EE445L – Lab 4: Internet of Things

Michael Park and Jack Zhao

02/26/16

1.0 OBJECTIVE

The objective of this lab is to learn how to use a wifi module, learn about synchrnous and asynchronous communication, and understand bascis of Internet communication. Primarily, tasks involved requesting and acquiring data from a web server through an access point (hotspot from phone), acquiring ADC data and posting it onto a web server, and measuring number of lost packets and time required to complete the data transfer/receive while communicating with the server.

2.0 SOFTWARE DESIGN

-Attached at the End

3.0 MEASUREMENT DATA

From Openweather.org, time (in cycles):
6304
13125190
5071574
9235847
30495834
20395833
7283958
9304829
29394050
18759384
7385928

Max: 30495834 * 12.5ns = 0.381197925 s
Min: 6304 * 12.5ns = 78.8 us
Avg: 151389220/10 = 15138922 * 12.5ns = 0.189236525 s

To 445L Server, time (in cycles):
7239642
14284630
2214068
3950239
59023949
20594859
5056986
4030589
5638237
29475843

Max: 59023949 * 12.5ns = 0.737799363 s
Min: 2214068 * 12.5ns = 27.67585 ms
Avg: 151509042/10 = 15150904 *12.5ns = 0.1893863 s

There are zero packet loss from the layer we see because, under the hood, TCP detects packet loss and performs retransmissions to ensure reliable messaging.

4.0 ANALYSIS AND DISCUSSION

1) In the client server paradigm, explain the sequence of internet communications sent from client to server and from server to client as the client saves data on the server. Assume the client already is connected to the wifi AP and the client knows the IP address of the server.

Client first initiates a request to a web server. The client's credentials may be stored in a database, and the web server accesses the database server as a client. An application server interprets the returned data by applyging some rules, and provides the ouutput to the web server. Lastly, the web server returns the result to the client. In each step of this sequence of client server message exchanges, a computer processes a request and returns data.

2)What is the purpose of the DNS?

Domain name server (DNS) manages a database that maps server domain names to IP addresses. Internet uses IP address to route client requests to get to servers. So whenever a user types in a domain name, the computer accesses DNS to find the corresponding IP address and that routes to the server. In short analogy, IP addresses are like phone numbers and DNS is a phone book.

3)What is the difference between UDP and TCP communication? More specifically when should we use UDP and when should we use TCP?

Transmission Control Protocol (TCP) ensures a reliable and ordered delivery of a stream of bytes for communication. It manges message acknowledgement and reransmission (by hand-shaking) in case of lost packets. So, it is connection oriented and once a connection is established, data can be sent bidirectional. User Datagram Protocol (UDP) on the other hand, is not dedicated to end to end connections and communication does not check readiness of receiver. It does not include acknowledgement, time out, and retransmission. So, it is simpler, connectionless, and multiple messages are sent as packets in chunks. Due to these characteristics, TCP is used to control segment size, rate of data exchange, flow control and network congestion. TCP is preferred where error correction facilities are required at network interface level. Email and file transfers are common applications. UDP is largely used by time sensitive applications as well as by servers that answer small queries from huge number of clients. Examples of UDP are DNS, Voice over IP, TFTP.

