```
//*****************************************************************************
//
// Bradley Manzo
// Thomas Ke
// EEC 172 SQ23
// Lab 6 Code
//
//*****************************************************************************
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>

// Simplelink includes
#include "simplelink.h"

//Driverlib includes
#include "hw_types.h"
#include "gpio.h"
#include "hw_apps_rcm.h"
#include "hw_common_reg.h"
#include "hw_memmap.h"
#include "hw_nvic.h"
#include "interrupt.h"
#include "prcm.h"
#include "hw_ints.h"
#include "rom.h"
#include "rom_map.h"
#include "spi.h"
#include "systick.h"
#include "utils.h"
#include "uart.h"

//Common interface includes
#include "gpio_if.h"
#include "common.h"
#include "uart_if.h"

// Pin configurations
#include "Adafruit_GFX.h"
#include "Adafruit_SSD1351.h"
#include "glcdfont.h"
```

```
#include "pin_mux_config.h"

#define MAX_URI_SIZE 128
#define URI_SIZE MAX_URI_SIZE + 1


#define APPLICATION_NAME        "SSL"
#define APPLICATION_VERSION     "1.1.1.EEC.Winter2017"
#define SERVER_NAME             "a2ttghdziztuu6-ats.iot.us-east-1.amazonaws.com"
#define GOOGLE_DST_PORT          8443

#define SL_SSL_CA_CERT "/cert/RootCA.der"
#define SL_SSL_PRIVATE "/cert/private.der"
#define SL_SSL_CLIENT  "/cert/client.der"

//NEED TO UPDATE THIS FOR IT TO WORK!
#define DATE                26    /* Current Date */
#define MONTH               5     /* Month 1-12 */
#define YEAR                2023  /* Current year */
#define HOUR                12    /* Time - hours */
#define MINUTE              0     /* Time - minutes */
#define SECOND              0     /* Time - seconds */

// JSON headers
#define POSTHEADER "POST /things/thomas_launchpad/shadow HTTP/1.1\n\r"
#define HOSTHEADER "Host: a2ttghdziztuu6-ats.iot.us-east-1.amazonaws.com\r\n"
#define CHEADER "Connection: Keep-Alive\r\n"
#define CTHEADER "Content-Type: application/json; charset=utf-8\r\n"
#define CLHEADER1 "Content-Length: "
#define CLHEADER2 "\r\n\r\n"

#define GETHEADER "GET /things/thomas_launchpad/shadow HTTP/1.1\n\r"


#define ADDRESS_START   "{\"state\": {\n\r\"desired\" : {\n\r\"address\" : \""
//char ADDRESS_START[23] = "\",\n\r\"address\" : \"";
#define LOCATION_START  "\",\n\r\"location\" : \":"
#define LOCATION_END   "\"\n\r}}}\n\r\n\r"
//*****************************************************************************
//                 GLOBAL VARIABLES -- Start
//*****************************************************************************

// some helpful macros for systick
// the cc3200's fixed clock frequency of 80 MHz
// note the use of ULL to indicate an unsigned long long constant
#define SYSCLKFREQ 80000000ULL

// macro to convert ticks to microseconds
#define TICKS_TO_US(ticks) \
    ((((ticks) / SYSCLKFREQ) * 1000000ULL) + \
    ((((ticks) % SYSCLKFREQ) * 1000000ULL) / SYSCLKFREQ))\

// macro to convert microseconds to ticks
#define US_TO_TICKS(us) ((SYSCLKFREQ / 1000000ULL) * (us))

// systick reload value set to 40ms period
// (PERIOD_SEC) * (SYSCLKFREQ) = PERIOD_TICKS
#define SYSTICK_RELOAD_VAL 3200000UL

#define MASTER_MODE      1

#define SPI_IF_BIT_RATE  100000
#define GPS_BAUD_RATE   9600
#define TR_BUFF_SIZE     100

#define BLACK           0x0000
#define BLUE            0x001F
#define GREEN           0x07E0
#define CYAN            0x07FF
#define RED             0xF800
#define MAGENTA         0xF81F
#define YELLOW          0xFFE0
#define WHITE           0xFFFF

//#define CONSOLE         UARTA1_BASE
//#define CONSOLE_PERIPH  PRCM_UARTA1
//#define UartGetChar()        MAP_UARTCharGet(CONSOLE)
//#define UartPutChar(c)       MAP_UARTCharPut(CONSOLE,c)
#define MAX_STRING_LENGTH    200


// track systick counter periods elapsed
// if it is not 0, we know the transmission ended
volatile int systick_cnt = 1;

extern void (* const g_pfnVectors[])(void);
volatile unsigned char P59_intstatus;
volatile unsigned long P59_intcount;
volatile unsigned char P2_intstatus;
volatile unsigned long P2_intcount;

unsigned long start_int;
unsigned long end_int;

char TextRx[MAX_STRING_LENGTH+1];
int TextRxLength = 0;
char TextTx[MAX_STRING_LENGTH+1];
int TextTxLength = 0;
int i = 0;
int match = 0;
int first = 0;
int xTx = 0;
int yTx = 0;
unsigned long ulStatus;
long lRetVal = -1;
int incorrect = 0;

// Global variables
char* Restaurant;
char* Rest_first_half;
char* Rest_second_half;
char* Address;
char* GPGGA = "$GPGGA";
char Coordinates[MAX_STRING_LENGTH + 1];
char Latitude_str[50];
char Longitude_str[50];
char New_Latitude_str[50];
char New_Longitude_str[50];
float Latitude_float = 0;
float Longitude_float = 0;
char Latitude_sign;
char Longitude_sign;
int Coordinate_check = 0;
int rand_restaurant = 0;
char NMEA_Message[MAX_STRING_LENGTH + 1];
int NMEA_Message_iterator = 0;
int no_coord = 0;

// Dictionaries for all restaurants in Davis
// As well as defined lengths for proper random selection
#define BURGER_NUM  9
```

```
char* Burger_Restaurants[BURGER_NUM][2] =
   {{"Burgers and Brew", "255 2nd St, Davis, CA 95616"},
   {"The Melt", "500 1st St Ste 13a, Davis, CA 95616"},
   {"Tommy J's Grill & Catering", "726 2nd St, Davis, CA 95616"},
   {"The Hotdogger", "129 E St, Davis, CA 95616"},
   {"Jack in the Box", "337 G St, Davis, CA 95616"},
   {"In-N-Out Burger", "1020 Olive Dr, Davis, CA 95616"},
   {"Carl's Jr.", "1616 E Covell Blvd, Davis, CA 95616"},
   {"Jack's Urban Eats", "1321 W Covell Blvd, Davis, CA 95616"},
   {"Burger King", "2026 Lyndell Terrace, Davis, CA 95616"}};

#define INDIAN_NUM  8
char* Indian_Restaurants[INDIAN_NUM][2] =
   {{"Yeti Restaurant", "234 E St, Davis, CA 95616"},
    {"Preethi India", "J B ANDERSON BUILDING, 715 2nd St, Davis, CA 95616"},
    {"Kathmandu Kitchen", "234 G St #4517, Davis, CA 95616"},
    {"Shah's Halal","146, Hutchison Dr, Davis, CA 95616"},
    {"The Halal Guys", "500 1st St C7, Davis, CA 95616"},
    {"Akka Kadai", "504 L St, Davis, CA 95616"},
    {"Sam's Restaurant", "301 B St, Davis, CA 95616"},
    {"Ali Baba", "220 3rd St, Davis, CA 95616"}};

#define THAI_NUM    5
char* Thai_Restaurants[THAI_NUM][2] =
   {{"Thai Canteen Davis","117 E St, Davis, CA 95616"},
    {"Sophia's Thai Bar","129 E St, Davis, CA 95616"},
    {"Red 88 Noodle Bar","223 G St, Davis, CA 95616"},
    {"Paste Thai","417 Mace Blvd i, Davis, CA 95618"},
    {"My Burma","500 1st St #11, Davis, CA 95616"}};

#define JAPANESE_NUM    13
char* Japanese_Restaurants[JAPANESE_NUM][2] =
   {{"Yuchan Shokudo","109 E St, Davis, CA 95616"},
    {"Zen Toro Bistro","132 E St #100, Davis, CA 95616"},
    {"Mikuni","500 1st St #19, Davis, CA 95616"},
    {"Hikari Sushi","110 F St Ste A, Davis, CA 95616"},
    {"Jusco","228 G St, Davis, CA 95616"},
    {"Zumapoke & Lush Ice","730 3rd St, Davis, CA 95616"},
    {"I Love Sushi", "620 W Covell Blvd suit number b, Davis, CA 95616"},
    {"Sushirrito","500 1st St Ste 13a, Davis, CA 95616"},
    {"Davis Sushi","1260 Lake Blvd Suite 103, Davis, CA 95616"},
    {"T-Kumi Ramen","Second Floor, 1260 Lake Blvd #267, Davis, CA 95616"},
    {"Good Friends","400 G St, Davis, CA 95616"},
    {"Nami Sushi","2880 5th St #105, Davis, CA 95618"},
  {"Huku Bistro","417 Mace Blvd D, Davis, CA 95618"}};

#define CHINESE_NUM 16
char* Chinese_Restaurants[CHINESE_NUM][2] =
   {{"Hunan Bar ","207 D St, Davis, CA 95616"},
   {"Tim's Kitchen","808 2nd St, Davis, CA 95616"},
   {"Tasty Kitchen","335 F St, Davis, CA 95616"},
   {"Davis Noodle City","129 E St #1d, Davis, CA 95616"},
   {"Dumpling House","129 E St, Davis, CA 95616"},
   {"Chengdu Style Restaurant","737 Russell Blvd, Davis, CA 95616"},
   {"Sesame INC","825 Russell Blvd #21, Davis, CA 95616"},
   {"Hometown Kitchen","330 G St, Davis, CA 95616"},
