID);
00675     memcpy((uint8_t *)&cfg.router.ssid, CONFIG_MESH_ROUTER_SSID, cfg.router.ssid_len);
00676     memcpy((uint8_t *)&cfg.router.password, CONFIG_MESH_ROUTER_PASSWD,
00677            strlen(CONFIG_MESH_ROUTER_PASSWD));
00678     /* mesh softAP */
00679     ESP_ERROR_CHECK(esp_mesh_set_ap_authmode(CONFIG_MESH_AP_AUTHMODE));
00680     cfg.mesh_ap.max_connection = CONFIG_MESH_AP_CONNECTIONS;
00681     memcpy((uint8_t *)&cfg.mesh_ap.password, CONFIG_MESH_AP_PASSWD,
00682            strlen(CONFIG_MESH_AP_PASSWD));
00683     ESP_ERROR_CHECK(esp_mesh_set_config(&cfg));
00684     /* mesh start */
00685     ESP_ERROR_CHECK(esp_mesh_start());
00686     ESP_LOGI(MESH_TAG, "mesh starts successfully, heap:%d, %s\n", esp_get_free_heap_size(),
00687              esp_mesh_is_root_fixed() ? "root fixed" : "root not fixed");
00688 }
```

2.1.2.2 disp_info()

```
void disp_info ( )
```

Display general information on the OLED screen about status of the device. This can range from if it is the master node to connection to the mesh status.

Definition at line 201 of file mesh_main.c.

```
00202 {
00203     char str[128] = {
00204         0,
00205     };
00206     display_clear();
00207     vTaskDelay(100 / portTICK_PERIOD_MS);
00208     if (esp_mesh_is_root())
00209     {
00210         strcat(str, "MASTER NODE\n");
00211     }
00212     else
00213     {
00214         strcat(str, "\n");
00215     }
00216     strcat(str, "MESH: ");
00217     if (is_mesh_connected)
00218     {
00219         strcat(str, "CONNECT\n");
00220     }
00221     else
00222     {
00223         strcat(str, "NOT CONN\n");
00224     }
00225     printf("PRINTING... %s\n", str);
00226     xTaskCreate(&task_ssd1306_display_text, "ssd1306_display_text", 2048,
00227                 (void *)str, 6, NULL);
00228 }
```

2.1.2.3 display_clear()

```
void display_clear ( )
```

clear the OLED screen of text

Definition at line 191 of file mesh_main.c.

```
00192 {
00193     xTaskCreate(&task_ssd1306_display_text, "ssd1306_display_text", 2048,
00194                 (void *)"                \n                \n                \n                \n
                \n                \n                \n                ", 6, NULL);
00195 }
```

2.1.2.4 esp_mesh_comm_p2p_start()

```
esp_err_t esp_mesh_comm_p2p_start (
            void  )
```

Create the Mesh RX and TX tasks for the FreeRTOS to handle

Definition at line 394 of file mesh_main.c.

```
00395 {
00396     static bool is_comm_p2p_started = false;
00397     if (!is_comm_p2p_started)
00398     {
00399         is_comm_p2p_started = true;
00400         xTaskCreate(esp_mesh_p2p_tx_main, "MPTX", 3072, NULL, 5, NULL);
00401         xTaskCreate(esp_mesh_p2p_rx_main, "MPRX", 3072, NULL, 5, NULL);
00402     }
00403     return ESP_OK;
00404 }
```

2.1.2.5 esp_mesh_p2p_rx_main()

```
void esp_mesh_p2p_rx_main (
            void * arg )
```

Task that collects any data sent over the mesh to this particular data.

Parameters

arg	any mesh arguments needed

Definition at line 289 of file mesh_main.c.

```
00290 {
00291     printf("RX START\n");
00292     int recv_count = 0;
00293     esp_err_t err;
00294     mesh_addr_t from;
00295     int send_count = 0;
00296     mesh_data_t data;
00297     int flag = 0;
00298     data.data = rx_buf;
00299     data.size = RX_SIZE;
00300 
00301     // Set master address to static
00302     mesh_addr_t serverAddr;
00303     IP4_ADDR(&serverAddr.mip.ip4, 192, 168, 0, 103); // 192.168.0.103 is my pc
00304     serverAddr.mip.port = 3000;                      // Server Port
00305 
00306     // Setup socket to GUI to send data to
00307     struct sockaddr_in tcpServerAddr;
00308     tcpServerAddr.sin_addr.s_addr = inet_addr(MESH_SERVER_IP);
00309     tcpServerAddr.sin_family = AF_INET;
00310     tcpServerAddr.sin_port = htons(MESH_PORT);
00311 
00312     is_running = true;
00313 
00314     int s;
00315     while (is_running)
00316     {
00317         printf("RX PHASE\n");
00318         data.size = RX_SIZE;
00319 
00320         // Gather the data sent to it from other nodes
00321         err = esp_mesh_recv(&from, &data, portMAX_DELAY, &flag, NULL, 0);
00322         if (err != ESP_OK || !data.size)
00323         {
00324             ESP_LOGE(MESH_TAG, "err:0x%x, size:%d", err, data.size);
00325             continue;
00326         }
00327 
00328         // If the node is root (master)
00329         if (esp_mesh_is_root())
00330         {
00331             ESP_LOGI(MESH_TAG, "*** I AM ROOT ***");
00332             uint8_t numOfDists = data.data[0];
00333             uint16_t minid = data.data[1] | (data.data[2] << 8);
00334             // Find minid in node order
00335             uint8_t idloc = getnodeorder(minid);
00336 
00337             // parse through the recieved data and structure the data for easy computing
00338             for (uint8_t i = 0; i < numOfDists; i++)
00339             {
00340                 setDistance(idloc, data.data[3 + 7 * i] | (data.data[4 + 7 * i] << 8), data.data[5 + 7
    * i] | (data.data[6 + 7 * i] << 8) | (data.data[7 + 7 * i] << 16) | (data.data[8 + 7 * i] << 24));
00341             }
00342 
00343             // compute the coordinates from the distance data
00344             get_coords();
00345 
00346             // Print Distance Table
00347             printf("Distance Table\n");
00348             for (uint8_t i = 0; i < 4; i++)
00349             {
00350                 for (uint8_t j = 0; j < 4; j++)
00351                 {
00352                     printf(" %f,\t ", dist[i][j]);
00353                 }
00354                 printf("\n");
00355             }
00356             printf("XYZ Size = %d\n", xyz.size);
00357 
00358             // Print Coords
00359             printf("Node 1: x = %f ,y = %f, z = %f\n", xyz.n1.x, xyz.n1.y, xyz.n1.z);
00360             printf("Node 2: x = %f ,y = %f, z = %f\n", xyz.n2.x, xyz.n2.y, xyz.n2.z);
00361             printf("Node 3: x = %f ,y = %f, z = %f\n", xyz.n3.x, xyz.n3.y, xyz.n3.z);
00362             printf("Node 4: x = %f ,y = %f, z = %f\n", xyz.n4.x, xyz.n4.y, xyz.n4.z);
00363 
00364             // Establish Socket connection
00365             s = socket(AF_INET, SOCK_STREAM, 0);
00366             connect(s, (struct sockaddr *)&tcpServerAddr, sizeof(tcpServerAddr));
```