Code Written for this Lab

```
/*
 * main.c - Example project for UT.6.02x Embedded Systems - Shape the World
 * Jonathan Valvano and Ramesh Yerraballi
 * July 14, 2015
 * Hardware requirements
    TM4C123 LaunchPad, optional Nokia5110
    CC3100 wifi booster and
    an internet access point with OPEN, WPA, or WEP security
```

```
// ******** main.c**************
// Modified by Michael Park, Jack Zhao
// Date Created: 02/16/2016
// Includes codes to connect to servers using AP. Collects ADC data.
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 02/23/2016



/*
 * Application Name     -   Get weather
 * Application Overview -   This is a sample application demonstrating how to
                         connect to openweathermap.org server and request for
            weather details of a city.
 * Application Details  -
http://processors.wiki.ti.com/index.php/CC31xx_SLS_Get_Weather_Application
 *                       doc\examples\sls_get_weather.pdf
 */
 /* CC3100 booster pack connections (unused pins can be used by user application)
Pin  Signal        Direction      Pin   Signal      Direction
P1.1  3.3 VCC         IN          P2.1  Gnd   GND       IN
P1.2  PB5 UNUSED      NA          P2.2  PB2   IRQ      OUT
P1.3  PB0 UART1_TX    OUT         P2.3  PE0   SSI2_CS  IN
P1.4  PB1 UART1_RX    IN          P2.4  PF0   UNUSED   NA
P1.5  PE4 nHIB        IN          P2.5  Reset nRESET   IN
P1.6  PE5 UNUSED      NA          P2.6  PB7  SSI2_MOSI IN
P1.7  PB4 SSI2_CLK    IN          P2.7  PB6  SSI2_MISO OUT
P1.8  PA5 UNUSED      NA          P2.8  PA4   UNUSED   NA
P1.9  PA6 UNUSED      NA          P2.9  PA3   UNUSED   NA
P1.10 PA7 UNUSED      NA          P2.10 PA2   UNUSED   NA

Pin  Signal        Direction      Pin   Signal       Direction
P3.1  +5  +5 V       IN           P4.1  PF2 UNUSED      OUT
P3.2  Gnd GND        IN           P4.2  PF3 UNUSED      OUT
P3.3  PD0 UNUSED     NA           P4.3  PB3 UNUSED      NA
P3.4  PD1 UNUSED     NA           P4.4  PC4 UART1_CTS   IN
P3.5  PD2 UNUSED     NA           P4.5  PC5 UART1_RTS   OUT
```

```
P3.6  PD3 UNUSED     NA          P4.6  PC6 UNUSED      NA
P3.7  PE1 UNUSED     NA          P4.7  PC7 NWP_LOG_TX  OUT
P3.8  PE2 UNUSED     NA          P4.8  PD6 WLAN_LOG_TX OUT
P3.9  PE3 UNUSED     NA          P4.9  PD7 UNUSED      IN (see R74)
P3.10 PF1 UNUSED     NA          P4.10 PF4 UNUSED      OUT(see R75)

UART0 (PA1, PA0) sends data to the PC via the USB debug cable, 115200 baud rate
Port A, SSI0 (PA2, PA3, PA5, PA6, PA7) sends data to Nokia5110 LCD

*/
#include "..\cc3100\simplelink\include\simplelink.h"
#include "../Shared/hw_memmap.h"
#include "../Shared/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "utils/cmdline.h"
#include "application_commands.h"
#include "LED.h"
#include "Nokia5110.h"
#include <string.h>
#include "ST7735.h"
#include "ADCSWTrigger.h"
#include "Timer0A.h"
#include "../Shared/tm4c123gh6pm.h"
#include <stdio.h>
#include "SysTick.h"
#define SSID_NAME  "Mike"        /* Access point name to connect to. */
#define SEC_TYPE   SL_SEC_TYPE_WPA
#define PASSKEY    "mpmp1234"        /* Password in case of secure AP */
#define BAUD_RATE   115200
void UART_Init(void){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
  GPIOPinConfigure(GPIO_PA0_U0RX);
  GPIOPinConfigure(GPIO_PA1_U0TX);
  GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
  UARTStdioConfig(0,BAUD_RATE,50000000);
}


#define MAX_RECV_BUFF_SIZE  1024
#define MAX_SEND_BUFF_SIZE  512
#define MAX_HOSTNAME_SIZE   40
#define MAX_SERVERNAME_SIZE 40
#define MAX_PASSKEY_SIZE    32
#define MAX_SSID_SIZE       32



#define SUCCESS             0

#define CONNECTION_STATUS_BIT   0
#define IP_AQUIRED_STATUS_BIT   1

/* Application specific status/error codes */
typedef enum{