   {"Four Seasons Gourmet","1601 Research Park Dr, Davis, CA 95618"},
   {"Open Rice Kitchen","Chen Building, 204 G St, Davis, CA 95616"},
   {"Davis Well Season","1753 Research Park Dr, Davis, CA 95618"},
   {"Hunan Style","630 G St, Davis, CA 95616"},
   {"Uniboil","132 E St #260, Davis, CA 95616"},
   {"Honey D Cafe","213 E St, Davis, CA 95616"},
   {"Panda Express","1491 W Covell Blvd, Davis, CA 95616"},
   {"Ding How Restaurant","640 W Covell Blvd, Davis, CA 95616"}};

#define VIET_NUM    4
char* Vietnamese_Restaurants[VIET_NUM][2] =
   {{"Pho King 4","226 3rd St, Davis, CA 95616"},
   {"Pho Tasty","301 G St, Davis, CA 95616"},
   {"Sit Lo Saigon","424 G St, Davis, CA 95616"},
   {"Dah Bao","2880 5th St #140, Davis, CA 95618"}};

#define ITALIAN_NUM 18
char* Italian_Restaurants[ITALIAN_NUM][2] =
   {{"Domino's Pizza","2038 Lyndell Terrace Ste 103, Davis, CA 95616"},
   {"Pizza Guys","505 L St, Davis, CA 95616"},
   {"Pizza & Pints","236 B St, Davis, CA 95616"},
   {"Woodstock's Pizza Davis","238 G St, Davis, CA 95616"},
   {"Blaze Pizza","212 F St, Davis, CA 95616"},
   {"Steve's Pizza","314 F St, Davis, CA 95616"},
   {"Uncle Vito's","524 2nd St, Davis, CA 95616"},
   {"Paesanos","139 G St, Davis, CA 95616"},
   {"Cenario's Pizza of Davis","1300 E Covell Blvd B, Davis, CA 95616"},
   {"Papa Murphy's | Take 'N' Bake Pizza","640 W Covell Blvd Suite G, Davis, CA 95616"},
   {"Mountain Mike's","1411 W Covell Blvd Suite 111, Davis, CA 95616"},
   {"Symposium","1620 E 8th St, Davis, CA 95616"},
   {"Little Caesars Pizza","1340 E Covell Blvd, Davis, CA 95616"},
   {"Lamppost Pizza","1260 Lake Blvd #113, Davis, CA 95616"},
   {"Fire Wings Davis","640 W Covell Blvd F, Davis, CA 95616"},
   {"Domino's Pizza","4120 Chiles Rd, Davis, CA 95618"},
   {"Osteria Fasulo","2657 Portage Bay E #8, Davis, CA 95616"},
   {"Wingstop","408 G St, Davis, CA 95616"}};

#define COFFEE_NUM  19
char* Coffee_Shops[COFFEE_NUM][2] =
   {{"Peet’s Coffee","Hutchison Dr, Davis, CA 95616"},
   {"The Coffee House","1 Shields Ave, Davis, CA 95616"},
   {"Philz Coffee","521 2nd St, Davis, CA 95616"},
   {"Dutch Bros Coffee","980 Olive Dr, Davis, CA 95616"},
   {"Temple Coffee Roasters","239 G St, Davis, CA 95616"},
   {"Cloud Forest Cafe","222 D St, Davis, CA 95616"},
   {"Pachamama Coffee Davis","130 G St, Davis, CA 95616"},
   {"Teaspoon Cafe","110 F St Suite D, Davis, CA 95616"},
   {"Mishka's Café","610 2nd St, Davis, CA 95616"},
   {"Black Frogs Coffee","431 G St, Davis, CA 95616"},
   {"3rd & U Café","223 3rd St, Davis, CA 95616"},
   {"Starbucks","623 2nd St, Davis, CA 95616"},
   {"Common Grounds Coffee","2171 Cowell Blvd, Davis, CA 95618"},
   {"Black Bear Diner Davis","255 2nd St, Davis, CA 95616"},
   {"Crepeville","330 3rd St, Davis, CA 95616"},
   {"Cafe Bernardo","234 D St, Davis, CA 95616"},
   {"Delta of Venus","122 B St, Davis, CA 95616"},
   {"IHOP","1745 Cowell Blvd, Davis, CA 95618"},
   {"Three Ladies Cafe","130 G St suite a, Davis, CA 95616"}};

#define MEXICAN_NUM 11
char* Mexican_Restaurants[MEXICAN_NUM][2] =
   {{"El Patio","200 E St, Davis, CA 95616"},
   {"Taqueria El Burrito","223 F St, Davis, CA 95616"},
   {"Taqueria Davis","505 L St, Davis, CA 95616"},
   {"Tres Hermanas","805 2nd St, Davis, CA 95616"},
   {"Guads Tacos & Beer","231 3rd St, Davis, CA 95616"},
   {"Taqueria Guadalajara","640 W Covell Blvd, Davis, CA 95616"},
   {"Dos Coyotes","1411 W Covell Blvd, Davis, CA 95616"},
   {"Chipotle","227 E St Ste 1, Davis, CA 95616"},
   {"Taco Bell","425 G St, Davis, CA 95616"},
   {"Dos Coyotes","2191 Cowell Blvd, Davis, CA 95618"},
   {"Taqueria Guadalajara","417 Mace Blvd A, Davis, CA 95618"}};

#define SANDWICH_NUM    11
char* Sandwich_Shops[SANDWICH_NUM][2] =
    {{"Ike's Sandwiches","212 F St B, Davis, CA 95616"},
    {"Subway","757 Russell Blvd Space 29, Davis, CA 95616"},
    {"Mr. Pickle's - Davis, CA","2191 Cowell Blvd F, Davis, CA 95618"},
    {"TOGO'S","1411 W Covell Blvd #105, Davis, CA 95616"},
    {"Panera Bread","609 3rd St, Davis, CA 95616"},
    {"Zia's Delicatessen","616 3rd St, Davis, CA 95616"},
    {"Subway","2014 Lyndell Terrace Suite B, Davis, CA 95616"},
    {"The Posh Bagel","206 F St, Davis, CA 95616"},
    {"Jack's Urban Eats","1321 W Covell Blvd, Davis, CA 95616"},
    {"Noah's NY Bagels","1411 W Covell Blvd Suite 114 A, Davis, CA 95616"},
    {"Nick The Greek","206 E St, Davis, CA 95616"}};

uint64_t delta = 0;
uint64_t delta_us = 0;

// Int to accumulate bits onto to form message
uint32_t message;
uint32_t prev_message;

// Variables to maintain repetition logic
char prev_char;
int repetitions = 0;
char character = 0;
// Array to maintain font color
int colors[7] = {BLUE, GREEN, CYAN, RED, MAGENTA, YELLOW, WHITE};
int font_count = 0;
// Array to store characters corresponding to repeated button presses
char letters3[6][3] = {{'A', 'B', 'C'},
            {'D', 'E', 'F'},
            {'G', 'H', 'I'},
            {'J', 'K', 'L'},
            {'M', 'N', 'O'},
            {'T', 'U', 'V'}};

char letters4[2][4] = {{'P', 'Q', 'R', 'S'},
            {'W', 'X', 'Y', 'Z'}};


//*****************************************************************************
//          GLOBAL VARIABLES -- End
//*****************************************************************************

// Application specific status/error codes
typedef enum{
  // Choosing -0x7D0 to avoid overlap w/ host-driver's error codes
```

```
    LAN_CONNECTION_FAILED = -0x7D0,
    INTERNET_CONNECTION_FAILED = LAN_CONNECTION_FAILED - 1,
    DEVICE_NOT_IN_STATION_MODE = INTERNET_CONNECTION_FAILED - 1,

    STATUS_CODE_MAX = -0xBB8
}e_AppStatusCodes;

typedef struct
{
   /* time */
   unsigned long tm_sec;
   unsigned long tm_min;
   unsigned long tm_hour;
   /* date */
   unsigned long tm_day;
   unsigned long tm_mon;
   unsigned long tm_year;
   unsigned long tm_week_day; //not required
   unsigned long tm_year_day; //not required
   unsigned long reserved[3];
}SlDateTime;


//*****************************************************************************
//                 GLOBAL VARIABLES -- Start
//*****************************************************************************
volatile unsigned long  g_ulStatus = 0;//SimpleLink Status
unsigned long  g_ulPingPacketsRecv = 0; //Number of Ping Packets received
unsigned long  g_ulGatewayIP = 0; //Network Gateway IP address
unsigned char  g_ucConnectionSSID[SSID_LEN_MAX+1]; //Connection SSID
unsigned char  g_ucConnectionBSSID[BSSID_LEN_MAX]; //Connection BSSID
signed char    *g_Host = SERVER_NAME;
SlDateTime g_time;
#if defined(ccs) || defined(gcc)
extern void (* const g_pfnVectors[])(void);
#endif
#if defined(ewarm)
extern uVectorEntry __vector_table;
#endif
//*****************************************************************************
//                 GLOBAL VARIABLES -- End
//*****************************************************************************
```

```
//*****************************************************************************
//                    LOCAL FUNCTION PROTOTYPES
//*****************************************************************************
static long WlanConnect();
static int set_time();
static void BoardInit(void);
static long InitializeAppVariables();
static int tls_connect();
static int connectToAccessPoint();
static int http_post(int);

//*****************************************************************************
// SimpleLink Asynchronous Event Handlers -- Start
//*****************************************************************************


//*****************************************************************************
//
//! \brief The Function Handles WLAN Events
//!