```
00367
00368             // Format data for GUI
00369             char str[110];
00370             sprintf(str, "%f,%f,%f;%f,%f,%f;%f,%f,%f;%f,%f,%f", xyz.n1.x, xyz.n1.y, xyz.n1.z,
      xyz.n2.x, xyz.n2.y, xyz.n2.z, xyz.n3.x, xyz.n3.y, xyz.n3.z, xyz.n4.x, xyz.n4.y, xyz.n4.z);
00371
00372
00373             // Send data to the GUI. If GUI is not connected, memory overflow errors may appear
00374             write(s, str, strlen(str));
00375             close(s);
00376
00377             // Print Node Order
00378             printf("\nNode Order\n");
00379             for (uint8_t i = 0; i < 4; i++)
00380             {
00381                 printf(" %d,\t ", nodeorder[i]);
00382             }
00383             printf("\n");
00384         }
00385         // Update OLED Screen with latest information
00386         disp_info();
00387     }
00388     vTaskDelete(NULL);
00389 }
```

2.1.2.6 esp_mesh_p2p_tx_main()

```
void esp_mesh_p2p_tx_main (
            void * arg )
```

Task to send data over the mesh. In particular it takes the data recieved by the decawave and sends it to the master node.

Definition at line 233 of file mesh_main.c.

```
00234 {
00235     int i;
00236     esp_err_t err;
00237     int send_count = 0;
00238     mesh_addr_t route_table[CONFIG_MESH_ROUTE_TABLE_SIZE];
00239     int route_table_size = 0;
00240     mesh_data_t data;
00241     data.data = tx_buf;
00242     data.size = sizeof(tx_buf);
00243     data.proto = MESH_PROTO_BIN;
00244
00245     is_running = true;
00246     // This task should always be running
00247     while (is_running)
00248     {
00249         // If root: gets routing table and updates local
00250         if (esp_mesh_is_root())
00251             esp_mesh_get_routing_table((mesh_addr_t *)&route_table, CONFIG_MESH_ROUTE_TABLE_SIZE * 6,
      &route_table_size);
00252
00253         // Everyone does this - Gets an array of distances from the decawave
00254         int errr = dwm_tag_loc_get(&tx_buf, NODE_TYPE_TAG);
00255         printf("err = %d\n", errr);
00256
00257         // If there is no error
00258         if (errr != -1)
00259         {
00260             // Send a msg to the root in the mesh (master) with the decawave data
00261             err = esp_mesh_send(&mesh_root_addr, &data, MESH_DATA_P2P, NULL, 0);
00262             if (err)
00263                 ESP_LOGE(MESH_TAG, "ERROR ON TRANSMIT");
00264         }
00265
00266         // Delay 100ms
00267         vTaskDelay(1 * 100 / portTICK_RATE_MS);
00268
00269         // If the node is root node make sure it doesn't use too much comutational energy with this
      task and slow it down.
00270         if (esp_mesh_is_root())
00271         {
00272             ESP_LOGI(MESH_TAG, "*** I AM ROOT ***");
00273             /* if route_table_size is less than 10, add delay to avoid watchdog in this task. */
00274             if (route_table_size < 10)
00275             {
00276                 vTaskDelay(1 * 100 / portTICK_RATE_MS);
```

```
00277             }
00278         }
00279     }
00280
00281     // If the program gets here, release task from memory or the task queue
00282     vTaskDelete(NULL);
00283 }
```

2.1.2.7 i2c_master_init()

```
void i2c_master_init ( )
```

I2C Start Master at 1Mhz

Definition at line 80 of file mesh_main.c.

```
00081 {
00082     i2c_config_t i2c_config = {
00083         .mode = I2C_MODE_MASTER,
00084         .sda_io_num = SDA_PIN,
00085         .scl_io_num = SCL_PIN,
00086         .sda_pullup_en = GPIO_PULLUP_ENABLE,
00087         .scl_pullup_en = GPIO_PULLUP_ENABLE,
00088         .master.clk_speed = 1000000};
00089     i2c_param_config(I2C_NUM_0, &i2c_config);
00090     i2c_driver_install(I2C_NUM_0, I2C_MODE_MASTER, 0, 0, 0);
00091 }
```

2.1.2.8 mesh_event_handler()

```
void mesh_event_handler (
             mesh_event_t event )
```

Mesh Event handler handles all status changes of the ESP If the node is root node, set it for the entire node to use that information to handle seperate tasks and to showcase it on the OLED screen

Definition at line 410 of file mesh_main.c.

```
00411 {
00412     mesh_addr_t id = {
00413         0,
00414     };
00415     static uint8_t last_layer = 0;
00416     ESP_LOGD(MESH_TAG, "esp_event_handler:%d", event.id);
00417
00418     switch (event.id)
00419     {
00420     case MESH_EVENT_STARTED:
00421         esp_mesh_get_id(&id);
00422         ESP_LOGI(MESH_TAG, "<MESH_EVENT_STARTED>ID:" MACSTR "", MAC2STR(id.addr));
00423         is_mesh_connected = false;
00424         mesh_layer = esp_mesh_get_layer();
00425         break;
00426     case MESH_EVENT_STOPPED:
00427         ESP_LOGI(MESH_TAG, "<MESH_EVENT_STOPPED>");
00428         is_mesh_connected = false;
00429         mesh_layer = esp_mesh_get_layer();
00430         break;
00431     case MESH_EVENT_CHILD_CONNECTED:
00432         ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHILD_CONNECTED>aid:%d, " MACSTR "",
00433                  event.info.child_connected.aid,
00434                  MAC2STR(event.info.child_connected.mac));
00435         break;
00436     case MESH_EVENT_CHILD_DISCONNECTED:
00437         ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHILD_DISCONNECTED>aid:%d, " MACSTR "",
00438                  event.info.child_disconnected.aid,
00439                  MAC2STR(event.info.child_disconnected.mac));
00440         break;
00441     case MESH_EVENT_ROUTING_TABLE_ADD:
00442         ESP_LOGW(MESH_TAG, "<MESH_EVENT_ROUTING_TABLE_ADD>add %d, new:%d",
00443                  event.info.routing_table.rt_size_change,
00444                  event.info.routing_table.rt_size_new);
00445         break;
```

