## Lab 4: The Internet of Things

Objectives

* Implement a system that connects to the internet via an IEEE 802.11 – Wifi module, CC3100
* Use DNS to convert name to IP address
* Configure a smart object that can retrieve data from a weather server using TCP
* Design a smart object that can store data onto an internet server using TCP

Measurement Data

Analysis and Discussion

1. The client sends a request to the server with a random port number requesting a socket. The server creates the socket and sends a reply to the client. The client then sends the data to the server socket and the data is saved in the proper location.
2. The DNS server allows users to request an IP address of a server knowing only the URL for the website they are trying to visit.
3. The main difference between UDP and TCP is that TCP is connection based and the packet is guaranteed to get to its destination in order while UDP is sent with any guarantee it will arrive. UDP is often used to sacrifice reliability for speed over using TCP.

Software:

/\*

\* 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

\* derived from TI's getweather example

\* Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/

\*

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the

\* distribution.

\*

\* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

\* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

\* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

\* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

\* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

\* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

\* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

\* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*

\*/

/\*

\* 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 "inc/hw\_memmap.h"

#include "inc/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 "ST7735.h"

#include "ADCSWTrigger.h"

#include <stdio.h>

#include <string.h>

#define SSID\_NAME "John's iPhone" /\* Access point name to connect to. \*/

#define SEC\_TYPE SL\_SEC\_TYPE\_WPA

#define PASSKEY "837fngro8gyna" /\* 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 2048

#define MAX\_SEND\_BUFF\_SIZE 512

#define MAX\_HOSTNAME\_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];

unsigned long DestinationIP;

int SockID;

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();

}

}

int32\_t strequal(const char \*str1, const char \*str2, uint32\_t length) {

for(uint32\_t i = 0; i < length; i += 1) {

if(str1[i] != str2[i])

return 0;

}

return 1;

}

#define MAX\_TEMP\_LENGTH 10

// null terminated buffer for temperature extraction

const char temperature\_buffer[MAX\_TEMP\_LENGTH+1];

const char formatted\_temperature\_buffer[MAX\_TEMP\_LENGTH+1+9];

char\* Extract\_Temperature(char \*received\_data) {

const char \*tempMatch = "\"temp\":";

const int tempLength = 7;

int32\_t index = -1;

// find index of temperature string

for(uint32\_t i = 0; i < MAX\_RECV\_BUFF\_SIZE; i += 1) {

if(strequal(tempMatch, &received\_data[i], tempLength)) {

index = i + tempLength;

break;

}

}

//copy temperature value part out

char \*temperatureString = &received\_data[index];

uint32\_t j;

for(j = 0; j < MAX\_TEMP\_LENGTH && received\_data[j + index] != ','; j += 1) {

temperatureString[j] = received\_data[j + index];

}

temperatureString[j] = '\0';

return temperatureString;

}

void Wifi\_Connect(char \*pConfig, SlSecParams\_t \*secParams) {

int32\_t retVal;

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");

}

const char \*REQ\_1 = " HTTP/1.1\r\nUser-Agent: Keil\r\nHost:";

const char \*REQ\_2 = "\r\n\r\n";

void cleanup(void) {

memset(&Recvbuff,0,MAX\_RECV\_BUFF\_SIZE);

memset(&SendBuff,0,MAX\_SEND\_BUFF\_SIZE);

memset(&HostName,0,MAX\_HOSTNAME\_SIZE);

DestinationIP = 0;;

SockID = 0;

}

char\* HTTP\_Request(const char \*hostName, uint16\_t port, const char \*method, const char \*request, char \*requestData1, char \*requestData2) {

SlSockAddrIn\_t Addr; int32\_t retVal; uint32\_t ASize = 0;

cleanup();

strcpy(HostName, hostName);

UARTprintf("\r\n\r\nUsing host: %s\r\n", HostName);

retVal = sl\_NetAppDnsGetHostByName(HostName, strlen(HostName),&DestinationIP, SL\_AF\_INET);

if(retVal == 0){

Addr.sin\_family = SL\_AF\_INET;