//! \param[in]  pWlanEvent - Pointer to WLAN Event Info
//!
//! \return None
//!
//*****************************************************************************
void SimpleLinkWlanEventHandler(SlWlanEvent_t *pWlanEvent) {
   if(!pWlanEvent) {
      return;
   }

   switch(pWlanEvent->Event) {
      case SL_WLAN_CONNECT_EVENT: {
         SET_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);

         //
         // Information about the connected AP (like name, MAC etc) will be
         // available in 'slWlanConnectAsyncResponse_t'.
         // Applications can use it if required
         //
         //  slWlanConnectAsyncResponse_t *pEventData = NULL;
         // pEventData = &pWlanEvent->EventData.STAandP2PModeWlanConnected;
         //

         // Copy new connection SSID and BSSID to global parameters
```

```
    memcpy(g_ucConnectionSSID,pWlanEvent->EventData.
           STAandP2PModeWlanConnected.ssid_name,
           pWlanEvent->EventData.STAandP2PModeWlanConnected.ssid_len);
    memcpy(g_ucConnectionBSSID,
           pWlanEvent->EventData.STAandP2PModeWlanConnected.bssid,
           SL_BSSID_LENGTH);

    UART_PRINT("[WLAN EVENT] STA Connected to the AP: %s , "
               "BSSID: %x:%x:%x:%x:%x:%x\n\r",
               g_ucConnectionSSID,g_ucConnectionBSSID[0],
               g_ucConnectionBSSID[1],g_ucConnectionBSSID[2],
               g_ucConnectionBSSID[3],g_ucConnectionBSSID[4],
               g_ucConnectionBSSID[5]);
}
break;

case SL_WLAN_DISCONNECT_EVENT: {
    slWlanConnectAsyncResponse_t*  pEventData = NULL;

    CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);
    CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_AQUIRED);

    pEventData = &pWlanEvent->EventData.STAandP2PModeDisconnected;

    // If the user has initiated 'Disconnect' request,
    //'reason_code' is SL_USER_INITIATED_DISCONNECTION
    if(SL_USER_INITIATED_DISCONNECTION == pEventData->reason_code) {
        UART_PRINT("[WLAN EVENT]Device disconnected from the AP: %s,"
            "BSSID: %x:%x:%x:%x:%x:%x on application's request \n\r",
                 g_ucConnectionSSID,g_ucConnectionBSSID[0],
                 g_ucConnectionBSSID[1],g_ucConnectionBSSID[2],
                 g_ucConnectionBSSID[3],g_ucConnectionBSSID[4],
                 g_ucConnectionBSSID[5]);
    }
    else {
        UART_PRINT("[WLAN ERROR]Device disconnected from the AP AP: %s, "
                 "BSSID: %x:%x:%x:%x:%x:%x on an ERROR..!! \n\r",
                 g_ucConnectionSSID,g_ucConnectionBSSID[0],
                 g_ucConnectionBSSID[1],g_ucConnectionBSSID[2],
                 g_ucConnectionBSSID[3],g_ucConnectionBSSID[4],
                 g_ucConnectionBSSID[5]);
    }
    memset(g_ucConnectionSSID,0,sizeof(g_ucConnectionSSID));
    memset(g_ucConnectionBSSID,0,sizeof(g_ucConnectionBSSID));
```

```
        }
        break;

        default: {
            UART_PRINT("[WLAN EVENT] Unexpected event [0x%x]\n\r",
                       pWlanEvent->Event);
        }
        break;
    }
}

//*****************************************************************************
//
//! \brief This function handles network events such as IP acquisition, IP
//!           leased, IP released etc.
//!
//! \param[in]  pNetAppEvent - Pointer to NetApp Event Info
//!
//! \return None
//!
//*****************************************************************************
void SimpleLinkNetAppEventHandler(SlNetAppEvent_t *pNetAppEvent) {
    if(!pNetAppEvent) {
        return;
    }

    switch(pNetAppEvent->Event) {
        case SL_NETAPP_IPV4_IPACQUIRED_EVENT: {
            SlIpV4AcquiredAsync_t *pEventData = NULL;

            SET_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_AQUIRED);

            //Ip Acquired Event Data
            pEventData = &pNetAppEvent->EventData.ipAcquiredV4;

            //Gateway IP address
            g_ulGatewayIP = pEventData->gateway;

            UART_PRINT("[NETAPP EVENT] IP Acquired: IP=%d.%d.%d.%d , "
                       "Gateway=%d.%d.%d.%d\n\r",
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,3),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,2),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,1),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,0),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,3),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,2),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,1),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,0));
        }
        break;

        default: {
            UART_PRINT("[NETAPP EVENT] Unexpected event [0x%x] \n\r",
                    pNetAppEvent->Event);
        }
        break;
    }
}


//*****************************************************************************
//
//! \brief This function handles HTTP server events
//!
//! \param[in]  pServerEvent - Contains the relevant event information
//! \param[in]    pServerResponse - Should be filled by the user with the
//!                                      relevant response information
//!
//! \return None
//!
//****************************************************************************
void SimpleLinkHttpServerCallback(SlHttpServerEvent_t *pHttpEvent, SlHttpServerResponse_t
*pHttpResponse) {
    // Unused in this application
}

//*****************************************************************************
//
//! \brief This function handles General Events
//!
//! \param[in]     pDevEvent - Pointer to General Event Info
//!
//! \return None
//!
//*****************************************************************************
void SimpleLinkGeneralEventHandler(SlDeviceEvent_t *pDevEvent) {
    if(!pDevEvent) {
        return;
    }

    //
    // Most of the general errors are not FATAL are are to be handled
    // appropriately by the application
    //
    UART_PRINT("[GENERAL EVENT] - ID=[%d] Sender=[%d]\n\n",
               pDevEvent->EventData.deviceEvent.status,
               pDevEvent->EventData.deviceEvent.sender);
}


//*****************************************************************************
//
//! This function handles socket events indication
//!
//! \param[in]      pSock - Pointer to Socket Event Info
//!
//! \return None
//!
//*****************************************************************************
void SimpleLinkSockEventHandler(SlSockEvent_t *pSock) {
    if(!pSock) {
        return;
    }

    switch( pSock->Event ) {
        case SL_SOCKET_TX_FAILED_EVENT:
            switch( pSock->socketAsyncEvent.SockTxFailData.status) {
                case SL_ECLOSE:
                    UART_PRINT("[SOCK ERROR] - close socket (%d) operation "
                               "failed to transmit all queued packets\n\n",
                                   pSock->socketAsyncEvent.SockTxFailData.sd);
                    break;
                default:
                    UART_PRINT("[SOCK ERROR] - TX FAILED  :  socket %d , reason "
                               "(%d) \n\n",
                               pSock->socketAsyncEvent.SockTxFailData.sd,
pSock->socketAsyncEvent.SockTxFailData.status);
                  break;
            }
            break;

        default:
          UART_PRINT("[SOCK EVENT] - Unexpected Event [%x0x]\n\n",pSock->Event);
        break;
   }
}


//*****************************************************************************
// SimpleLink Asynchronous Event Handlers -- End
//*****************************************************************************


//*****************************************************************************
//
//! \brief This function initializes the application variables
//!
//! \param    0 on success else error code
//!
//! \return None
//!