```
00446     case MESH_EVENT_ROUTING_TABLE_REMOVE:
00447         ESP_LOGW(MESH_TAG, "<MESH_EVENT_ROUTING_TABLE_REMOVE>remove %d, new:%d",
00448                  event.info.routing_table.rt_size_change,
00449                  event.info.routing_table.rt_size_new);
00450         break;
00451     case MESH_EVENT_NO_PARENT_FOUND:
00452         ESP_LOGI(MESH_TAG, "<MESH_EVENT_NO_PARENT_FOUND>scan times:%d",
00453                  event.info.no_parent.scan_times);
00454         /* TODO handler for the failure */
00455         break;
00456     case MESH_EVENT_PARENT_CONNECTED:
00457         esp_mesh_get_id(&id);
00458         mesh_layer = event.info.connected.self_layer;
00459         memcpy(&mesh_parent_addr.addr, event.info.connected.connected.bssid, 6);
00460         ESP_LOGI(MESH_TAG,
00461                  "<MESH_EVENT_PARENT_CONNECTED>layer:%d-->%d, parent:" MACSTR "%s, ID:" MACSTR "",
00462                  last_layer, mesh_layer, MAC2STR(mesh_parent_addr.addr),
00463                  esp_mesh_is_root() ? "<ROOT>" : (mesh_layer == 2) ? "<layer2>" : "",
      MAC2STR(id.addr));
00464         last_layer = mesh_layer;
00465         // mesh_connected_indicator(mesh_layer);
00466         is_mesh_connected = true;
00467         if (esp_mesh_is_root())
00468         {
00469             tcpip_adapter_dhcpc_start(TCPIP_ADAPTER_IF_STA);
00470         }
00471         esp_mesh_comm_p2p_start();
00472         break;
00473     case MESH_EVENT_PARENT_DISCONNECTED:
00474         ESP_LOGI(MESH_TAG,
00475                  "<MESH_EVENT_PARENT_DISCONNECTED>reason:%d",
00476                  event.info.disconnected.reason);
00477         is_mesh_connected = false;
00478         // mesh_disconnected_indicator();
00479         mesh_layer = esp_mesh_get_layer();
00480         break;
00481     case MESH_EVENT_LAYER_CHANGE:
00482         mesh_layer = event.info.layer_change.new_layer;
00483         ESP_LOGI(MESH_TAG, "<MESH_EVENT_LAYER_CHANGE>layer:%d-->%d%s",
00484                  last_layer, mesh_layer,
00485                  esp_mesh_is_root() ? "<ROOT>" : (mesh_layer == 2) ? "<layer2>" : "");
00486         last_layer = mesh_layer;
00487         // mesh_connected_indicator(mesh_layer);
00488         break;
00489     case MESH_EVENT_ROOT_ADDRESS:
00490         ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_ADDRESS>root address:" MACSTR "",
00491                  MAC2STR(event.info.root_addr.addr));
00492         mesh_root_addr = event.info.root_addr;
00493
00494         break;
00495     case MESH_EVENT_ROOT_GOT_IP:
00496         /* root starts to connect to server */
00497         ESP_LOGI(MESH_TAG,
00498                  "<MESH_EVENT_ROOT_GOT_IP>sta ip: " IPSTR ", mask: " IPSTR ", gw: " IPSTR,
00499                  IP2STR(&event.info.got_ip.ip_info.ip),
00500                  IP2STR(&event.info.got_ip.ip_info.netmask),
00501                  IP2STR(&event.info.got_ip.ip_info.gw));
00502         gotIP = true;
00503         break;
00504     case MESH_EVENT_ROOT_LOST_IP:
00505         ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_LOST_IP>");
00506         break;
00507     case MESH_EVENT_VOTE_STARTED:
00508         ESP_LOGI(MESH_TAG,
00509                  "<MESH_EVENT_VOTE_STARTED>attempts:%d, reason:%d, rc_addr:" MACSTR "",
00510                  event.info.vote_started.attempts,
00511                  event.info.vote_started.reason,
00512                  MAC2STR(event.info.vote_started.rc_addr.addr));
00513         break;
00514     case MESH_EVENT_VOTE_STOPPED:
00515         ESP_LOGI(MESH_TAG, "<MESH_EVENT_VOTE_STOPPED>");
00516         break;
00517     case MESH_EVENT_ROOT_SWITCH_REQ:
00518         ESP_LOGI(MESH_TAG,
00519                  "<MESH_EVENT_ROOT_SWITCH_REQ>reason:%d, rc_addr:" MACSTR "",
00520                  event.info.switch_req.reason,
00521                  MAC2STR(event.info.switch_req.rc_addr.addr));
00522         break;
00523     case MESH_EVENT_ROOT_SWITCH_ACK:
00524         /* new root */
00525         mesh_layer = esp_mesh_get_layer();
00526         esp_mesh_get_parent_bssid(&mesh_parent_addr);
00527         ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_SWITCH_ACK>layer:%d, parent:" MACSTR "", mesh_layer,
      MAC2STR(mesh_parent_addr.addr));
00528         break;
00529     case MESH_EVENT_TODS_STATE:
00530         ESP_LOGI(MESH_TAG, "<MESH_EVENT_TODS_REACHABLE>state:%d",
```

```
00531                 event.info.toDS_state);
00532         break;
00533     case MESH_EVENT_ROOT_FIXED:
00534         ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_FIXED>%s",
00535                 event.info.root_fixed.is_fixed ? "fixed" : "not fixed");
00536         break;
00537     case MESH_EVENT_ROOT_ASKED_YIELD:
00538         ESP_LOGI(MESH_TAG,
00539                 "<MESH_EVENT_ROOT_ASKED_YIELD>" MACSTR ", rssi:%d, capacity:%d",
00540                 MAC2STR(event.info.root_conflict.addr),
00541                 event.info.root_conflict.rssi,
00542                 event.info.root_conflict.capacity);
00543         break;
00544     case MESH_EVENT_CHANNEL_SWITCH:
00545         ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHANNEL_SWITCH>new channel:%d",
       event.info.channel_switch.channel);
00546         break;
00547     case MESH_EVENT_SCAN_DONE:
00548         ESP_LOGI(MESH_TAG, "<MESH_EVENT_SCAN_DONE>number:%d",
00549                 event.info.scan_done.number);
00550         break;
00551     case MESH_EVENT_NETWORK_STATE:
00552         ESP_LOGI(MESH_TAG, "<MESH_EVENT_NETWORK_STATE>is_rootless:%d",
00553                 event.info.network_state.is_rootless);
00554         break;
00555     case MESH_EVENT_STOP_RECONNECTION:
00556         ESP_LOGI(MESH_TAG, "<MESH_EVENT_STOP_RECONNECTION>");
00557         break;
00558     case MESH_EVENT_FIND_NETWORK:
00559         ESP_LOGI(MESH_TAG, "<MESH_EVENT_FIND_NETWORK>new channel:%d, router BSSID:" MACSTR "",
00560                 event.info.find_network.channel, MAC2STR(event.info.find_network.router_bssid));
00561         break;
00562     case MESH_EVENT_ROUTER_SWITCH:
00563         ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROUTER_SWITCH>new router:%s, channel:%d, " MACSTR "",
00564                 event.info.router_switch.ssid, event.info.router_switch.channel,
       MAC2STR(event.info.router_switch.bssid));
00565         break;
00566     default:
00567         ESP_LOGI(MESH_TAG, "unknown id:%d", event.id);
00568         break;
00569     }
00570 }
```

2.1.2.9 ssd1306_init()

```
void ssd1306_init ( )
```

Initialize OLED Screen using I2C

Definition at line 96 of file mesh_main.c.

