#include <base64.h>

#include <ESP8266WiFi.h>

#include <SoftwareSerial.h>

#include <Ticker.h>

#include <RESTClient.h>

#include "./GarageDoor.h"

#include <ESP8266httpUpdate.h>

#include <ESP8266HTTPClient.h>

const char\* ssid = \_ssid;

const char\* fingerprint = "289509731da223e5218031c38108dc5d014e829b";

WiFiClientSecure client;

extern "C" {

#include "c\_types.h"

#include "ets\_sys.h"

#include "os\_type.h"

#include "osapi.h"

#include "mem.h"

#include "user\_interface.h"

#include "smartconfig.h"

#include "lwip/opt.h"

#include "lwip/err.h"

#include "lwip/dns.h"

}

boolean BLEChipStatus = false;

boolean BLEScanStatus = false;

boolean garageDoorStatus = false;

int bleChipFault = 0;

int noOfScanNamespaces = 0;

int noOfScanUUIDs = 0;

String instructionCode = "";

SoftwareSerial SoftSerial(4, 5); // receive, transmit, inverse\_logic, buffer size , false, 1536

HTTPClient http;

RestClient garageDoorRestClient = RestClient("34.83.30.196",80);

//RestClient garageDoorRestClient = RestClient(RESTServer, RESTPort);

Ticker esp\_heartbeat; // write to discrete logic (hardware logic to reset if esp hangs) every 120 secs

int esp\_hb = 120; // Write hb every 120 secs

boolean esp\_hb\_ticker\_triggered = false;

Ticker ble\_ticker; // ticker for ble chip status check

boolean ble\_chip\_status\_tickerTriggered = false;

int bleChippoll = 300; // ble chip status poll interval

Ticker nearbyBLE;

boolean nearbyBLE\_tickerTriggered = false;

int nearbyBLEpoll = 10;

Ticker sendEmail;

boolean sendEmail\_tickerTriggered = false;

int sendEmailPoll = 300;

Ticker incrCounter;

boolean incrCounter\_tickerTriggered = false;

int incrCounterPoll = 25;

// Ticker Func to enable nearbyBLE devices bool

void nearby\_ble\_ticker\_trigger() {

Serial.println("Turning get BLE frames trigger ON");

nearbyBLE\_tickerTriggered = true;

}

// Ticker Func to send email periodically

void sendEmail\_tickerTrigger() {

Serial.println("Turning sendEmail ticker ON");

sendEmail\_tickerTriggered = true;

}

// Ticker Func to increment counter

void incrCounter\_tickerTrigger() {

Serial.println("Turning incrCounter ticker ON");

incrCounter\_tickerTriggered = true;

}

// Ticker Func to write esp hb

void esp\_hb\_ticker\_trigger() {

Serial.println("Turning ESP heart beat write trigger ON");

esp\_hb\_ticker\_triggered = true;

}

// Ticker Func for ble chip check

void ble\_chip\_status\_tickerTrigger() {

Serial.println("Turning BLE chip status check trigger ON");

ble\_chip\_status\_tickerTriggered = true;

}

void write\_esp\_hb() {

if (esp\_hb\_ticker\_triggered == true) {

digitalWrite(16, HIGH);

delay(10);

digitalWrite(16, LOW);

}

}

void http\_ota\_update() {

String firmware\_update\_url = "http://"+String("34.83.30.196")+":"+String(80)+"/firmwareupdate"; // Use FQDN of cloud server

Serial.print("Updating firmware from ");

Serial.println(firmware\_update\_url.c\_str());

t\_httpUpdate\_return ret = ESPhttpUpdate.update(firmware\_update\_url.c\_str());

// if successful, ESP will restart

switch(ret) {

case HTTP\_UPDATE\_FAILED:

Serial.print("HTTP UPDATE FAILD Error ");

Serial.print(String(ESPhttpUpdate.getLastError())); // only available in esp arduino staging version

Serial.print(" ");

Serial.println(ESPhttpUpdate.getLastErrorString()); // only available in esp arduino staging version

break;

case HTTP\_UPDATE\_NO\_UPDATES:

Serial.println("HTTP UPDATE - NO UPDATE");

break;

case HTTP\_UPDATE\_OK:

Serial.println("HTTP UPDATE OK");

instructionCode = "";

break;