//*****************************************************************************
static long InitializeAppVariables() {
   g_ulStatus = 0;
   g_ulGatewayIP = 0;
   g_Host = SERVER_NAME;
   memset(g_ucConnectionSSID,0,sizeof(g_ucConnectionSSID));
   memset(g_ucConnectionBSSID,0,sizeof(g_ucConnectionBSSID));
   return SUCCESS;
}


//*****************************************************************************
//! \brief This function puts the device in its default state. It:
//!           - Set the mode to STATION
//!           - Configures connection policy to Auto and AutoSmartConfig
//!           - Deletes all the stored profiles
//!           - Enables DHCP
//!           - Disables Scan policy
//!           - Sets Tx power to maximum
//!           - Sets power policy to normal
//!           - Unregister mDNS services
//!           - Remove all filters
//!
//! \param   none
//! \return  On success, zero is returned. On error, negative is returned
```

```
//****************************************************************************
static long ConfigureSimpleLinkToDefaultState() {
   SlVersionFull   ver = {0};
   _WlanRxFilterOperationCommandBuff_t  RxFilterIdMask = {0};

   unsigned char ucVal = 1;
   unsigned char ucConfigOpt = 0;
   unsigned char ucConfigLen = 0;
   unsigned char ucPower = 0;

   long lRetVal = -1;
   long lMode = -1;

   lMode = sl_Start(NULL, NULL, NULL);
   ASSERT_ON_ERROR(lMode);

   // If the device is not in station-mode, try configuring it in station-mode
   if (ROLE_STA != lMode) {
      if (ROLE_AP == lMode) {
         // If the device is in AP mode, we need to wait for this event
         // before doing anything
         while(!IS_IP_ACQUIRED(g_ulStatus)) {
#ifndef SL_PLATFORM_MULTI_THREADED
           _SlNonOsMainLoopTask();
#endif
         }
      }

      // Switch to STA role and restart
      lRetVal = sl_WlanSetMode(ROLE_STA);
      ASSERT_ON_ERROR(lRetVal);

      lRetVal = sl_Stop(0xFF);
      ASSERT_ON_ERROR(lRetVal);

      lRetVal = sl_Start(0, 0, 0);
      ASSERT_ON_ERROR(lRetVal);

      // Check if the device is in station again
      if (ROLE_STA != lRetVal) {
         // We don't want to proceed if the device is not coming up in STA-mode
         return DEVICE_NOT_IN_STATION_MODE;
      }
   }
```

```
    // Get the device's version-information
    ucConfigOpt = SL_DEVICE_GENERAL_VERSION;
    ucConfigLen = sizeof(ver);
    lRetVal = sl_DevGet(SL_DEVICE_GENERAL_CONFIGURATION, &ucConfigOpt,
                        &ucConfigLen, (unsigned char *)(&ver));
    ASSERT_ON_ERROR(lRetVal);

    UART_PRINT("Host Driver Version: %s\n\r",SL_DRIVER_VERSION);
    UART_PRINT("Build Version %d.%d.%d.%d.31.%d.%d.%d.%d.%d.%d.%d.%d\n\r",
    ver.NwpVersion[0],ver.NwpVersion[1],ver.NwpVersion[2],ver.NwpVersion[3],
    ver.ChipFwAndPhyVersion.FwVersion[0],ver.ChipFwAndPhyVersion.FwVersion[1],
    ver.ChipFwAndPhyVersion.FwVersion[2],ver.ChipFwAndPhyVersion.FwVersion[3],
    ver.ChipFwAndPhyVersion.PhyVersion[0],ver.ChipFwAndPhyVersion.PhyVersion[1],
    ver.ChipFwAndPhyVersion.PhyVersion[2],ver.ChipFwAndPhyVersion.PhyVersion[3]);

    // Set connection policy to Auto + SmartConfig
    //      (Device's default connection policy)
    lRetVal = sl_WlanPolicySet(SL_POLICY_CONNECTION,
                        SL_CONNECTION_POLICY(1, 0, 0, 0, 1), NULL, 0);
    ASSERT_ON_ERROR(lRetVal);

    // Remove all profiles
    lRetVal = sl_WlanProfileDel(0xFF);
    ASSERT_ON_ERROR(lRetVal);



    //
    // Device in station-mode. Disconnect previous connection if any
    // The function returns 0 if 'Disconnected done', negative number if already
    // disconnected Wait for 'disconnection' event if 0 is returned, Ignore
    // other return-codes
    //
    lRetVal = sl_WlanDisconnect();
    if(0 == lRetVal) {
      // Wait
      while(IS_CONNECTED(g_ulStatus)) {
#ifndef SL_PLATFORM_MULTI_THREADED
          _SlNonOsMainLoopTask();
#endif
      }
    }
    // Enable DHCP client
    lRetVal = sl_NetCfgSet(SL_IPV4_STA_P2P_CL_DHCP_ENABLE,1,1,&ucVal);
    ASSERT_ON_ERROR(lRetVal);

    // Disable scan
    ucConfigOpt = SL_SCAN_POLICY(0);
    lRetVal = sl_WlanPolicySet(SL_POLICY_SCAN , ucConfigOpt, NULL, 0);
    ASSERT_ON_ERROR(lRetVal);

    // Set Tx power level for station mode
    // Number between 0-15, as dB offset from max power - 0 will set max power
    ucPower = 0;
    lRetVal = sl_WlanSet(SL_WLAN_CFG_GENERAL_PARAM_ID,
            WLAN_GENERAL_PARAM_OPT_STA_TX_POWER, 1, (unsigned char *)&ucPower);
    ASSERT_ON_ERROR(lRetVal);

    // Set PM policy to normal
    lRetVal = sl_WlanPolicySet(SL_POLICY_PM , SL_NORMAL_POLICY, NULL, 0);
    ASSERT_ON_ERROR(lRetVal);

    // Unregister mDNS services
    lRetVal = sl_NetAppMDNSUnRegisterService(0, 0);
    ASSERT_ON_ERROR(lRetVal);

    // Remove  all 64 filters (8*8)
    memset(RxFilterIdMask.FilterIdMask, 0xFF, 8);
    lRetVal = sl_WlanRxFilterSet(SL_REMOVE_RX_FILTER, (_u8 *)&RxFilterIdMask,
                    sizeof(_WlanRxFilterOperationCommandBuff_t));
    ASSERT_ON_ERROR(lRetVal);

    lRetVal = sl_Stop(SL_STOP_TIMEOUT);
    ASSERT_ON_ERROR(lRetVal);

    InitializeAppVariables();

    return lRetVal; // Success
}


//*****************************************************************************
//
//! Board Initialization & Configuration
//!
//! \param  None
```

```
//!
//! \return None
//
//*****************************************************************************
static void BoardInit(void) {
/* In case of TI-RTOS vector table is initialize by OS itself */
#ifndef USE_TIRTOS
  //
  // Set vector table base
  //
#if defined(ccs)
    MAP_IntVTableBaseSet((unsigned long)&g_pfnVectors[0]);
#endif
#if defined(ewarm)
    MAP_IntVTableBaseSet((unsigned long)&__vector_table);
#endif
#endif
    //
    // Enable Processor
    //
    MAP_IntMasterEnable();
    MAP_IntEnable(FAULT_SYSTICK);

    PRCMCC3200MCUInit();
}

/**
 * Reset SysTick Counter
 */
static inline void SysTickReset(void) {
    // any write to the ST_CURRENT register clears it
    // after clearing it automatically gets reset without
    // triggering exception logic
    // see reference manual section 3.2.1
    HWREG(NVIC_ST_CURRENT) = 1;

    // clear the global count variable
    systick_cnt = 1;
}

/**
 * SysTick Interrupt Handler
 *
 * Keep track of whether the systick counter wrapped
 */
static void SysTickHandler(void) {
    // increment every time the systick handler fires
    systick_cnt++;
}
//*****************************************************************************
//
//! \brief Connecting to a WLAN Accesspoint
//!
//!  This function connects to the required AP (SSID_NAME) with Security
//!  parameters specified in te form of macros at the top of this file
//!
//! \param  None
//!
//! \return  0 on success else error code
//!
//! \warning    If the WLAN connection fails or we don't aquire an IP
//!           address, It will be stuck in this function forever.