```
00097 {
00098     esp_err_t espRc;
00099
00100     i2c_cmd_handle_t cmd = i2c_cmd_link_create();
00101
00102     i2c_master_start(cmd);
00103     i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true);
00104     i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_STREAM, true);
00105
00106     i2c_master_write_byte(cmd, OLED_CMD_SET_CHARGE_PUMP, true);
00107     i2c_master_write_byte(cmd, 0x14, true);
00108
00109     i2c_master_write_byte(cmd, OLED_CMD_SET_SEGMENT_REMAP, true); // reverse left-right mapping
00110     i2c_master_write_byte(cmd, OLED_CMD_SET_COM_SCAN_MODE, true); // reverse up-bottom mapping
00111
00112     i2c_master_write_byte(cmd, OLED_CMD_DISPLAY_ON, true);
00113     i2c_master_stop(cmd);
00114
00115     espRc = i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
00116     if (espRc == ESP_OK)
00117     {
00118         ESP_LOGI(tag, "OLED configured successfully");
00119     }
00120     else
00121     {
00122         ESP_LOGE(tag, "OLED configuration failed. code: 0x%.2X", espRc);
00123     }
00124     i2c_cmd_link_delete(cmd);
00125 }
```

2.1.2.10 task_ssd1306_display_text()

```
void task_ssd1306_display_text (
            const void * arg_text )
```

Display Text on the OLED screen

Parameters

arg_text	string with text to display

Definition at line 131 of file mesh_main.c.

```
00132 {
00133     char *text = (char *)arg_text;
00134     uint8_t text_len = strlen(text);
00135 
00136     i2c_cmd_handle_t cmd;
00137 
00138     uint8_t cur_page = 0;
00139 
00140     cmd = i2c_cmd_link_create();
00141     i2c_master_start(cmd);
00142     i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true);
00143 
00144     i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_STREAM, true);
00145     i2c_master_write_byte(cmd, 0x00, true); // reset column
00146     i2c_master_write_byte(cmd, 0x10, true);
00147     i2c_master_write_byte(cmd, 0xB0 | cur_page, true); // reset page
00148 
00149     i2c_master_stop(cmd);
00150     i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
00151     i2c_cmd_link_delete(cmd);
00152 
00153     for (uint8_t i = 0; i < text_len; i++)
00154     {
00155         if (text[i] == '\n')
00156         {
00157             cmd = i2c_cmd_link_create();
00158             i2c_master_start(cmd);
00159             i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true);
00160 
00161             i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_STREAM, true);
00162             i2c_master_write_byte(cmd, 0x00, true); // reset column
00163             i2c_master_write_byte(cmd, 0x10, true);
00164             i2c_master_write_byte(cmd, 0xB0 | ++cur_page, true); // increment page
00165 
00166             i2c_master_stop(cmd);
00167             i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
00168             i2c_cmd_link_delete(cmd);
00169         }
00170         else
00171         {
00172             cmd = i2c_cmd_link_create();
00173             i2c_master_start(cmd);
00174             i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true);
00175 
00176             i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_DATA_STREAM, true);
00177             i2c_master_write(cmd, font8x8_basic_tr[(uint8_t)text[i]], 8, true);
00178 
00179             i2c_master_stop(cmd);
00180             i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
00181             i2c_cmd_link_delete(cmd);
00182         }
00183     }
00184 
00185     vTaskDelete(NULL);
00186 }
```

2.2 mesh_main.c

```
00001 
00002 /*
00003 Mesh Main
00004 Anmol Modur
```

```
00005 3/15/19
00006 */
00007
00008 /* --- Include Libraries --- */
00009 #include <string.h>
00010 // Mesh Libraries
00011 #include "esp_wifi.h"
00012 #include "esp_system.h"
00013 #include "esp_event_loop.h"
00014 #include "esp_log.h"
00015 #include "esp_mesh.h"
00016 #include "esp_mesh_internal.h"
00017 // #include "mesh_light.h"
00018 #include "nvs_flash.h"
00019 // Decawave Libraries
00020 #include "dwm1001.h"
00021 #include "esp_nvs.h"
00022 #include "esp_interrupt.h"
00023 #include "triangulation.h"
00024 // Lwip Libraries
00025 #include "lwip/err.h"
00026 #include "lwip/sockets.h"
00027 #include "lwip/sys.h"
00028 #include "lwip/netdb.h"
00029 #include "lwip/dns.h"
00030
00031 #include "driver/gpio.h"
00032 #include "driver/i2c.h"
00033 #include "ssd1366.h"
00034 #include "font8x8_basic.h"
00035 /*******************************************************
00036  *                Macros
00037  *******************************************************/
00038 //#define MESH_P2P_TOS_OFF
00039
00040 /*******************************************************
00041  *                Constants
00042  *******************************************************/
00043 #define RX_SIZE (1500)
00044 #define TX_SIZE (1460)
00045 #define MESH_SERVER_IP "192.168.0.103"
00046 #define MESH_PORT 3000
00047 #define SDA_PIN GPIO_NUM_21
00048 #define SCL_PIN GPIO_NUM_22
00049 #define tag "SSD1306"
00050 /*******************************************************
00051  *                Variable Definitions
00052  *******************************************************/
00053 static const char *MESH_TAG = "mesh_main";
00054 static const uint8_t MESH_ID[6] = {0x77, 0x77, 0x77, 0x77, 0x77, 0x77};
00055 static uint8_t tx_buf[TX_SIZE] = {
00056     0,
00057 };
00058 static uint8_t rx_buf[RX_SIZE] = {
00059     0,
00060 };
00061 static bool is_running = true;
00062 static bool is_mesh_connected = false;
00063 static mesh_addr_t mesh_parent_addr;
00064 static int mesh_layer = -1;
00065 static mesh_addr_t mesh_root_addr;
00066 static dwm_resp_tag_loc_get loc;
00067 static bool gotIP = false;
00068
00069 /*******************************************************
00070  *                Function Declarations
00071  *******************************************************/
00072
00073 /*******************************************************
00074  *                Function Definitions
00075  *******************************************************/
00076
00080 void i2c_master_init()
00081 {
00082     i2c_config_t i2c_config = {
00083         .mode = I2C_MODE_MASTER,
00084         .sda_io_num = SDA_PIN,
00085         .scl_io_num = SCL_PIN,
00086         .sda_pullup_en = GPIO_PULLUP_ENABLE,
00087         .scl_pullup_en = GPIO_PULLUP_ENABLE,
00088         .master.clk_speed = 1000000};
00089     i2c_param_config(I2C_NUM_0, &i2c_config);
00090     i2c_driver_install(I2C_NUM_0, I2C_MODE_MASTER, 0, 0, 0);
00091 }
00092
00096 void ssd1306_init()
00097 {
```