}

}

void check\_ble\_chip\_status() {

Serial.println("Checking BLE Chip Status");

for (int i=0; i<2; i++) { // check ble chip status twice

char\* nrf\_resp = nrf\_send\_recv("nrfstatus", true); // called with true. should free memory

if (nrf\_resp) {

Serial.print("BLE Chip Response = \""); Serial.print(String(nrf\_resp)); Serial.println("\"");

if(String(nrf\_resp) == "NRFOK") {

Serial.println("BLE Chip Status OK");

BLEChipStatus = true;

//change\_led\_status();

bleChipFault = 0;

free(nrf\_resp);

break;

} else {

free(nrf\_resp);

BLEChipStatus = false;

//change\_led\_status();

bleChipFault++;

if (bleChipFault > 10) {

Serial.println("ESP to BLE data path down. Resetting ESP");

ESP.restart();

}

delay(1000);

if (i == 0) {

Serial.println("Retrying...");

}

}

}

}

if (BLEChipStatus == false) {

Serial.println("WARNING!! BLE Chip NOT responding.");

}

if (bleChipFault > 2) {

Serial.println("ERROR!! BLE Chip NOT Available. Resetting chip...");

reset\_ble\_chip();

}

//sprint("\n", 2, "noline");

//timeStamped = false; // so that sprint will timestamp next print

Serial.println("Turning OFF BLE chip status check Ticker trigger");

ble\_chip\_status\_tickerTriggered = false;

}

void check\_ble\_chip\_scan\_status() {

Serial.println("Checking BLE Chip Scan Status");

for (int i=0; i<2; i++) { // check ble chip status twice

char\* nrf\_resp = nrf\_send\_recv("scanning?", true); // called with true. should free memory

if (nrf\_resp) {

Serial.print("BLE Chip Response = \""); Serial.print(String(nrf\_resp)); Serial.println("\"");

if(String(nrf\_resp) == "NRFOK") {

Serial.println("BLE Chip Scan Status OK");

BLEScanStatus = true;

//change\_led\_status();

bleChipFault = 0;

free(nrf\_resp);

break;

} else {

free(nrf\_resp);

BLEScanStatus = false;

Serial.println("WARNING!! BLE Chip NOT Scanning. Enabling...");

start\_ble\_scan();

bleChipFault++;

if (bleChipFault > 10) {

Serial.println("ESP to BLE data path down. Resetting ESP");

ESP.restart();

}

delay(1000);

if (i == 0) {

Serial.println("Retrying...");

}

}

}

}

if (BLEChipStatus == false) {

//change\_led\_status();

}

if (bleChipFault > 2) {

Serial.println("ERROR!! BLE Chip NOT Available. Resetting chip...");

reset\_ble\_chip();

}

}

void start\_ble\_scan() {

if (!BLEScanStatus) {

if (noOfScanNamespaces > 0) {

Serial.println("Sending Namespaces to BLE chip");

send\_namespaces\_uuids\_to\_ble("Namespace");

delay(200);

}

if (noOfScanUUIDs > 0) {

Serial.println("Sending UUIDs to BLE chip");

send\_namespaces\_uuids\_to\_ble("UUID");

delay(200);

}

int scanFault = 0;

Serial.println("Starting BLE scanning");

if (BLEChipStatus == true) {

for (int i=0; i<4; i++) { // try starting BLE scan more than once

char\* nrf\_resp = nrf\_send\_recv("blescanst", true);

if (nrf\_resp) {

Serial.print("BLE Chip Response = \""); Serial.print(String(nrf\_resp)); Serial.println("\"");

if(String(nrf\_resp) == "NRFOK") {

Serial.println("BLE Scan Initiated");

BLEScanStatus = true;

//change\_led\_status();

scanFault = 0;

free(nrf\_resp);

break;

} else {

free(nrf\_resp);

BLEScanStatus = false;

//change\_led\_status();

scanFault++;

if (i < 3) {

Serial.println("Retrying...");

delay(1000);

}

if (scanFault > 2) {

Serial.println("ERROR!! Not able to start BLE scan. Resetting chip...");

reset\_ble\_chip();

}

}

}

}

} else if (!BLEChipStatus) {

Serial.println("BLE chip is down. Could NOT enable BLE scan");

}

}

}

void post\_to\_cloud(String bleResponse) {

Serial.println("Posting BLE nearby devices list to cloud");

bleResponse = bleResponse + String(FIRMWARE\_VER);