//
//*****************************************************************************
// Register Interrupt Handler
// P59 handler wired to IR receiver
static void GPIOA0IntHandler(void)
{
    unsigned long ulStatus;
    // Records interrupt status of IR receiver from GPIO
    ulStatus = MAP_GPIOIntStatus(GPIOA0_BASE, true);
    // Clears interrupt status from GPIO
    MAP_GPIOIntClear(GPIOA0_BASE, ulStatus);
    // Records the current time and calculates duration since last
    delta = systick_cnt*SYSTICK_RELOAD_VAL - SysTickValueGet();
    // Resets SysTick count and repetitions
    SysTickReset();
    // Converts clock cycles to milliseconds
    delta_us = TICKS_TO_US(delta);// clear interrupts on GPIOA0
    // Sets IR Int received flag high
    P59_intstatus = 1;
    P59_intcount++;
}

static void UARTA1IntHandler(void)
{
    unsigned long ulStatus;
    // Records interrupt status of UART
```

```
ulStatus = MAP_UARTIntStatus(UARTA1_BASE, true);
// Clears interrupt status from UART
UARTIntClear(UARTA1_BASE, ulStatus);
// As long as there are chars to receive, build string
while(UARTCharsAvail(UARTA1_BASE))
{
   TextRx[TextRxLength] = UARTCharGetNonBlocking(UARTA1_BASE);
   TextRxLength++;
}
// Iteratively builds potential match of $GPGGA and records in buffer when found
if(TextRx[TextRxLength] == '$')
{
   NMEA_Message_iterator = 0;
   NMEA_Message[NMEA_Message_iterator] = TextRx[TextRxLength];
   match++;
}
else if(TextRx[TextRxLength] == 'G' && match == 1)
{
   NMEA_Message_iterator++;
   NMEA_Message[NMEA_Message_iterator] = TextRx[TextRxLength];
   match++;
}
else if(TextRx[TextRxLength] == 'P' && match == 2)
{
   NMEA_Message_iterator++;
   NMEA_Message[NMEA_Message_iterator] = TextRx[TextRxLength];
   match++;
}
else if(TextRx[TextRxLength] == 'G' && match == 3)
{
   NMEA_Message_iterator++;
   NMEA_Message[NMEA_Message_iterator] = TextRx[TextRxLength];
   match++;
}
else if(TextRx[TextRxLength] == 'G' && match == 4)
{
   NMEA_Message_iterator++;
   NMEA_Message[NMEA_Message_iterator] = TextRx[TextRxLength];
   match++;
}
else if(TextRx[TextRxLength] == 'A' && match == 5)
{
   NMEA_Message_iterator++;
   NMEA_Message[NMEA_Message_iterator] = TextRx[TextRxLength];
```

```
        match++;
    }
    else if(match == 6)
    {
        NMEA_Message_iterator++;
        NMEA_Message[NMEA_Message_iterator] = TextRx[TextRxLength];
        if (TextRx[TextRxLength] == '\n')
        {
            // Confirms if valid $GPGGA sentence
            if(NMEA_Message_iterator > 60)
                Coordinate_check = 1;
            else
                match = 0;
        }
    }
    else
        match = 0;
    // Sets UART Int received flag high
    if (TextRxLength > MAX_STRING_LENGTH)
        TextRxLength = 0;
    P2_intstatus = 1;
    P2_intcount++;
}
//static void UARTA1IntHandler(void)
//{
//    unsigned long ulStatus;
//
//    // Records interrupt status of UART
//    ulStatus = MAP_UARTIntStatus(CONSOLE, true);
//    // Clears interrupt status from UART
//    MAP_UARTIntClear(CONSOLE, ulStatus);
//    // As long as there are chars to receive, build string
//    while(UARTCharsAvail(CONSOLE))
//    {
//        TextRx[TextRxLength] = UARTCharGetNonBlocking(CONSOLE);
//        TextRxLength++;
//    }
//    // Sets UART Int received flag high
//    P2_intstatus = 1;
//    P2_intcount++;
//}

static void SysTickInit(void) {
    // configure the reset value for the systick countdown register
    MAP_SysTickPeriodSet(SYSTICK_RELOAD_VAL);

    // register interrupts on the systick module
    MAP_SysTickIntRegister(SysTickHandler);

    // enable interrupts on systick
    // (trigger SysTickHandler when countdown reaches 0)
    MAP_SysTickIntEnable();

    // enable the systick module itself
    MAP_SysTickEnable();
}

static long WlanConnect() {
    SlSecParams_t secParams = {0};
    long lRetVal = 0;

    secParams.Key = SECURITY_KEY;
    secParams.KeyLen = strlen(SECURITY_KEY);
    secParams.Type = SECURITY_TYPE;

    UART_PRINT("Attempting connection to access point: ");
    UART_PRINT(SSID_NAME);
    UART_PRINT("... ...");
    lRetVal = sl_WlanConnect(SSID_NAME, strlen(SSID_NAME), 0, &secParams, 0);
    ASSERT_ON_ERROR(lRetVal);

    UART_PRINT(" Connected!!!\n\r");


    // Wait for WLAN Event
    while((!IS_CONNECTED(g_ulStatus)) || (!IS_IP_ACQUIRED(g_ulStatus))) {
        // Toggle LEDs to Indicate Connection Progress
        _SlNonOsMainLoopTask();
        GPIO_IF_LedOff(MCU_IP_ALLOC_IND);
        MAP_UtilsDelay(800000);
        _SlNonOsMainLoopTask();
        GPIO_IF_LedOn(MCU_IP_ALLOC_IND);
        MAP_UtilsDelay(800000);
    }

    return SUCCESS;
```

```
}

//*****************************************************************************
//
//! This function updates the date and time of CC3200.
//!
//! \param None
//!
//! \return
//!     0 for success, negative otherwise
//!
//*****************************************************************************

static int set_time() {
    long retVal;

    g_time.tm_day = DATE;
    g_time.tm_mon = MONTH;
    g_time.tm_year = YEAR;
    g_time.tm_sec = HOUR;
    g_time.tm_hour = MINUTE;
    g_time.tm_min = SECOND;

    retVal = sl_DevSet(SL_DEVICE_GENERAL_CONFIGURATION,
                       SL_DEVICE_GENERAL_CONFIGURATION_DATE_TIME,
                       sizeof(SlDateTime),(unsigned char *)(&g_time));

    ASSERT_ON_ERROR(retVal);
    return SUCCESS;
}

long printErrConvenience(char * msg, long retVal) {
    UART_PRINT(msg);
    GPIO_IF_LedOn(MCU_RED_LED_GPIO);
    return retVal;
}
//*****************************************************************************
//
//! This function demonstrates how certificate can be used with SSL.
//! The procedure includes the following steps:
//! 1) connect to an open AP
//! 2) get the server name via a DNS request
//! 3) define all socket options and point to the CA certificate
//! 4) connect to the server via TCP
```

```
//!
//! \param None
//!
//! \return  0 on success else error code
//! \return  LED1 is turned solid in case of success
//!    LED2 is turned solid in case of failure
//!
//*****************************************************************************
static int tls_connect() {
    SlSockAddrIn_t    Addr;
    int    iAddrSize;
    unsigned char    ucMethod = SL_SO_SEC_METHOD_TLSV1_2;
    unsigned int uiIP,uiCipher =
SL_SEC_MASK_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA;
    long lRetVal = -1;
    int iSockID;

    lRetVal = sl_NetAppDnsGetHostByName(g_Host, strlen((const char *)g_Host),
                                    (unsigned long*)&uiIP, SL_AF_INET);

    if(lRetVal < 0) {
        return printErrConvenience("Device couldn't retrieve the host name \n\r", lRetVal);
    }

    Addr.sin_family = SL_AF_INET;
    Addr.sin_port = sl_Htons(GOOGLE_DST_PORT);
    Addr.sin_addr.s_addr = sl_Htonl(uiIP);
    iAddrSize = sizeof(SlSockAddrIn_t);
    //
    // opens a secure socket
    //
    iSockID = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, SL_SEC_SOCKET);
    if( iSockID < 0 ) {
        return printErrConvenience("Device unable to create secure socket \n\r", lRetVal);
    }

    //
    // configure the socket as TLS1.2
    //