   DEVICE_NOT_IN_STATION_MODE = -0x7D0,/* Choosing this number to avoid overlap w/
host-driver's error codes */

   STATUS_CODE_MAX = -0xBB8
}e_AppStatusCodes;



/* Status bits - These are used to set/reset the corresponding bits in 'g_Status' */
typedef enum{
   STATUS_BIT_CONNECTION =  0, /* If this bit is:
                         *      1 in 'g_Status', the device is connected to the AP
                         *      0 in 'g_Status', the device is not connected to the AP
                         */
```

```
    STATUS_BIT_IP_AQUIRED,       /* If this bit is:
                            *      1 in 'g_Status', the device has acquired an IP
                            *      0 in 'g_Status', the device has not acquired an IP
                            */

}e_StatusBits;



#define SET_STATUS_BIT(status_variable, bit)    status_variable |= (1<<(bit))
#define CLR_STATUS_BIT(status_variable, bit)    status_variable &= ~(1<<(bit))
#define GET_STATUS_BIT(status_variable, bit)    (0 != (status_variable & (1<<(bit))))
#define IS_CONNECTED(status_variable)           GET_STATUS_BIT(status_variable, \
                                                STATUS_BIT_CONNECTION)
#define IS_IP_AQUIRED(status_variable)          GET_STATUS_BIT(status_variable, \
                                                STATUS_BIT_IP_AQUIRED)

typedef struct{
    UINT8 SSID[MAX_SSID_SIZE];
    INT32 encryption;
    UINT8 password[MAX_PASSKEY_SIZE];
}UserInfo;

/*
 * GLOBAL VARIABLES -- Start
 */

char Recvbuff[MAX_RECV_BUFF_SIZE];
char SendBuff[MAX_SEND_BUFF_SIZE];
char HostName[MAX_HOSTNAME_SIZE];
char ServerName[MAX_SERVERNAME_SIZE];
unsigned long DestinationIP;
int SockID;
volatile uint32_t ADCvalue;

//globals for partf
volatile uint32_t counter;
```

```
uint32_t timeg[10];
uint32_t timep[10];
//index 0 for getting data
//index 1 for posting data
uint32_t current_time[2];
uint32_t maxVal[2];
uint32_t minVal[2];
uint32_t avgVal[2];
uint32_t Lost_Packet[2];

void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode
void obtainADC(void);
void tracktime(void);

typedef enum{
   CONNECTED = 0x01,
   IP_AQUIRED = 0x02,
   IP_LEASED = 0x04,
   PING_DONE = 0x08

}e_Status;
UINT32  g_Status = 0;
/*
 * GLOBAL VARIABLES -- End
 */



 /*
 * STATIC FUNCTION DEFINITIONS  -- Start
 */

static int32_t configureSimpleLinkToDefaultState(char *);
```

```
/*
 * STATIC FUNCTION DEFINITIONS -- End
 */



void Crash(uint32_t time){
  while(1){
    for(int i=time;i;i--){};
    LED_RedToggle();
  }
}
/*
 * Application's entry point
 */
// 1) change Austin Texas to your city
// 2) you can change metric to imperial if you want temperature in F
//#define REQUEST "GET /data/2.5/weather?q=Austin%20Texas&units=metric HTTP/1.1\r\nUser-
Agent: Keil\r\nHost:api.openweathermap.org\r\nAccept: */*\r\n\r\n"
#define REQUEST "GET /data/2.5/weather?q=Austin
%20Texas&APPID=358461513dd1b88b40a929ed100a6eea
HTTP/1.1\r\nHost:api.openweathermap.org\r\n\r\n"


#define PAYLOAD "GET /query?city=Austin%20Texas&id=Michael%20Park%20and%20Jack
%20Zhao&greet="
#define PAYLOAD_END "&edxcode=8086 HTTP/1.1\r\nUser-Agent: Keil\r\nHost:
embsysmooc.appspot.com\r\n\r\n"
//#define PAYLOAD "GET /query?city=Austin%20Texas&id=Mike%20Park&greet=Int%20Temp
%3D21C&edxcode=8086 HTTP/1.1\r\nUser-Agent: Keil\r\nHost:
embsysmooc.appspot.com\r\n\r\n"


int main(void)
{
//define variables
int32_t retVal;
SlSecParams_t secParams;
char temp[8];
char ch[13];
  char *pConfig = NULL; INT32 ASize = 0; SlSockAddrIn_t  Addr;
char poststring[512];
char adcfixed[12];

//initilize and connect serial
//DisableInterrupts();
counter=0;
  initClk();        // PLL 50 MHz
  UART_Init();      // Send data to PC, 115200 bps
  LED_Init();       // initialize LaunchPad I/O
SysTick_Init();
ST7735_InitR(INITR_REDTAB);
ADC0_InitSWTriggerSeq3_Ch9();         // allow time to finish activating           ****