//const char \*c = bleResponse.c\_str();

//int statusCode = garageDoorRestClient.post("/device", bleResponse.c\_str(), &resp); //This is for 3rd Party RESTClient library

http.begin(EXTERNAL\_IP\_ADDRESS);

int statusCode = http.POST(bleResponse.c\_str());

String resp = http.getString(); //Get the request response payload

http.end();

Serial.print("statusCode is "); Serial.println(String(statusCode));

if (statusCode != 9999 && statusCode != 200) {

Serial.print("ERROR !! RESTClient POST \"NEARBY\" - Response from server:"); Serial.println(resp);

} else if (statusCode == 9999) {

Serial.print("ERROR !! RESTClient POST \"NEARBY\" - TCP connection timeout");

WiFiClient::stopAll();

} else if (statusCode == 200) {

Serial.print("Response from REST server is "); Serial.println(resp);

instructionCode = resp;

}

if (instructionCode == "firmwareupdate") {

Serial.println("Received instruction for firmware upgrade");

http\_ota\_update();

}

}

boolean get\_nearby\_ble\_frames() {

boolean ble\_resp = false;

char nearbyBLEdevicesBuf[1024];

for (int k=0;k<18;k++) {

//Serial.print("Outer iter "); Serial.println(String(k));

for (int bPtr = 0; bPtr<sizeof(nearbyBLEdevicesBuf); bPtr++) {

memset(&nearbyBLEdevicesBuf[bPtr], 0, sizeof(nearbyBLEdevicesBuf[bPtr]));

}

const char header[] = "NEARBY BLE DEVICES LIST, ";

strcpy(nearbyBLEdevicesBuf, header);

//strcpy(nearbyBLEdevicesBuf, "NEARBY BLE DEVICES LIST, ");

int i = sizeof(header) - 1;

//Serial.print("Header size = "); Serial.println(String(i));

char \* cmd = "bleframes";

/\* nRF sends 10 packets at a time to save on serial buf

\* nRF buf size = 180 packets each ~ 50 bytes (~100 bytes when sending as HEX)

\* ESP input buffer size (nearbyBLEdevicesBuf) = 2600. Beyong 2600 REST/TCP library has issues and is timing out

\* so send bleframes command 3 times (to limit to 2600) and receive max 30 frames, repeat this 7 times

\* TBD - need a handshake as to how many has nRF buffered

\* may be nRF should first send the count (# of packed in its buffer)

\*/

//Serial.print("Sending command \""); Serial.print(String(cmd)); Serial.println("\" to nRF");

int j = 0;

// 10 sec timeout for while loop

unsigned long startTime = millis();

while (j<1) {

//Serial.print("Inner iter "); Serial.println(String(j));

// send command to nRF

//SoftSerial.flush();

//delay(200);

SoftSerial.write(cmd, 9);

//Serial.println("Sending command to BLE chip");

SoftSerial.flush();

delay(200);

// catch any command errors from nRF

char errorCode[6];

int recvPtr = 0;

int bufStartPtr = i;

// recv from nRF. TBD - nRF is sending HEX now (uses double memory & bandwidth). This needs to change to binary all the way to backend.

// waiting for backend to support converting binary to HEX. MQTT also can be used to solve this

//Serial.println("nRF:");

while(SoftSerial.available() > 0) {

ble\_resp = true;

char inChar = SoftSerial.read();

//Serial.print(String(inChar)); Serial.print(" ");

//delay(1); // REMOVE CHANGE - working even without delay

nearbyBLEdevicesBuf[i] = inChar;

//Serial.println(String(nearbyBLEdevicesBuf[i]));

i++;

// capture first 5 chars into errorCode

if (recvPtr<5) {

errorCode[recvPtr] = inChar;

recvPtr++;

} else if (recvPtr == 5) {

errorCode[5] = '\0';

//Serial.print("BLE Chip response is "); Serial.println(String(errorCode));

}

}

// if command went trough go to next iter

if (strcmp(errorCode, "ERROR") != 0) {

j++;

} else if (strcmp(errorCode, "ERROR") == 0) {

Serial.println("WARNING !! BLE Chip command time out. Retrying...");

i = bufStartPtr;

//Serial.print("Buffer pointer @ "); Serial.println(String(i));