    lRetVal = sl_SetSockOpt(iSockID, SL_SOL_SOCKET, SL_SO_SECMETHOD, &ucMethod,\
                              sizeof(ucMethod));
    if(lRetVal < 0) {
        return printErrConvenience("Device couldn't set socket options \n\r", lRetVal);
    }
```

```
//
//configure the socket as ECDHE RSA WITH AES256 CBC SHA
//
lRetVal = sl_SetSockOpt(iSockID, SL_SOL_SOCKET, SL_SO_SECURE_MASK, &uiCipher,\
                    sizeof(uiCipher));
if(lRetVal < 0) {
    return printErrConvenience("Device couldn't set socket options \n\r", lRetVal);
}

//
//configure the socket with CA certificate - for server verification
//
lRetVal = sl_SetSockOpt(iSockID, SL_SOL_SOCKET, \
                    SL_SO_SECURE_FILES_CA_FILE_NAME, \
                    SL_SSL_CA_CERT, \
                    strlen(SL_SSL_CA_CERT));

if(lRetVal < 0) {
    return printErrConvenience("Device couldn't set socket options \n\r", lRetVal);
}

//configure the socket with Client Certificate - for server verification
//
lRetVal = sl_SetSockOpt(iSockID, SL_SOL_SOCKET, \
            SL_SO_SECURE_FILES_CERTIFICATE_FILE_NAME, \
                            SL_SSL_CLIENT, \
                    strlen(SL_SSL_CLIENT));

if(lRetVal < 0) {
    return printErrConvenience("Device couldn't set socket options \n\r", lRetVal);
}

//configure the socket with Private Key - for server verification
//
lRetVal = sl_SetSockOpt(iSockID, SL_SOL_SOCKET, \
        SL_SO_SECURE_FILES_PRIVATE_KEY_FILE_NAME, \
        SL_SSL_PRIVATE, \
                    strlen(SL_SSL_PRIVATE));

if(lRetVal < 0) {
    return printErrConvenience("Device couldn't set socket options \n\r", lRetVal);
}
```

```
    /* connect to the peer device - Google server */
    lRetVal = sl_Connect(iSockID, ( SlSockAddr_t *)&Addr, iAddrSize);

    if(lRetVal < 0) {
        UART_PRINT("Device couldn't connect to server:");
        UART_PRINT(SERVER_NAME);
        UART_PRINT("\n\r");
        return printErrConvenience("Device couldn't connect to server \n\r", lRetVal);
    }
    else {
        UART_PRINT("Device has connected to the website:");
        UART_PRINT(SERVER_NAME);
        UART_PRINT("\n\r");
    }

    GPIO_IF_LedOff(MCU_RED_LED_GPIO);
    GPIO_IF_LedOn(MCU_GREEN_LED_GPIO);
    return iSockID;
}






int connectToAccessPoint() {
    long lRetVal = -1;
    GPIO_IF_LedConfigure(LED1|LED3);

    GPIO_IF_LedOff(MCU_RED_LED_GPIO);
    GPIO_IF_LedOff(MCU_GREEN_LED_GPIO);

    lRetVal = InitializeAppVariables();
    ASSERT_ON_ERROR(lRetVal);

    //
    // Following function configure the device to default state by cleaning
    // the persistent settings stored in NVMEM (viz. connection profiles &
    // policies, power policy etc)
    //
    // Applications may choose to skip this step if the developer is sure
    // that the device is in its default state at start of applicaton
    //
    // Note that all profiles and persistent settings that were done on the
    // device will be lost
    //
    lRetVal = ConfigureSimpleLinkToDefaultState();
    if(lRetVal < 0) {
      if (DEVICE_NOT_IN_STATION_MODE == lRetVal)
          UART_PRINT("Failed to configure the device in its default state \n\r");

      return lRetVal;
    }

    UART_PRINT("Device is configured in default state \n\r");

    CLR_STATUS_BIT_ALL(g_ulStatus);

    ///
    // Assumption is that the device is configured in station mode already
    // and it is in its default state
    //
    lRetVal = sl_Start(0, 0, 0);
    if (lRetVal < 0 || ROLE_STA != lRetVal) {
        UART_PRINT("Failed to start the device \n\r");
        return lRetVal;
    }

    UART_PRINT("Device started as STATION \n\r");

    //
    //Connecting to WLAN AP
    //
    lRetVal = WlanConnect();
    if(lRetVal < 0) {
        UART_PRINT("Failed to establish connection w/ an AP \n\r");
        GPIO_IF_LedOn(MCU_RED_LED_GPIO);
        return lRetVal;
    }

    UART_PRINT("Connection established w/ AP and IP is aquired \n\r");
    return 0;
}
static int http_post(int iTLSSockID){
    char acSendBuff[512];
    char acRecvbuff[1460];
    char cCLLength[200];
    char* pcBufHeaders;
```

```
int lRetVal = 0;
char newJSON[200] = ADDRESS_START;
int dataLength = 0;

pcBufHeaders = acSendBuff;
strcpy(pcBufHeaders, POSTHEADER);
pcBufHeaders += strlen(POSTHEADER);
strcpy(pcBufHeaders, HOSTHEADER);
pcBufHeaders += strlen(HOSTHEADER);
strcpy(pcBufHeaders, CHEADER);
pcBufHeaders += strlen(CHEADER);
strcpy(pcBufHeaders, "\r\n\r\n");

// Encapsulate the address and coordinates within JSON packet
strcat(newJSON, Address);
strcat(newJSON, LOCATION_START);
strcat(newJSON, New_Latitude_str);
strcat(newJSON, ", ");
strcat(newJSON, New_Longitude_str);
strcat(newJSON, LOCATION_END);
dataLength = strlen(newJSON);
strcpy(pcBufHeaders, CTHEADER);
pcBufHeaders += strlen(CTHEADER);
strcpy(pcBufHeaders, CLHEADER1);

pcBufHeaders += strlen(CLHEADER1);
sprintf(cCLLength, "%d", dataLength);

strcpy(pcBufHeaders, cCLLength);
pcBufHeaders += strlen(cCLLength);
strcpy(pcBufHeaders, CLHEADER2);
pcBufHeaders += strlen(CLHEADER2);

// post JSON header + text
strcpy(pcBufHeaders, newJSON);
pcBufHeaders += strlen(newJSON);
int testDataLength = strlen(pcBufHeaders);
//
// Send the packet to the server */
//
lRetVal = sl_Send(iTLSSockID, acSendBuff, strlen(acSendBuff), 0);
if(lRetVal < 0) {
    UART_PRINT("POST failed. Error Number: %i\n\r",lRetVal);
    sl_Close(iTLSSockID);
```

```
        GPIO_IF_LedOn(MCU_RED_LED_GPIO);
        return lRetVal;
    }
    lRetVal = sl_Recv(iTLSSockID, &acRecvbuff[0], sizeof(acRecvbuff), 0);
    if(lRetVal < 0) {
        UART_PRINT("Received failed. Error Number: %i\n\r",lRetVal);
        //sl_Close(iSSLSockID);
        GPIO_IF_LedOn(MCU_RED_LED_GPIO);
        return lRetVal;
    }
    else {
        acRecvbuff[lRetVal+1] = '\0';
        UART_PRINT(acRecvbuff);
        UART_PRINT("\n\r\n\r");
    }

    //sl_Stop(SL_STOP_TIMEOUT);
    return 0;
}
static int http_get(int iTLSSockID){
    char acSendBuff[512];
    char acRecvbuff[1460];
    char* pcBufHeaders;
    int lRetVal = 0;

    pcBufHeaders = acSendBuff;
    strcpy(pcBufHeaders, GETHEADER);
    pcBufHeaders += strlen(GETHEADER);
    strcpy(pcBufHeaders, HOSTHEADER);
    pcBufHeaders += strlen(HOSTHEADER);
    strcpy(pcBufHeaders, CHEADER);
    pcBufHeaders += strlen(CHEADER);
    strcpy(pcBufHeaders, "\r\n\r\n");

    int testDataLength = strlen(pcBufHeaders);

    UART_PRINT(acSendBuff);


    //
    // Send the packet to the server */
    //
    lRetVal = sl_Send(iTLSSockID, acSendBuff, strlen(acSendBuff), 0);
    if(lRetVal < 0) {
        UART_PRINT("POST failed. Error Number: %i\n\r",lRetVal);
        sl_Close(iTLSSockID);
        GPIO_IF_LedOn(MCU_RED_LED_GPIO);
        return lRetVal;
    }
    lRetVal = sl_Recv(iTLSSockID, &acRecvbuff[0], sizeof(acRecvbuff), 0);
    if(lRetVal < 0) {
        UART_PRINT("Received failed. Error Number: %i\n\r",lRetVal);
        //sl_Close(iSSLSockID);
        GPIO_IF_LedOn(MCU_RED_LED_GPIO);
           return lRetVal;
    }
    else {
        acRecvbuff[lRetVal+1] = '\0';
        UART_PRINT(acRecvbuff);
        UART_PRINT("\n\r\n\r");
    }

    return 0;
}
//*****************************************************************************
//
//! Main
//!
//! \param  none
//!
//! \return None
//!