ADC0_SAC_R &= 0xFFFFFFF8;                                                   //64x hardware
oversample
  UARTprintf("Weather App\n");
  retVal = configureSimpleLinkToDefaultState(pConfig); // set policies
  if(retVal < 0)Crash(4000000);
  retVal = sl_Start(0, pConfig, 0);
  if((retVal < 0) || (ROLE_STA != retVal) ) Crash(8000000);
  secParams.Key = PASSKEY;
  secParams.KeyLen = strlen(PASSKEY);
  secParams.Type = SEC_TYPE; // OPEN, WPA, or WEP
  sl_WlanConnect(SSID_NAME, strlen(SSID_NAME), 0, &secParams, 0);
  while((0 == (g_Status&CONNECTED)) || (0 == (g_Status&IP_AQUIRED))){
    _SlNonOsMainLoopTask();
  }
  UARTprintf("Connected\n");



//Main while loop
while(1)
   {
    strcpy(HostName,"openweathermap.org");



     //timer start//
     current_time[0]=NVIC_ST_CURRENT_R;
     //timer start//

     retVal = sl_NetAppDnsGetHostByName(HostName,strlen(HostName),&DestinationIP,
SL_AF_INET);

     //???????????
     if(retVal != 0 )
         Lost_Packet[0]++;
    //???????????

     if(retVal == 0)
     {
      Addr.sin_family = SL_AF_INET;
      Addr.sin_port = sl_Htons(80);
      Addr.sin_addr.s_addr = sl_Htonl(DestinationIP);// IP to big endian
      ASize = sizeof(SlSockAddrIn_t);
      SockID = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, 0);

         if( SockID >= 0 )
         {
       retVal = sl_Connect(SockID, ( SlSockAddr_t *)&Addr, ASize);
      }

         if((SockID >= 0)&&(retVal >= 0))
         {
       strcpy(SendBuff,REQUEST);
       sl_Send(SockID, SendBuff, strlen(SendBuff), 0);// Send the HTTP GET
       sl_Recv(SockID, Recvbuff, MAX_RECV_BUFF_SIZE, 0);// Receive response
       sl_Close(SockID);
       LED_GreenOn();
       UARTprintf("\r\n\r\n");
       UARTprintf(Recvbuff);  UARTprintf("\r\n");
                 //timer end//
                 current_time[0] = current_time[0]-NVIC_ST_CURRENT_R;
                 //timer end//

                 //parse and print json string
                 for(int i =0; i<MAX_RECV_BUFF_SIZE; i++)
                 {
                         if(Recvbuff[i] =='t'){
                                 if(Recvbuff[i+1] =='e'){
                                         if(Recvbuff[i+2] =='m'){
                                                 if(Recvbuff[i+3] =='p'){
                                                         i+=6;
                                                         for(int j=0;j<6;j++){
                                                                 temp[j]=Recvbuff[j+i];
                                                         }
                                                         break;
                                                 }
                                         }
                                 }
                         }
                 }
                 // Print Temp to LCD screen
                 ST7735_SetCursor(0,0);
                 ST7735_OutString("Temp = ");
                 ST7735_SetCursor(0,20);
                 ST7735_OutString(temp);
                 ST7735_OutString("C");
     }
    }


         //ADC voltage meter
         obtainADC();
         sprintf(adcfixed,"Voltage=%d", ADCvalue);
/*      char a[2];
         a[0] = adcfixed[11];
       a[1] = adcfixed[10];
       for(int i=0; i<13; i++)
         {
                 if(i==9) {adcfixed[i] = '.';}
                 else if (i>9) {break;}
                 else {adcfixed[i] = adcfixed[i];}
         }
         adcfixed[10] = a[0];
         adcfixed[11] = a[1];
*/
         ST7735_SetCursor(0,10);
         ST7735_OutString(adcfixed);



     for(int x=0;x<512;x++){
         poststring[x]=0;
     }


       //copy over string
         strcpy(poststring,PAYLOAD);
         strcat(poststring,adcfixed);
         strcat(poststring, PAYLOAD_END);