}

// if multiple command fails the break after timeout

if (millis() - startTime > 10000) {

Serial.println("ERROR !! BLE Chip Command timed out");

break;

}

}

String tempBLEdevicesBuf = String(nearbyBLEdevicesBuf);

String distNum = tempBLEdevicesBuf.substring(57,59);

Serial.println(distNum);

// null terminate char array

nearbyBLEdevicesBuf[i] = '\0';

//Serial.print("Data for REST POST = \""); Serial.print(nearbyBLEdevicesBuf); Serial.println("\"");

if (nearbyBLEdevicesBuf[sizeof(header) - 1] != '\0' && (String(nearbyBLEdevicesBuf) != "NEARBY BLE DEVICES LIST, NO FRAMES CAPTURED")) {

Serial.print("FRAME CAPTURED = \""); Serial.print(nearbyBLEdevicesBuf); Serial.println("\"");

post\_to\_cloud(tempBLEdevicesBuf);

Serial.println("Checking door status");

//PRANAV

// Code to check signal strength and set garageDoorStatus boolean status

} else {

Serial.println("No BLE packets captured");

}

}

return ble\_resp;

}

char\* nrf\_send\_recv(char\* cmd, boolean return\_resp) { // NOTE if called with return\_resp = true calling func should free memory

boolean ble\_resp = false;

char\* nRFReply = (char\*)malloc(6);

int i = 0;

if (strcmp(cmd, "0") != 0) {

// send to nRF

Serial.print("Sending command \""); Serial.print(String(cmd)); Serial.println("\" to nRF");

// TBD - remove all unwanted flush calls. Implement handshake between ESP and BLE chip

SoftSerial.flush();

delay(200);

SoftSerial.write(cmd, 9);

SoftSerial.flush();

}

delay(100);

// recv from nRF

Serial.print("nRF:");

boolean nl = false;

// TBD - loop through until timeout and break when EOF seen from nRF

//unsigned long startTime = millis();

//while(true) {

while(SoftSerial.available() > 0) {

ble\_resp = true;

char inChar = SoftSerial.read();

delay(1);

if (inChar == 0x0A) {

Serial.println(String(""));

nl = true;

} else {

if (nl == true) {

Serial.print("nRF:");

nl = false;

}

Serial.print(String(inChar));

if (i<5) {

nRFReply[i] = inChar;

i++;

}

}

}

//if (millis() - startTime > 2000) {

// break;

//}

//}

if (ble\_resp == false) {

Serial.println("No response");

}

nRFReply[5] = '\0';

if (return\_resp == true) {

return nRFReply;

} else {

Serial.println("Freeing memory");

free(nRFReply);

}

}

void send\_namespaces\_uuids\_to\_ble(String type) {

boolean dataToSend = false;

if (type == "Namespace" && noOfScanNamespaces > 0) {

Serial.println("Sending namespaces to BLE chip");

dataToSend = true;

}

else if (type == "UUID" && noOfScanUUIDs > 0) {

Serial.println("Sending UUIDs to BLE chip");

dataToSend = true;

}

if (BLEChipStatus && dataToSend) {

// send command to BLE chip

Serial.println("Sending command xferuuids to BLE chip");

nrf\_send\_recv("xferuuids");

if (type == "Namespace") {

// TBD Write a separate function to send data and pass the struct element pointer

// send no of elements being sent to BLE chip

Serial.print("No of namespaces being sent = "); Serial.println(String(noOfScanNamespaces));

SoftSerial.write(noOfScanNamespaces);

SoftSerial.flush();

// read reply

nrf\_send\_recv("0");

// send data

for (int i=0; i<noOfScanNamespaces; i++) {

if (sizeof(scanOnly[i].uidNamespace) != 0) {

Serial.print("Length of Namespace to broadcast is :");

Serial.println(String(sizeof(scanOnly[i].uidNamespace)));

// send length to BLE chip

SoftSerial.write(sizeof(scanOnly[i].uidNamespace));

SoftSerial.flush();

// read reply

nrf\_send\_recv("0");

// send frame bytes to BLE chip

Serial.print("Data to send is :");

for (int j=0; j<sizeof(scanOnly[i].uidNamespace); j++) {

Serial.print(String(scanOnly[i].uidNamespace[j], HEX));

Serial.print(" ");

SoftSerial.write(scanOnly[i].uidNamespace[j]);

delay(3);