Addr.sin\_port = sl\_Htons(port);

Addr.sin\_addr.s\_addr = sl\_Htonl(DestinationIP);// IP to big endian

ASize = sizeof(SlSockAddrIn\_t);

SockID = -1;

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)){

uint32\_t copyIndex = 0;

strcpy(&SendBuff[copyIndex], method); copyIndex += strlen(method);

strcpy(&SendBuff[copyIndex], " "); copyIndex += 1;

strcpy(&SendBuff[copyIndex], request); copyIndex += strlen(request);

if(requestData1) {

strcpy(&SendBuff[copyIndex], requestData1);

copyIndex += strlen(requestData1);

}

if(requestData2) {

strcpy(&SendBuff[copyIndex], requestData2);

copyIndex += strlen(requestData2);

}

strcpy(&SendBuff[copyIndex], REQ\_1); copyIndex += strlen(REQ\_1);

strcpy(&SendBuff[copyIndex], hostName); copyIndex += strlen(hostName);

strcpy(&SendBuff[copyIndex], REQ\_2); copyIndex += strlen(REQ\_2);

SendBuff[copyIndex] = '\0';

UARTprintf("Sending request: %s\r\n\r\n", SendBuff);

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);

return Recvbuff;

}

}

return NULL;

}

char voltagestring\_buff[20];

char\* VoltageToString(uint32\_t sample) {

uint32\_t i;

voltagestring\_buff[19] = '\0';

for(i = 18; sample != 0; i--) {

voltagestring\_buff[i] = (char)('0' + sample % 10);

sample /= 10;

}

return &voltagestring\_buff[i+1];

}

/\*

\* Application's entry point

\*/

int main(void){

SlSecParams\_t secParams;

char \*pConfig = NULL;

initClk(); // PLL 50 MHz

UART\_Init(); // Send data to PC, 115200 bps

LED\_Init(); // initialize LaunchPad I/O

ADC0\_InitSWTriggerSeq3\_Ch9(); //initialize ADC sampler

ST7735\_InitR(INITR\_REDTAB);

ST7735\_SetCursor(1,1);

printf("Lab4C\n");

Wifi\_Connect(pConfig, &secParams);

UARTprintf("Weather App\n");

while(1){

LED\_GreenOn();

char \*weather\_data = HTTP\_Request(

"api.openweathermap.org", 80,

"GET", "/data/2.5/weather?q=Austin%20Texas&units=metric&APPID=d6e361f259c47a6ea9837d41b1856b03",

NULL,

NULL

);

LED\_GreenOff();

UARTprintf("\r\n\r\n");

UARTprintf(weather\_data); UARTprintf("\r\n");

ST7735\_SetCursor(0,4);

printf("Temp = %6s C\n", Extract\_Temperature(weather\_data));

uint32\_t sample = ADC0\_InSeq3();

LED\_GreenOn();

char \*send\_data = HTTP\_Request(

// embsysmooc or embedded-systems-server?

"embsysmooc.appspot.com", 80,

"GET", "/query?city=Austin%20Texas&id=John%20Starich%20and%20Jon%20Ambrose&edxcode=8086&greet=Voltage~",

VoltageToString(sample),

"V"

);

LED\_GreenOff();

UARTprintf("\r\n\r\n");

UARTprintf(send\_data); UARTprintf("\r\n");

printf("Voltage~%luV\n", sample);

LED\_GreenOn();

char \*custom = HTTP\_Request(

"tomcat.johnstarich.com", 80,

"GET", "/%22temp%22:1000,",

NULL,

NULL

);

LED\_GreenOff();

UARTprintf("\r\n\r\n");

UARTprintf(send\_data); UARTprintf("\r\n");

UARTprintf("Custom temp: %s", custom); UARTprintf("\r\n");

printf("Custom temp: %s\n", Extract\_Temperature(custom));

while(Board\_Input()==0){}; // wait for touch

}

}

/\*!

\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;

}

}

/\*

\* \* ASYNCHRONOUS EVENT HANDLERS -- End

\*/