         //SEND TCP PAYLOAD
       strcpy(ServerName,"embsysmooc.appspot.com");


         //timer start//
         current_time[1] = NVIC_ST_CURRENT_R;
         //timer start//
         retVal = sl_NetAppDnsGetHostByName(ServerName,strlen(ServerName),&DestinationIP,
SL_AF_INET);
         if(retVal == 0)
         {
                 Addr.sin_family = SL_AF_INET;
                 Addr.sin_port = sl_Htons(80);
                 Addr.sin_addr.s_addr = sl_Htonl(DestinationIP);// IP to big endian
                 ASize = sizeof(SlSockAddrIn_t);
                 SockID = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, 0);
                 if( SockID >= 0 ){retVal = sl_Connect(SockID, ( SlSockAddr_t *)&Addr, ASize);}
                 if((SockID >= 0)&&(retVal >= 0))
                 {
                         strcpy(SendBuff,poststring);
                         //strcat(SendBuff,ch);
                         //strcat(SendBuff, PAYLOAD_END);
```

```
                    sl_Send(SockID, SendBuff, strlen(SendBuff), 0);// Send the HTTP GET
                    sl_Recv(SockID, Recvbuff, MAX_RECV_BUFF_SIZE, 0);// Receive
response
                    sl_Close(SockID);

                    //timer end//
                    current_time[1] = current_time[1] - NVIC_ST_CURRENT_R;
                    //timer end//

                    LED_GreenOn();
                    UARTprintf("\r\n\r\n");
                    UARTprintf(Recvbuff);  UARTprintf("\r\n");
               }
          }      //store info on webserver

   if(counter<10)
        tracktime();

   while(Board_Input()==0){}; // wait for touch
   //for(int i=0; i<10000; i++);
   LED_GreenOff();
 }
}

void tracktime(void)
{
   //timer method
   if(counter<10)
   {
        timeg[counter]=current_time[0];
        timep[counter]=current_time[1];
        counter++;
   }
   if(counter==10)
   {
        maxVal[0]=0;
        minVal[0]=0;
        avgVal[0]=0;
        maxVal[1]=0;
        minVal[1]=0;
        avgVal[1]=0;
        for(int i =0;i<counter;i++)
        {
            if(timeg[i] > maxVal[0]){maxVal[0]=timeg[i];}
            if(timeg[i] < minVal[0]){minVal[0]=timeg[i];}
            avgVal[0]+=timeg[i];
        }
        for(int i =0;i<counter;i++)
        {
            if(timep[i] > maxVal[1]){maxVal[1]=timep[i];}
            if(timep[i] < minVal[1]){minVal[1]=timep[i];}
            avgVal[1]+=timep[i];
        }
        avgVal[0]/=10;
        avgVal[1]/=10;
    }
}

/*!
  \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

  \param[in]     none

  \return        On success, zero is returned. On error, negative is returned
*/
static int32_t configureSimpleLinkToDefaultState(char *pConfig){
 SlVersionFull   ver = {0};
 UINT8           val = 1;
 UINT8           configOpt = 0;
 UINT8           configLen = 0;
 UINT8           power = 0;

 INT32          retVal = -1;
 INT32          mode = -1;

 mode = sl_Start(0, pConfig, 0);



   /* If the device is not in station-mode, try putting it in station-mode */
 if (ROLE_STA != mode){
   if (ROLE_AP == mode){
         /* If the device is in AP mode, we need to wait for this event before doing anything */
     while(!IS_IP_AQUIRED(g_Status));
   }

       /* Switch to STA role and restart */
   retVal = sl_WlanSetMode(ROLE_STA);

   retVal = sl_Stop(0xFF);

   retVal = sl_Start(0, pConfig, 0);

       /* Check if the device is in station again */
   if (ROLE_STA != retVal){
         /* We don't want to proceed if the device is not coming up in station-mode */
     return DEVICE_NOT_IN_STATION_MODE;
   }
 }
   /* Get the device's version-information */
 configOpt = SL_DEVICE_GENERAL_VERSION;
 configLen = sizeof(ver);
 retVal = sl_DevGet(SL_DEVICE_GENERAL_CONFIGURATION, &configOpt, &configLen,
(unsigned char *)(&ver));
```