}

SoftSerial.flush();

Serial.print("\n");

Serial.println("Done sending data to nRF");

delay(100);

// read reply

nrf\_send\_recv("0");

} else {

Serial.println("ERROR - Namespaces empty");

}

}

} else if (type = "UUID") {

// TBD Write a separate function to send data and pass the struct element pointer

// send no of elements being sent to BLE chip

Serial.print("No of UUIDs being sent = "); Serial.println(String(noOfScanUUIDs));

SoftSerial.write(noOfScanUUIDs);

SoftSerial.flush();

// read reply

nrf\_send\_recv("0");

// send data

for (int i=0; i<noOfScanUUIDs; i++) {

if (sizeof(scanOnly[i].uuid) != 0) {

Serial.print("Length of Namespace to broadcast is :");

Serial.println(String(sizeof(scanOnly[i].uuid)));

// send length to BLE chip

SoftSerial.write(sizeof(scanOnly[i].uuid));

SoftSerial.flush();

// read reply

nrf\_send\_recv("0");

// send frame bytes to BLE chip

Serial.print("Data to send is :");

for (int j=0; j<sizeof(scanOnly[i].uuid); j++) {

Serial.print(String(scanOnly[i].uuid[j], HEX));

Serial.print(" ");

SoftSerial.write(scanOnly[i].uuid[j]);

delay(3);

}

SoftSerial.flush();

Serial.print("\n");

Serial.println("Done sending data to nRF");

delay(100);

// read reply

nrf\_send\_recv("0");

} else {

Serial.println("ERROR - Namespaces empty");

}

}

}

} else if (!BLEChipStatus) {

Serial.println("ERROR!! BLE Chip NOT Ready");

} else if (noOfScanNamespaces == 0 || noOfScanUUIDs == 0) {

Serial.println("ERROR!! Namespaces or UUIDs empty. Nothing to send");

}

}

void reset\_ble\_chip() {

digitalWrite(15, LOW);

delay(10);

digitalWrite(15, HIGH);

Serial.println("WARNING!! GPIO 15 set to HIGH as part of BLE chip reset.");

}

void send\_email() {

//PRANAV

// write code to check a counter and send email if counter if greater than x

}

void door\_open\_duration() {

//PRANAV

// write code to check if door open and increment counter up until 12; if door closed, counter = 0

}

/\*

int emailFnc() {

//testing section for email

Serial.println("Attempting to connect to GMAIL server");

if (client.connect(\_GMailServer, 465) == 1) {

Serial.println(F("Connected"));

} else {

Serial.print(F("Connection failed:"));

return 0;

}

if (!response())

return 0;

Serial.println(F("Sending EHLO"));

client.println("EHLO gmail.com");

if (!response())

return 0;

Serial.println(F("Sending auth login"));

client.println("auth login");

if (!response())

return 0;

Serial.println(F("Sending User"));

client.println(base64::encode(\_mailUser));

if (!response())

return 0;

Serial.println(F("Sending Password"));

client.println(base64::encode(\_mailPassword));

if (!response())

return 0;

Serial.println(F("Sending From"));

client.println(F("MAIL FROM: <pranaavn@gmail.com>"));

if (!response())

return 0;

Serial.println(F("Sending To"));

client.println(F("RCPT To: <pranaavn@gmail.com>"));

if (!response())

return 0;

Serial.println(F("Sending DATA"));

client.println(F("DATA"));

if (!response())

return 0;

Serial.println(F("Sending email"));

client.println(F("To: GarageOS<bugsbunny@made.up>"));

client.println(F("From: hallo@gmail.com"));

client.println(F("Subject: GarageOS Alert\r\n"));

client.println(F("This email was sent securely via an encrypted mail link.\n"));

client.println(F("Garage Currently Open"));

client.println(F("."));

if (!response())

return 0;

Serial.println(F("Sending QUIT"));

client.println(F("QUIT"));

if (!response())

return 0;

client.stop();

Serial.println(F("Disconnected"));

return 1;

//end testing section

}

\*/

void setup() {

// put your setup code here, to run once:

// Cleanup any WiFi configs

// turn led on to indicate setup

pinMode(STATUS\_LED, OUTPUT);

digitalWrite(STATUS\_LED, HIGH);

// moved BLE chip bring up to after serial initialization for using sprint

// ESP heart beat pin set to OUTPUT

pinMode(16, OUTPUT);