```
00098     esp_err_t espRc;
00099
00100     i2c_cmd_handle_t cmd = i2c_cmd_link_create();
00101
00102     i2c_master_start(cmd);
00103     i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS « 1) | I2C_MASTER_WRITE, true);
00104     i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_STREAM, true);
00105
00106     i2c_master_write_byte(cmd, OLED_CMD_SET_CHARGE_PUMP, true);
00107     i2c_master_write_byte(cmd, 0x14, true);
00108
00109     i2c_master_write_byte(cmd, OLED_CMD_SET_SEGMENT_REMAP, true); // reverse left-right mapping
00110     i2c_master_write_byte(cmd, OLED_CMD_SET_COM_SCAN_MODE, true); // reverse up-bottom mapping
00111
00112     i2c_master_write_byte(cmd, OLED_CMD_DISPLAY_ON, true);
00113     i2c_master_stop(cmd);
00114
00115     espRc = i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
00116     if (espRc == ESP_OK)
00117     {
00118         ESP_LOGI(tag, "OLED configured successfully");
00119     }
00120     else
00121     {
00122         ESP_LOGE(tag, "OLED configuration failed. code: 0x%.2X", espRc);
00123     }
00124     i2c_cmd_link_delete(cmd);
00125 }
00126
00131 void task_ssd1306_display_text(const void *arg_text)
00132 {
00133     char *text = (char *)arg_text;
00134     uint8_t text_len = strlen(text);
00135
00136     i2c_cmd_handle_t cmd;
00137
00138     uint8_t cur_page = 0;
00139
00140     cmd = i2c_cmd_link_create();
00141     i2c_master_start(cmd);
00142     i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS « 1) | I2C_MASTER_WRITE, true);
00143
00144     i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_STREAM, true);
00145     i2c_master_write_byte(cmd, 0x00, true); // reset column
00146     i2c_master_write_byte(cmd, 0x10, true);
00147     i2c_master_write_byte(cmd, 0xB0 | cur_page, true); // reset page
00148
00149     i2c_master_stop(cmd);
00150     i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
00151     i2c_cmd_link_delete(cmd);
00152
00153     for (uint8_t i = 0; i < text_len; i++)
00154     {
00155         if (text[i] == '\n')
00156         {
00157             cmd = i2c_cmd_link_create();
00158             i2c_master_start(cmd);
00159             i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS « 1) | I2C_MASTER_WRITE, true);
00160
00161             i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_STREAM, true);
00162             i2c_master_write_byte(cmd, 0x00, true); // reset column
00163             i2c_master_write_byte(cmd, 0x10, true);
00164             i2c_master_write_byte(cmd, 0xB0 | ++cur_page, true); // increment page
00165
00166             i2c_master_stop(cmd);
00167             i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
00168             i2c_cmd_link_delete(cmd);
00169         }
00170         else
00171         {
00172             cmd = i2c_cmd_link_create();
00173             i2c_master_start(cmd);
00174             i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS « 1) | I2C_MASTER_WRITE, true);
00175
00176             i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_DATA_STREAM, true);
00177             i2c_master_write(cmd, font8x8_basic_tr[(uint8_t)text[i]], 8, true);
00178
00179             i2c_master_stop(cmd);
00180             i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
00181             i2c_cmd_link_delete(cmd);
00182         }
00183     }
00184
00185     vTaskDelete(NULL);
00186 }
00187
00191 void display_clear()
```

```
00192 {
00193     xTaskCreate(&task_ssd1306_display_text, "ssd1306_display_text", 2048,
00194                 (void *)"                 \n                \n                \n                \n                \n
              \n                \n                \n                ", 6, NULL);
00195 }
00196
00201 void disp_info()
00202 {
00203     char str[128] = {
00204         0,
00205     };
00206     display_clear();
00207     vTaskDelay(100 / portTICK_PERIOD_MS);
00208     if (esp_mesh_is_root())
00209     {
00210         strcat(str, "MASTER NODE\n");
00211     }
00212     else
00213     {
00214         strcat(str, "\n");
00215     }
00216     strcat(str, "MESH: ");
00217     if (is_mesh_connected)
00218     {
00219         strcat(str, "CONNECT\n");
00220     }
00221     else
00222     {
00223         strcat(str, "NOT CONN\n");
00224     }
00225     printf("PRINTING... %s\n", str);
00226     xTaskCreate(&task_ssd1306_display_text, "ssd1306_display_text", 2048,
00227                 (void *)str, 6, NULL);
00228 }
00229
00233 void esp_mesh_p2p_tx_main(void *arg)
00234 {
00235     int i;
00236     esp_err_t err;
00237     int send_count = 0;
00238     mesh_addr_t route_table[CONFIG_MESH_ROUTE_TABLE_SIZE];
00239     int route_table_size = 0;
00240     mesh_data_t data;
00241     data.data = tx_buf;
00242     data.size = sizeof(tx_buf);
00243     data.proto = MESH_PROTO_BIN;
00244
00245     is_running = true;
00246     // This task should always be running
00247     while (is_running)
00248     {
00249         // If root: gets routing table and updates local
00250         if (esp_mesh_is_root())
00251             esp_mesh_get_routing_table((mesh_addr_t *)&route_table, CONFIG_MESH_ROUTE_TABLE_SIZE * 6,
      &route_table_size);
00252
00253         // Everyone does this - Gets an array of distances from the decawave
00254         int errr = dwm_tag_loc_get(&tx_buf, NODE_TYPE_TAG);
00255         printf("err = %d\n", errr);
00256
00257         // If there is no error
00258         if (errr != -1)
00259         {
00260             // Send a msg to the root in the mesh (master) with the decawave data
00261             err = esp_mesh_send(&mesh_root_addr, &data, MESH_DATA_P2P, NULL, 0);
00262             if (err)
00263                 ESP_LOGE(MESH_TAG, "ERROR ON TRANSMIT");
00264         }
00265
00266         // Delay 100ms
00267         vTaskDelay(1 * 100 / portTICK_RATE_MS);
00268
00269         // If the node is root node make sure it doesn't use too much comutational energy with this
      task and slow it down.
00270         if (esp_mesh_is_root())
00271         {
00272             ESP_LOGI(MESH_TAG, "*** I AM ROOT ***");
00273             /* if route_table_size is less than 10, add delay to avoid watchdog in this task. */
00274             if (route_table_size < 10)
00275             {
00276                 vTaskDelay(1 * 100 / portTICK_RATE_MS);
00277             }
00278         }
00279     }
00280
00281     // If the program gets here, release task from memory or the task queue
00282     vTaskDelete(NULL);
```