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

  /* Remove all profiles */
 retVal = sl_WlanProfileDel(0xFF);


  /*
   * 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
   */
 retVal = sl_WlanDisconnect();
 if(0 == retVal){
     /* Wait */
   while(IS_CONNECTED(g_Status));
 }

  /* Enable DHCP client*/
 retVal = sl_NetCfgSet(SL_IPV4_STA_P2P_CL_DHCP_ENABLE,1,1,&val);

  /* Disable scan */
 configOpt = SL_SCAN_POLICY(0);
 retVal = sl_WlanPolicySet(SL_POLICY_SCAN , configOpt, NULL, 0);

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

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

  /* TBD - Unregister mDNS services */
 retVal = sl_NetAppMDNSUnRegisterService(0, 0);
```

```
  retVal = sl_Stop(0xFF);


  g_Status = 0;
  memset(&Recvbuff,0,MAX_RECV_BUFF_SIZE);
  memset(&SendBuff,0,MAX_SEND_BUFF_SIZE);
  memset(&HostName,0,MAX_HOSTNAME_SIZE);
  DestinationIP = 0;;
  SockID = 0;


  return retVal; /* Success */
}




/*
 * * ASYNCHRONOUS EVENT HANDLERS -- Start
 */

/*!
    \brief This function handles WLAN events

    \param[in]      pWlanEvent is the event passed to the handler

    \return         None

    \note

    \warning
*/
void SimpleLinkWlanEventHandler(SlWlanEvent_t *pWlanEvent){
  switch(pWlanEvent->Event){
    case SL_WLAN_CONNECT_EVENT:
```

```
    {
      SET_STATUS_BIT(g_Status, STATUS_BIT_CONNECTION);

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

    case SL_WLAN_DISCONNECT_EVENT:
    {
      sl_protocol_wlanConnectAsyncResponse_t*  pEventData = NULL;

      CLR_STATUS_BIT(g_Status, STATUS_BIT_CONNECTION);
      CLR_STATUS_BIT(g_Status, 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){
        UARTprintf(" Device disconnected from the AP on application's request \r\n");
      }
      else{
        UARTprintf(" Device disconnected from the AP on an ERROR..!! \r\n");
      }
    }
    break;

    default:
    {
      UARTprintf(" [WLAN EVENT] Unexpected event \r\n");
    }
    break;
  }
}

/*!
    \brief This function handles events for IP address acquisition via DHCP
           indication

    \param[in]      pNetAppEvent is the event passed to the handler

    \return         None

    \note

    \warning
*/
void SimpleLinkNetAppEventHandler(SlNetAppEvent_t *pNetAppEvent){
  switch(pNetAppEvent->Event)
  {
    case SL_NETAPP_IPV4_ACQUIRED:
    {

      SET_STATUS_BIT(g_Status, STATUS_BIT_IP_AQUIRED);
        /*
             * Information about the connected AP's ip, gateway, DNS etc
             * will be available in 'SlIpV4AcquiredAsync_t' - Applications
             * can use it if required
             *
             * SlIpV4AcquiredAsync_t *pEventData = NULL;
             * pEventData = &pNetAppEvent->EventData.ipAcquiredV4;
             * <gateway_ip> = pEventData->gateway;
             *
             */

    }
    break;

    default:
    {
        UARTprintf(" [NETAPP EVENT] Unexpected event \r\n");
    }
    break;
  }
}

/*!
    \brief This function handles callback for the 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

    \note

    \warning
*/
void SimpleLinkHttpServerCallback(SlHttpServerEvent_t *pHttpEvent,
                                  SlHttpServerResponse_t *pHttpResponse){
    /*
     * This application doesn't work with HTTP server - Hence these
     * events are not handled here
     */
  UARTprintf(" [HTTP EVENT] Unexpected event \r\n");
}

/*!
    \brief This function handles general error events indication

    \param[in]      pDevEvent is the event passed to the handler
```

```
   \return        None
*/
void SimpleLinkGeneralEventHandler(SlDeviceEvent_t *pDevEvent){
   /*
    * Most of the general errors are not FATAL are are to be handled
    * appropriately by the application
    */
  UARTprintf(" [GENERAL EVENT] \r\n");
}

/*!
   \brief This function handles socket events indication

   \param[in]     pSock is the event passed to the handler

   \return        None
*/
void SimpleLinkSockEventHandler(SlSockEvent_t *pSock){
  switch( pSock->Event )
  {
   case SL_NETAPP_SOCKET_TX_FAILED:
   {
         /*
         * TX Failed
         *
         * Information about the socket descriptor and status will be
         * available in 'SlSockEventData_t' - Applications can use it if
         * required
         *
         * SlSockEventData_t *pEventData = NULL;
         * pEventData = & pSock->EventData;
         */
     switch( pSock->EventData.status )
     {
      case SL_ECLOSE:
        UARTprintf(" [SOCK EVENT] Close socket operation failed to transmit all queued
packets\r\n");
```

```
          break;



        default:
          UARTprintf(" [SOCK EVENT] Unexpected event \r\n");
          break;
      }
    }
    break;

    default:
      UARTprintf(" [SOCK EVENT] Unexpected event \r\n");
    break;
  }
}
void obtainADC(void){
ADCvalue=ADC0_InSeq3();
}

/*
 * * ASYNCHRONOUS EVENT HANDLERS -- End
 */
```