// Turn ON ESP heart beat write

esp\_heartbeat.attach(esp\_hb, esp\_hb\_ticker\_trigger);

WiFi.softAPdisconnect(true); // disconnect ESP AP

WiFi.disconnect(); // disconnect "from" AP

delay(1000);

// Open serial communications and wait for port to open:

Serial.begin(115200);

delay(1);

unsigned long startTime = millis();

while (!Serial) {

if (millis() - startTime > 20000) {

Serial.println("ERROR!!: NOT able to initialize serial. Rebooting...");

ESP.restart();

break;

}

; // wait for serial port

}

delay(100);

// print image version

Serial.print("Firmware version is ");

Serial.println(String(FIRMWARE\_VER));

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(\_ssid, \_password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("My IP address: ");

Serial.println(WiFi.localIP());

// BLE chip reset pin default HIGH. Write LOW to reset nRF

Serial.println("Bringing up BLE chip");

pinMode(15, OUTPUT);

digitalWrite(15, LOW);

delay(10);

digitalWrite(15, HIGH);

// Soft serial init

Serial.println("Inititlizing soft serial");

SoftSerial.begin(38400);

startTime = millis();

while (!SoftSerial) {

if (millis() - startTime > 20000) {

Serial.println("ERROR!!: NOT able to initialize SOFT serial. Rebooting...");

ESP.restart();

break;

}

; // wait for serial port

}

Serial.println("Waiting for BLE chip to boot...");

delay(10000);

garageDoorRestClient.setContentType("text/plain");

garageDoorRestClient.setUserAgent("ESP-8266/GarageOS");

check\_ble\_chip\_status();

/\*WiFi.begin(CUST\_SSID, PASS);

Serial.print("Connecting to WiFi AP");

startTime = millis();

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

if (millis() - startTime > WIFI\_STA\_TIMEOUT) {

//ESP.reset();

Serial.print("ERROR !! Timeout connecting to WiFi AP");

break;

}

}\*/

// Define namespace & UUID filter

uint8\_t uuid1[] = {0x6F, 0x37, 0xDA, 0x6F, 0xB9, 0x7B, 0x40, 0x9F, 0xAA, 0x61, 0xBF, 0x54, 0xB8, 0xE9, 0xAE, 0x4E};

memcpy(scanOnly[0].uuid, uuid1, sizeof uuid1);

noOfScanUUIDs++;

uint8\_t uuid2[] = {0xD5, 0xC4, 0x9A, 0x56, 0x0C, 0x18, 0x4D, 0x42, 0x8C, 0x57, 0x16, 0x42, 0x9D, 0xAD, 0x26, 0xB3};

memcpy(scanOnly[1].uuid, uuid2, sizeof uuid2);

noOfScanUUIDs++;

esp\_heartbeat.attach(esp\_hb, esp\_hb\_ticker\_trigger);

ble\_ticker.attach(bleChippoll, ble\_chip\_status\_tickerTrigger);

nearbyBLE.attach(nearbyBLEpoll, nearby\_ble\_ticker\_trigger);

//sendEmail.attach(sendEmailPoll, sendEmail\_tickerTrigger);

incrCounter.attach(incrCounterPoll, incrCounter\_tickerTrigger);

}

void loop() {

// put your main code here, to run repeatedly:

if (nearbyBLE\_tickerTriggered == true) {

if (BLEScanStatus == true) {

if (!get\_nearby\_ble\_frames()) {

check\_ble\_chip\_scan\_status();

// print image version

Serial.print("Firmware version is ");

Serial.println(String(FIRMWARE\_VER));

}

//nrf\_send\_recv("bleframes", false); // To print frames on console rather than to send to cloud

nearbyBLE\_tickerTriggered = false;

} else {

start\_ble\_scan();

nearbyBLE\_tickerTriggered = false;

}

}

/\*if (sendEmail\_tickerTriggered == true) {

// call fnctn to sendEmail here

send\_email();

sendEmail\_tickerTriggered = false;

}\*/

if (incrCounter\_tickerTriggered == true) {

// call fnctn to increment counter here if garage door is open

door\_open\_duration();

incrCounter\_tickerTriggered = false;

}

// Write ESP heart beat

write\_esp\_hb();

//<new>

esp\_hb\_ticker\_triggered = false;

//</new>

} // END LOOP