```
00283 }
00284
00289 void esp_mesh_p2p_rx_main(void *arg)
00290 {
00291     printf("RX START\n");
00292     int recv_count = 0;
00293     esp_err_t err;
00294     mesh_addr_t from;
00295     int send_count = 0;
00296     mesh_data_t data;
00297     int flag = 0;
00298     data.data = rx_buf;
00299     data.size = RX_SIZE;
00300
00301     // Set master address to static
00302     mesh_addr_t serverAddr;
00303     IP4_ADDR(&serverAddr.mip.ip4, 192, 168, 0, 103); // 192.168.0.103 is my pc
00304     serverAddr.mip.port = 3000;                      // Server Port
00305
00306     // Setup socket to GUI to send data to
00307     struct sockaddr_in tcpServerAddr;
00308     tcpServerAddr.sin_addr.s_addr = inet_addr(MESH_SERVER_IP);
00309     tcpServerAddr.sin_family = AF_INET;
00310     tcpServerAddr.sin_port = htons(MESH_PORT);
00311
00312     is_running = true;
00313
00314     int s;
00315     while (is_running)
00316     {
00317         printf("RX PHASE\n");
00318         data.size = RX_SIZE;
00319
00320         // Gather the data sent to it from other nodes
00321         err = esp_mesh_recv(&from, &data, portMAX_DELAY, &flag, NULL, 0);
00322         if (err != ESP_OK || !data.size)
00323         {
00324             ESP_LOGE(MESH_TAG, "err:0x%x, size:%d", err, data.size);
00325             continue;
00326         }
00327
00328         // If the node is root (master)
00329         if (esp_mesh_is_root())
00330         {
00331             ESP_LOGI(MESH_TAG, "*** I AM ROOT ***");
00332             uint8_t numOfDists = data.data[0];
00333             uint16_t minid = data.data[1] | (data.data[2] << 8);
00334             // Find minid in node order
00335             uint8_t idloc = getnodeorder(minid);
00336
00337             // parse through the recieved data and structure the data for easy computing
00338             for (uint8_t i = 0; i < numOfDists; i++)
00339             {
00340                 setDistance(idloc, data.data[3 + 7 * i] | (data.data[4 + 7 * i] << 8), data.data[5 + 7
    * i] | (data.data[6 + 7 * i] << 8) | (data.data[7 + 7 * i] << 16) | (data.data[8 + 7 * i] << 24));
00341             }
00342
00343             // compute the coordinates from the distance data
00344             get_coords();
00345
00346             // Print Distance Table
00347             printf("Distance Table\n");
00348             for (uint8_t i = 0; i < 4; i++)
00349             {
00350                 for (uint8_t j = 0; j < 4; j++)
00351                 {
00352                     printf(" %f,\t ", dist[i][j]);
00353                 }
00354                 printf("\n");
00355             }
00356             printf("XYZ Size = %d\n", xyz.size);
00357
00358             // Print Coords
00359             printf("Node 1: x = %f ,y = %f, z = %f\n", xyz.n1.x, xyz.n1.y, xyz.n1.z);
00360             printf("Node 2: x = %f ,y = %f, z = %f\n", xyz.n2.x, xyz.n2.y, xyz.n2.z);
00361             printf("Node 3: x = %f ,y = %f, z = %f\n", xyz.n3.x, xyz.n3.y, xyz.n3.z);
00362             printf("Node 4: x = %f ,y = %f, z = %f\n", xyz.n4.x, xyz.n4.y, xyz.n4.z);
00363
00364             // Establish Socket connection
00365             s = socket(AF_INET, SOCK_STREAM, 0);
00366             connect(s, (struct sockaddr *)&tcpServerAddr, sizeof(tcpServerAddr));
00367
00368             // Format data for GUI
00369             char str[110];
00370             sprintf(str, "%f,%f,%f;%f,%f,%f;%f,%f,%f;%f,%f,%f", xyz.n1.x, xyz.n1.y, xyz.n1.z,
    xyz.n2.x, xyz.n2.y, xyz.n2.z, xyz.n3.x, xyz.n3.y, xyz.n3.z, xyz.n4.x, xyz.n4.y, xyz.n4.z);
00371
```

```
00372
00373             // Send data to the GUI. If GUI is not connected, memory overflow errors may appear
00374             write(s, str, strlen(str));
00375             close(s);
00376
00377             // Print Node Order
00378             printf("\nNode Order\n");
00379             for (uint8_t i = 0; i < 4; i++)
00380             {
00381                 printf(" %d,\t ", nodeorder[i]);
00382             }
00383             printf("\n");
00384         }
00385         // Update OLED Screen with latest information
00386         disp_info();
00387     }
00388     vTaskDelete(NULL);
00389 }
00390
00394 esp_err_t esp_mesh_comm_p2p_start(void)
00395 {
00396     static bool is_comm_p2p_started = false;
00397     if (!is_comm_p2p_started)
00398     {
00399         is_comm_p2p_started = true;
00400         xTaskCreate(esp_mesh_p2p_tx_main, "MPTX", 3072, NULL, 5, NULL);
00401         xTaskCreate(esp_mesh_p2p_rx_main, "MPRX", 3072, NULL, 5, NULL);
00402     }
00403     return ESP_OK;
00404 }
00405
00410 void mesh_event_handler(mesh_event_t event)
00411 {
00412     mesh_addr_t id = {
00413         0,
00414     };
00415     static uint8_t last_layer = 0;
00416     ESP_LOGD(MESH_TAG, "esp_event_handler:%d", event.id);
00417
00418     switch (event.id)
00419     {
00420     case MESH_EVENT_STARTED:
00421         esp_mesh_get_id(&id);
00422         ESP_LOGI(MESH_TAG, "<MESH_EVENT_STARTED>ID:" MACSTR "", MAC2STR(id.addr));
00423         is_mesh_connected = false;
00424         mesh_layer = esp_mesh_get_layer();
00425         break;
00426     case MESH_EVENT_STOPPED:
00427         ESP_LOGI(MESH_TAG, "<MESH_EVENT_STOPPED>");
00428         is_mesh_connected = false;
00429         mesh_layer = esp_mesh_get_layer();
00430         break;
00431     case MESH_EVENT_CHILD_CONNECTED:
00432         ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHILD_CONNECTED>aid:%d, " MACSTR "",
00433                  event.info.child_connected.aid,
00434                  MAC2STR(event.info.child_connected.mac));
00435         break;
00436     case MESH_EVENT_CHILD_DISCONNECTED:
00437         ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHILD_DISCONNECTED>aid:%d, " MACSTR "",
00438                  event.info.child_disconnected.aid,
00439                  MAC2STR(event.info.child_disconnected.mac));
00440         break;
00441     case MESH_EVENT_ROUTING_TABLE_ADD:
00442         ESP_LOGW(MESH_TAG, "<MESH_EVENT_ROUTING_TABLE_ADD>add %d, new:%d",
00443                  event.info.routing_table.rt_size_change,
00444                  event.info.routing_table.rt_size_new);
00445         break;
00446     case MESH_EVENT_ROUTING_TABLE_REMOVE:
00447         ESP_LOGW(MESH_TAG, "<MESH_EVENT_ROUTING_TABLE_REMOVE>remove %d, new:%d",
00448                  event.info.routing_table.rt_size_change,
00449                  event.info.routing_table.rt_size_new);
00450         break;
00451     case MESH_EVENT_NO_PARENT_FOUND:
00452         ESP_LOGI(MESH_TAG, "<MESH_EVENT_NO_PARENT_FOUND>scan times:%d",
00453                  event.info.no_parent.scan_times);
00454         /* TODO handler for the failure */
00455         break;
00456     case MESH_EVENT_PARENT_CONNECTED:
00457         esp_mesh_get_id(&id);
00458         mesh_layer = event.info.connected.self_layer;
00459         memcpy(&mesh_parent_addr.addr, event.info.connected.connected.bssid, 6);
00460         ESP_LOGI(MESH_TAG,
00461                  "<MESH_EVENT_PARENT_CONNECTED>layer:%d-->%d, parent:" MACSTR "%s, ID:" MACSTR "",
00462                  last_layer, mesh_layer, MAC2STR(mesh_parent_addr.addr),
00463                  esp_mesh_is_root() ? "<ROOT>" : (mesh_layer == 2) ? "<layer2>" : "",
    MAC2STR(id.addr));
00464         last_layer = mesh_layer;
```