//*****************************************************************************
void main() {

    // Seed RNG
    srand(time(NULL));
    //
    // Initialize board configuration
    //
    BoardInit();

    PinMuxConfig();

    //
    // Enable the SPI module clock
    //
    MAP_PRCMPeripheralClkEnable(PRCM_GSPI,PRCM_RUN_MODE_CLK);
```

```
//
// Reset the peripheral
//
MAP_PRCMPeripheralReset(PRCM_GSPI);

//
// Reset SPI
//
MAP_SPIReset(GSPI_BASE);

//
// Configure SPI interface to OLED
//
MAP_SPIConfigSetExpClk(GSPI_BASE,MAP_PRCMPeripheralClockGet(PRCM_GSPI),
                SPI_IF_BIT_RATE,SPI_MODE_MASTER,SPI_SUB_MODE_0,
                (SPI_SW_CTRL_CS |
                SPI_4PIN_MODE |
                SPI_TURBO_OFF |
                SPI_CS_ACTIVELOW |
                SPI_WL_8));

//
// Enable SPI for communication to OLED
//
MAP_SPIEnable(GSPI_BASE);

Adafruit_Init();

// Enable SysTick
SysTickInit();

// Update connection status on OLED
fillRect(0,0,128,8,BLACK);
setTextSize(1);
setTextColor(WHITE, BLACK);
Outstr("not connected");

// Register Interrupt Handler
// (Port, pointer to handler function)
MAP_GPIOIntRegister(GPIOA0_BASE, GPIOA0IntHandler);

// Configure Falling Edge
// (Port, bit-packed pin select, interrupt trigger mechanism)
```

```
    MAP_GPIOIntTypeSet(GPIOA0_BASE, 0x10, GPIO_FALLING_EDGE);

    // Interrupt Status
    // (Port, True: masked interupt status, false: raw interrupt status)
    // Returns the current interupt status enumerated as a bit field
    // of the values described in GPIOIntEnable()
    ulStatus = MAP_GPIOIntStatus(GPIOA0_BASE, false);

    // Clear Interrupt
    // (Port, with field returned from status above)
    MAP_GPIOIntClear(GPIOA0_BASE, ulStatus);

    // clear global variables
    P59_intstatus = 0;
    P59_intcount = 0;

    // Enable Interrupt
    // (Port, Flags)
    MAP_GPIOIntEnable(GPIOA0_BASE, 0x10);

    SysTickReset();

    // Initialize UART Terminal
    InitTerm();

    // Clear UART Terminal
    ClearTerm();

    // Configure UART to A1BASE for NEO-6M GPS Module

MAP_UARTConfigSetExpClk(UARTA1_BASE,MAP_PRCMPeripheralClockGet(PRCM_UARTA
1),
                    GPS_BAUD_RATE, (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE
|
                     UART_CONFIG_PAR_NONE));

   // Configure UART FIFO queue
   UARTFIFODisable(UARTA1_BASE);

   MAP_UARTIntRegister(UARTA1_BASE, UARTA1IntHandler);

   UARTFIFOLevelSet(UARTA1_BASE, UART_FIFO_TX1_8, UART_FIFO_RX1_8);

   // Initialize UART interrupts
```

```
ulStatus = MAP_UARTIntStatus(UARTA1_BASE, false);

UARTIntClear(UARTA1_BASE, ulStatus);

UARTIntEnable(UARTA1_BASE,UART_INT_RX);

// Print GUI for restaurant options
 xTx = 0;
 yTx = 0;
 fillScreen(BLACK);
 setTextSize(1);
 setTextColor(GREEN, BLACK);
 setCursor(xTx, yTx);
 Outstr("Connected ");
 xTx = 0;
 yTx += 8;
 setCursor(xTx, yTx);
 setTextColor(WHITE, BLACK);
 Outstr("I:Indian   B:Burger");
 xTx = 0;
 yTx += 8;
 setCursor(xTx, yTx);
 Outstr("T:Thai     P:Pizza");
 xTx = 0;
 yTx += 8;
 setCursor(xTx, yTx);
 Outstr("CH:Chinese S:Sandwich");
 xTx = 0;
 yTx += 8;
 setCursor(xTx, yTx);
 Outstr("M:Mexican  C:Cafe");
 xTx = 0;
 yTx += 8;
 setCursor(xTx, yTx);
 Outstr("J:Japanese");
 // Position in pixels
 // Text to Transmit Position
 xTx = 0;
 yTx = 64;
 setCursor(xTx, yTx);
 setTextSize(1);
 setTextColor(WHITE, BLACK);

 while (1) {
    while (P59_intstatus == 0 && P2_intstatus == 0) {;}
    // If GPIO Interrupt (IR) Receivied
    if(P59_intstatus)
    {
      setCursor(xTx, yTx);
      // clear flag
      P59_intstatus=0;
      // If longer than standard repeat, stop remembering past input
      if(delta_us > 300000)
      {
        repetitions = 0;
        prev_message = 0;
        prev_char = 0;
      }
      // If larger than "1" and not garbage data, decode the message
      if((delta_us > 2500) && (delta_us < 300000) && (message > 2))
      {
        // If message message is new, and previous char wasn't a debug character, increment
the x position
        if(message != prev_message && prev_char != '!' && prev_char != '1' && prev_char !=
'2' && prev_char != '3' && character != '2')
        {
          // Append character to Transmitting Text
          TextTx[TextTxLength] = character;
          TextTxLength += 1;
          // If at edge of screen, go down to beginning of new line (\n\r)
          if(xTx >= 120)
          {
            xTx = 0;
            if(yTx < 120)
              yTx += 8;
            else
              yTx = 64;
          }
          // Otherwise increment by width of character
          else
          {
            xTx += 6;
          }
        }
        // If last remembered word is the same: increment repetitions
        // otherwise, message is done repeating and should print
```

```
/*========================================================================*/
            //      Infrared Decoding

/*========================================================================*/
            // 0 Button (Space)
            if(message == 0b00000010111111010000000011111111)
            {
               character = ' ';

            }
            // 1 Button pressed (Font Color Change)
            else if(message == 0b00000010111111011000000001111111)
            {
               if(font_count < 6)
                  font_count++;
               else
                  font_count = 0;
               character = '1';
            }
            // 2 button pressed
            else if(message == 0b00000010111111010100000010111111)
            {
               if(prev_message == message)
                  repetitions++;
               else
                  repetitions = 0;
               //letters = {'A', 'B', 'C'};
               if (repetitions > (sizeof(letters3[0]) - 1))
                  repetitions = repetitions - (sizeof(letters3[0]));
               character = letters3[0][repetitions];
            }
            // 3 button pressed
            else if(message == 0b00000010111111011100000000111111)
            {
               if(prev_message == message)
                  repetitions++;
               else
                  repetitions = 0;
               //letters = {'D', 'E', 'F'};
               if (repetitions > (sizeof(letters3[0]) - 1))
                  repetitions = repetitions - (sizeof(letters3[0]));
               character = letters3[1][repetitions];
            }
```

```
// 4 button pressed
else if(message == 0b00000010111111010010000011011111)
{
   if(prev_message == message)
      repetitions++;
   else
      repetitions = 0;
   //letters = {'G', 'H', 'I'};
   if (repetitions > (sizeof(letters3[0]) - 1))
      repetitions = repetitions - (sizeof(letters3[0]));
   character = letters3[2][repetitions];
}
// 5 button pressed
else if(message == 0b00000010111111011010000001011111)
{
   if(prev_message == message)
      repetitions++;
   else
      repetitions = 0;
   //letters = {'J', 'K', 'L'};
   if (repetitions > (sizeof(letters3[0]) - 1))
      repetitions = repetitions - (sizeof(letters3[0]));
   character = letters3[3][repetitions];
}
// 6 button pressed
else if(message == 0b00000010111111010110000010011111)
{
   if(prev_message == message)
      repetitions++;
   else
      repetitions = 0;
   //letters = {'M', 'N', 'O'};
   if (repetitions > (sizeof(letters3[0]) - 1))
      repetitions = repetitions - (sizeof(letters3[0]));
   character = letters3[4][repetitions];
}
// 7 button pressed
else if(message == 0b00000010111111011110000000011111)
{
   if(prev_message == message)
      repetitions++;
   else
      repetitions = 0;
   //letters = {'P', 'Q', 'R', 'S'};
```

```
    if (repetitions > (sizeof(letters4[0]) - 1))
        repetitions = repetitions - (sizeof(letters4[0]));
    character = letters4[0][repetitions];
}
// 8 button pressed
else if(message == 0b00000010111111010001000011101111)
{
    if(prev_message == message)
        repetitions++;
    else
        repetitions = 0;
    //letters = {'T', 'U', 'V'};
    if (repetitions > (sizeof(letters3[0]) - 1))
        repetitions = repetitions - (sizeof(letters3[0]));
    character = letters3[5][repetitions];
}
// 9 button pressed
else if(message == 0b00000010111111011001000001101111)
{
    if(prev_message == message)
        repetitions++;
    else
        repetitions = 0;
    //letters = {'W', 'X', 'Y', 'Z'};
    if (repetitions > (sizeof(letters4[0]) - 1))
        repetitions = repetitions - (sizeof(letters4[0]));
    character = letters4[1][repetitions];
}
// Enter button pressed (MUTE)
else if(message == 0b00000010111111010000100011110111)
{
    character = '2';
}
// Delete button pressed (LAST)
else if(message == 0b00000010111111010000001011111101)
{
    //if(xTx >= 6)
    TextTx[TextTxLength] = '/0';
    // By removing from scope
    TextTxLength--;
    character = '3';
    fillRect(xTx,yTx,6, 8,BLACK);
    xTx -= 6;
}
```

```
            else
            {
            // Otherwise, debugging character
                  character = '!';
            }
            prev_message = message;
            prev_char = character;
            // If not a debugging or a function character, print the character to the screen