```
00465         // mesh_connected_indicator(mesh_layer);
00466         is_mesh_connected = true;
00467         if (esp_mesh_is_root())
00468         {
00469             tcpip_adapter_dhcpc_start(TCPIP_ADAPTER_IF_STA);
00470         }
00471         esp_mesh_comm_p2p_start();
00472         break;
00473     case MESH_EVENT_PARENT_DISCONNECTED:
00474         ESP_LOGI(MESH_TAG,
00475                  "<MESH_EVENT_PARENT_DISCONNECTED>reason:%d",
00476                  event.info.disconnected.reason);
00477         is_mesh_connected = false;
00478         // mesh_disconnected_indicator();
00479         mesh_layer = esp_mesh_get_layer();
00480         break;
00481     case MESH_EVENT_LAYER_CHANGE:
00482         mesh_layer = event.info.layer_change.new_layer;
00483         ESP_LOGI(MESH_TAG, "<MESH_EVENT_LAYER_CHANGE>layer:%d-->%d%s",
00484                  last_layer, mesh_layer,
00485                  esp_mesh_is_root() ? "<ROOT>" : (mesh_layer == 2) ? "<layer2>" : "");
00486         last_layer = mesh_layer;
00487         // mesh_connected_indicator(mesh_layer);
00488         break;
00489     case MESH_EVENT_ROOT_ADDRESS:
00490         ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_ADDRESS>root address:" MACSTR "",
00491                  MAC2STR(event.info.root_addr.addr));
00492         mesh_root_addr = event.info.root_addr;
00493
00494         break;
00495     case MESH_EVENT_ROOT_GOT_IP:
00496         /* root starts to connect to server */
00497         ESP_LOGI(MESH_TAG,
00498                  "<MESH_EVENT_ROOT_GOT_IP>sta ip: " IPSTR ", mask: " IPSTR ", gw: " IPSTR,
00499                  IP2STR(&event.info.got_ip.ip_info.ip),
00500                  IP2STR(&event.info.got_ip.ip_info.netmask),
00501                  IP2STR(&event.info.got_ip.ip_info.gw));
00502         gotIP = true;
00503         break;
00504     case MESH_EVENT_ROOT_LOST_IP:
00505         ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_LOST_IP>");
00506         break;
00507     case MESH_EVENT_VOTE_STARTED:
00508         ESP_LOGI(MESH_TAG,
00509                  "<MESH_EVENT_VOTE_STARTED>attempts:%d, reason:%d, rc_addr:" MACSTR "",
00510                  event.info.vote_started.attempts,
00511                  event.info.vote_started.reason,
00512                  MAC2STR(event.info.vote_started.rc_addr.addr));
00513         break;
00514     case MESH_EVENT_VOTE_STOPPED:
00515         ESP_LOGI(MESH_TAG, "<MESH_EVENT_VOTE_STOPPED>");
00516         break;
00517     case MESH_EVENT_ROOT_SWITCH_REQ:
00518         ESP_LOGI(MESH_TAG,
00519                  "<MESH_EVENT_ROOT_SWITCH_REQ>reason:%d, rc_addr:" MACSTR "",
00520                  event.info.switch_req.reason,
00521                  MAC2STR(event.info.switch_req.rc_addr.addr));
00522         break;
00523     case MESH_EVENT_ROOT_SWITCH_ACK:
00524         /* new root */
00525         mesh_layer = esp_mesh_get_layer();
00526         esp_mesh_get_parent_bssid(&mesh_parent_addr);
00527         ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_SWITCH_ACK>layer:%d, parent:" MACSTR "", mesh_layer,
    MAC2STR(mesh_parent_addr.addr));
00528         break;
00529     case MESH_EVENT_TODS_STATE:
00530         ESP_LOGI(MESH_TAG, "<MESH_EVENT_TODS_REACHABLE>state:%d",
00531                  event.info.toDS_state);
00532         break;
00533     case MESH_EVENT_ROOT_FIXED:
00534         ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_FIXED>%s",
00535                  event.info.root_fixed.is_fixed ? "fixed" : "not fixed");
00536         break;
00537     case MESH_EVENT_ROOT_ASKED_YIELD:
00538         ESP_LOGI(MESH_TAG,
00539                  "<MESH_EVENT_ROOT_ASKED_YIELD>" MACSTR ", rssi:%d, capacity:%d",
00540                  MAC2STR(event.info.root_conflict.addr),
00541                  event.info.root_conflict.rssi,
00542                  event.info.root_conflict.capacity);
00543         break;
00544     case MESH_EVENT_CHANNEL_SWITCH:
00545         ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHANNEL_SWITCH>new channel:%d",
    event.info.channel_switch.channel);
00546         break;
00547     case MESH_EVENT_SCAN_DONE:
00548         ESP_LOGI(MESH_TAG, "<MESH_EVENT_SCAN_DONE>number:%d",
00549                  event.info.scan_done.number);
```

```
00550         break;
00551     case MESH_EVENT_NETWORK_STATE:
00552         ESP_LOGI(MESH_TAG, "<MESH_EVENT_NETWORK_STATE>is_rootless:%d",
00553                  event.info.network_state.is_rootless);
00554         break;
00555     case MESH_EVENT_STOP_RECONNECTION:
00556         ESP_LOGI(MESH_TAG, "<MESH_EVENT_STOP_RECONNECTION>");
00557         break;
00558     case MESH_EVENT_FIND_NETWORK:
00559         ESP_LOGI(MESH_TAG, "<MESH_EVENT_FIND_NETWORK>new channel:%d, router BSSID:" MACSTR "",
00560                  event.info.find_network.channel, MAC2STR(event.info.find_network.router_bssid));
00561         break;
00562     case MESH_EVENT_ROUTER_SWITCH:
00563         ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROUTER_SWITCH>new router:%s, channel:%d, " MACSTR "",
00564                  event.info.router_switch.ssid, event.info.router_switch.channel,
      MAC2STR(event.info.router_switch.bssid));
00565         break;
00566     default:
00567         ESP_LOGI(MESH_TAG, "unknown id:%d", event.id);
00568         break;
00569     }
00570 }
00571
00575 void app_main(void)
00576 {
00577
00578     /*******************************************************************************/
00579     /* ---------- ----------       Global Initialization   ---------- ---------- */
00580     /*******************************************************************************/
00581     loc.id = (uint16_t *)malloc(sizeof(uint16_t) * NODE_LIMIT);   // Allocate # of slots for distance
      id's
00582     loc.dist = (uint32_t *)malloc(sizeof(uint32_t) * NODE_LIMIT); // Allocate # of slots for distances
00583     loc.qf = (uint8_t *)malloc(sizeof(uint8_t) * NODE_LIMIT);     // Allocate # of slots for quality
      factor of distances
00584
00585     /*******************************************************************************/
00586     /* ---------- ----------       SPI Initialization      ---------- ---------- */
00587     /*******************************************************************************/
00588     /* --- SPI Initializaion Start --- */
00589     spi_init();
00590     // -- Add slight delay to let all systems comfortably boot up
00591     vTaskDelay(1000 / portTICK_PERIOD_MS);
00592
00593     /*******************************************************************************/
00594     /* ---------- ----------       I2C Initialization      ---------- ---------- */
00595     /*******************************************************************************/
00596
00597     i2c_master_init();
00598     ssd1306_init();
00599
00600     // Display information on OLED
00601     disp_info();
00602
00603     /*******************************************************************************/
00604     /* ---------- ----------    Node Configuration Check   ---------- ---------- */
00605     /*******************************************************************************/
00606     dwm_resp_cfg_get dwm1cfg, dwm2cfg;
00607     dwm_resp_err err1, err2;
00608     bool rst = false;
00609     // -- Check if designated Anchor DWM is configured as desired
00610     dwm2cfg = dwm_cfg_get(NODE_TYPE_ANC);
00611     if ((dwm2cfg.cfg.cfg_bytes[0] != DWM2_CFG_BYTE0) && (dwm2cfg.cfg.cfg_bytes[1] != DWM2_CFG_BYTE1))
00612     {
00613         uint8_t anccfg_arg = 0x9A, intcfg_arg = 0x02;
00614         err2 = dwm_anc_set(anccfg_arg, NODE_TYPE_ANC);
00615         err2 = dwm_rst(NODE_TYPE_ANC);
00616         rst = true;
00617     }
00618     // -- Check if designated Anchor DWM is configured as desired
00619     dwm1cfg = dwm_cfg_get(NODE_TYPE_TAG);
00620     if ((dwm1cfg.cfg.cfg_bytes[0] != DWM1_CFG_BYTE0) && (dwm1cfg.cfg.cfg_bytes[1] != DWM1_CFG_BYTE1))
00621     {
00622         uint8_t tagcfg_arg[] = {0x9A, 0x00}, intcfg_arg = 0x02;
00623         err1 = dwm_tag_set(tagcfg_arg, NODE_TYPE_TAG);
00624         err1 = dwm_int_cfg(intcfg_arg, NODE_TYPE_TAG); // Does not seem to work, will ignore
      interrupts for now
00625         err1 = dwm_rst(NODE_TYPE_TAG);
00626         rst = true;
00627     }
00628     // -- If a DWM module had to reconfigure, we wait a few moments for it to reboot & connect to
      other DWMs
00629     if (rst == true)
00630         vTaskDelay(5000 / portTICK_PERIOD_MS);
00631
00632     /*******************************************************************************/
00633     /* ---------- ----------    Mesh Initializaion Start   ---------- ---------- */
00634     /*******************************************************************************/
```

```
00635     // ESP_ERROR_CHECK(mesh_light_init());
00636     ESP_ERROR_CHECK(nvs_flash_init());
00637     /*  tcpip initialization */
00638     tcpip_adapter_init();
00639     /* for mesh
00640      * stop DHCP server on softAP interface by default
00641      * stop DHCP client on station interface by default
00642      * */
00643     ESP_ERROR_CHECK(tcpip_adapter_dhcps_stop(TCPIP_ADAPTER_IF_AP));
00644     ESP_ERROR_CHECK(tcpip_adapter_dhcpc_stop(TCPIP_ADAPTER_IF_STA));
00645 #if 0
00646     /* static ip settings */
00647     tcpip_adapter_ip_info_t sta_ip;
00648     sta_ip.ip.addr = ipaddr_addr("192.168.1.102");
00649     sta_ip.gw.addr = ipaddr_addr("192.168.1.1");
00650     sta_ip.netmask.addr = ipaddr_addr("255.255.255.0");
00651     tcpip_adapter_set_ip_info(WIFI_IF_STA, &sta_ip);
00652 #endif
00653     /*  wifi initialization */
00654     ESP_ERROR_CHECK(esp_event_loop_init(NULL, NULL));
00655     wifi_init_config_t config = WIFI_INIT_CONFIG_DEFAULT();
00656     ESP_ERROR_CHECK(esp_wifi_init(&config));
00657     ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_FLASH));
00658     ESP_ERROR_CHECK(esp_wifi_start());
00659     /*  mesh initialization */
00660     ESP_ERROR_CHECK(esp_mesh_init());
00661     ESP_ERROR_CHECK(esp_mesh_set_max_layer(CONFIG_MESH_MAX_LAYER));
00662     ESP_ERROR_CHECK(esp_mesh_set_vote_percentage(1));
00663     ESP_ERROR_CHECK(esp_mesh_set_ap_assoc_expire(10));
00664 #ifdef MESH_FIX_ROOT
00665     ESP_ERROR_CHECK(esp_mesh_fix_root(1));
00666 #endif
00667     mesh_cfg_t cfg = MESH_INIT_CONFIG_DEFAULT();
00668     /* mesh ID */
00669     memcpy((uint8_t *)&cfg.mesh_id, MESH_ID, 6);
00670     /* mesh event callback */
00671     cfg.event_cb = &mesh_event_handler;
00672     /* router */
00673     cfg.channel = CONFIG_MESH_CHANNEL;
00674     cfg.router.ssid_len = strlen(CONFIG_MESH_ROUTER_SSID);
00675     memcpy((uint8_t *)&cfg.router.ssid, CONFIG_MESH_ROUTER_SSID, cfg.router.ssid_len);
00676     memcpy((uint8_t *)&cfg.router.password, CONFIG_MESH_ROUTER_PASSWD,
00677            strlen(CONFIG_MESH_ROUTER_PASSWD));
00678     /* mesh softAP */
00679     ESP_ERROR_CHECK(esp_mesh_set_ap_authmode(CONFIG_MESH_AP_AUTHMODE));
00680     cfg.mesh_ap.max_connection = CONFIG_MESH_AP_CONNECTIONS;
00681     memcpy((uint8_t *)&cfg.mesh_ap.password, CONFIG_MESH_AP_PASSWD,
00682            strlen(CONFIG_MESH_AP_PASSWD));
00683     ESP_ERROR_CHECK(esp_mesh_set_config(&cfg));
00684     /* mesh start */
00685     ESP_ERROR_CHECK(esp_mesh_start());
00686     ESP_LOGI(MESH_TAG, "mesh starts successfully, heap:%d, %s\n", esp_get_free_heap_size(),
00687              esp_mesh_is_root_fixed() ? "root fixed" : "root not fixed");
00688 }
```

Index