            if(character != '!' && character != '1' && character != '2' && character != '3')
            {
               yTx = 64;
               xTx = 0;
               UART_PRINT("character: %c\n\r", character);
               drawChar(xTx, yTx, character, WHITE, BLACK, 1);

            }
            // If Enter button is pressed, transmit the text
            if(character == '2')
            {
               // Iniatiate connection to AWS server if valid post
               if(TextTx[0] == 'B' || TextTx[0] == 'I' || TextTx[0] == 'T' || TextTx[0] == 'J' || (TextTx[0]
== 'C' && TextTx[1] == 'H') || TextTx[0] == 'P' || TextTx[0] == 'C' || TextTx[0] == 'M' || TextTx[0] ==
'S' || TextTx[0] == 'W')
               {

                  //Connect the CC3200 to the local access point
                  lRetVal = connectToAccessPoint();
                  //Set time so that encryption can be used
                  lRetVal = set_time();
                  if(lRetVal < 0) {
                     UART_PRINT("Unable to set time in the device");
                     LOOP_FOREVER();
                  }
                  //Connect to the website with TLS encryption
                  lRetVal = tls_connect();
                  if(lRetVal < 0) {
                     ERR_PRINT(lRetVal);
                  }
               }
               // calculate random restaurant depending on message and post to server
               if(TextTx[0] == 'B')
               {
                  rand_restaurant = rand() % (BURGER_NUM - 1);
                  Restaurant = Burger_Restaurants[rand_restaurant][0];
```

```
    Address = Burger_Restaurants[rand_restaurant][1];
    http_post(lRetVal);
}
else if(TextTx[0] == 'I')
{
    rand_restaurant = rand() % (INDIAN_NUM - 1);
    Restaurant = Indian_Restaurants[rand_restaurant][0];
    Address = Indian_Restaurants[rand_restaurant][1];
    http_post(lRetVal);
}
else if(TextTx[0] == 'T')
{
    rand_restaurant = rand() % (THAI_NUM - 1);
    Restaurant = Thai_Restaurants[rand_restaurant][0];
    Address = Thai_Restaurants[rand_restaurant][1];
    http_post(lRetVal);
}
else if(TextTx[0] == 'J')
{
    rand_restaurant = rand() % (JAPANESE_NUM - 1);
    Restaurant = Japanese_Restaurants[rand_restaurant][0];
    Address = Japanese_Restaurants[rand_restaurant][1];
    http_post(lRetVal);
}
else if(TextTx[0] == 'C' && TextTx[1] == 'H')
{
    rand_restaurant = rand() % (CHINESE_NUM - 1);
    Restaurant = Chinese_Restaurants[rand_restaurant][0];
    Address = Chinese_Restaurants[rand_restaurant][1];
    http_post(lRetVal);
}
else if(TextTx[0] == 'P')
{
    rand_restaurant = rand() % (ITALIAN_NUM - 1);
    Restaurant = Italian_Restaurants[rand_restaurant][0];
    Address = Italian_Restaurants[rand_restaurant][1];
    http_post(lRetVal);
}
else if(TextTx[0] == 'C')
{
    rand_restaurant = rand() % (COFFEE_NUM - 1);
    Restaurant = Coffee_Shops[rand_restaurant][0];
    Address = Coffee_Shops[rand_restaurant][1];
    http_post(lRetVal);
```

```
                }
                else if(TextTx[0] == 'M')
                {
                   rand_restaurant = rand() % (MEXICAN_NUM - 1);
                   Restaurant = Mexican_Restaurants[rand_restaurant][0];
                   Address = Mexican_Restaurants[rand_restaurant][1];
                   http_post(lRetVal);
                }
                else if(TextTx[0] == 'S')
                {
                   rand_restaurant = rand() % (SANDWICH_NUM - 1);
                   Restaurant = Sandwich_Shops[rand_restaurant][0];
                   Address = Sandwich_Shops[rand_restaurant][1];
                   http_post(lRetVal);
                }
                else if(TextTx[0] == 'W')
                {
                   http_get(lRetVal);
                }
                else
                {
                   if(!first)
                      first = 1;
                   else
                   {
                      incorrect = 1;
                      xTx = 0;
                      yTx = 72;
                      setTextSize(1);
                      setTextColor(RED, BLACK);
                      setCursor(xTx, yTx);
                      Outstr("Incorrect Input");
                   }
                }
                // Ends connection to AWS for a valid post
                if(TextTx[0] == 'W' || TextTx[0] == 'S' || TextTx[0] == 'M' || TextTx[0] == 'C' || TextTx[0]
== 'P' || (TextTx[0] == 'C' && TextTx[1] == 'H') || TextTx[0] == 'J' || TextTx[0] == 'B' || TextTx[0] ==
'I' || TextTx[0] == 'T')
                {
                   sl_Stop(SL_STOP_TIMEOUT);
                }
                rand_restaurant = 0;
                if(incorrect == 0 && first == 1)
                {
                    fillRect(0,8,128,128,BLACK);
                    xTx = 0;
                    yTx = 8;
                    setTextSize(1);
                    setTextColor(WHITE, BLACK);
                    setCursor(xTx, yTx);
                    Outstr("Getting directions to:");
                    yTx = 16;
                    xTx = 0;
                    setCursor(xTx, yTx);
                    Outstr(Restaurant);

                    yTx += 8;
                    xTx = 0;
                    setCursor(xTx, yTx);
                    setTextColor(WHITE, BLACK);
                    Outstr("I:Indian   B:Burger");
                    xTx = 0;
                    yTx += 8;
                    setCursor(xTx, yTx);
                    Outstr("T:Thai     P:Pizza");
                    xTx = 0;
                    yTx += 8;
                    setCursor(xTx, yTx);
                    Outstr("CH:Chinese S:Sandwich");
                    xTx = 0;
                    yTx += 8;
                    setCursor(xTx, yTx);
                    Outstr("M:Mexican  C:Cafe");
                    xTx = 0;
                    yTx += 8;
                    setCursor(xTx, yTx);
                    Outstr("J:Japanese");
//                   }
                }
                incorrect = 0;
                //http_get(lRetVal);
                //sl_Stop(SL_STOP_TIMEOUT);

                TextTxLength = 0;
                memset(TextTx, 0, sizeof TextTx);
                Restaurant = NULL;
                Address = NULL;
```

```
            setCursor(0, 64);
            xTx = 0;
            yTx = 64;
        }
        // Resets repetitions
        message = 0;
    }
    // If time is between 1300 and 2500 ms, accumulate a "1"
    else if(delta_us < 2500 && delta_us > 1300)
    {
        message = message << 1;
        message = message + 1;
    }
    // If time is less than 1300 ms, accumulate a "0"
    else //if(delta_us > 0 && delta_us < 1300)
    {
        message = message << 1;
    }
    start_int = 0;
}
// Enters when valid NMEA $GPGGA sentence is recorded
if(P2_intstatus && !no_coord)
{
    P2_intstatus=0;
    UARTIntDisable(UARTA1_BASE, UART_INT_RX);
    if(Coordinate_check)
    {
        UARTDisable(UARTA1_BASE);
        UART_PRINT("%s\n",NMEA_Message);
        strcpy(Coordinates, NMEA_Message);
        int comma_count = 0;
        int Latitude_iterator = 0;
        int Longitude_iterator = 0;
        for(i = 0; i < (int)strlen(Coordinates); i++)
        {
            if(Coordinates[i] == ',')
                comma_count++;
            if(comma_count == 2 && Coordinates[i] != ',')
            {
                Latitude_str[Latitude_iterator] = Coordinates[i];
                Latitude_iterator++;
            }
            if(comma_count == 3 && Coordinates[i] != ',')
            {
```

```
        Latitude_sign = Coordinates[i];
    }
    if(comma_count == 4 && Coordinates[i] != ',')
    {
        Longitude_str[Longitude_iterator] = Coordinates[i];
        Longitude_iterator++;
    }
    if(comma_count == 5 && Coordinates[i] != ',')
    {
        Longitude_sign = Coordinates[i];
    }
}
// Convert Coordinates from strings to floats
Latitude_float = atof(Latitude_str);
Longitude_float = atof(Longitude_str);

// Record Degrees
int Latitude_int = Latitude_float/100;
int Longitude_int = Longitude_float/100;

// Calculate minutes
float Latitude_min = (Latitude_float - Latitude_int*100)/60;
float Longitude_min = (Longitude_float - Longitude_int*100)/60;

// Compose final coordinate format
Latitude_float = Latitude_int + Latitude_min;
Longitude_float = Longitude_int + Longitude_min;

// Assign proper sign depending on hemisphere
if(Latitude_sign == 'S')
    Latitude_float = -Latitude_float;
if(Longitude_sign == 'W')
    Longitude_float = -Longitude_float;

// convert back to string to be encapsulated in JSON
snprintf(New_Latitude_str, 50, "%f", Latitude_float);
snprintf(New_Longitude_str, 50, "%f", Longitude_float);

xTx = 60;
yTx = 0;
setCursor(xTx, yTx);
setTextSize(1);
setTextColor(GREEN, BLACK);
Outstr("located!");
```

```
                no_coord = 1;
                UART_PRINT("Latitude: %s, Longitude: %s\n", New_Latitude_str,
New_Longitude_str);
            }
        }
    }
}
//*****************************************************************************
//
// Close the Doxygen group.
//! @}
//
//*****************************************************************************
```