Aaron Bading

8/11/21

CODE

Code was developed in Visual Studio code with PlatformIO

// **********INSIDE UNIT/ Receiver code Below **********

```
#include <Arduino.h>

#include <esp_now.h>
#include <WiFi.h>

#include "pitches.h"
#include "songs.h"

 #define SPEAKER_PIN 0 // this is the pin that will be referred to in the sketch
 #define OUTPUT_PIN 25 // this is the actual physical pin the buzzer will be conn
ected to

TaskHandle_t task1; // create task for core 0
TaskHandle_t task2; // create task for core 1
void core0(void *pvParameter); // this is the method core 0 will call
void core1(void *pvParameter); // this is the method core 1 will call

// Must match the sender structure
typedef struct struct_message {
   bool buzzer;
    bool timer;
} struct_message;

// Create a struct_message called myData
struct_message myData;

int internalLed = 2;
int externalLed1 = 4;  // green led for timer
int externalLed2 = 5; // blue led for buzzer

// for timer functionality
unsigned long prevMillis;
unsigned long currentMillis;
const unsigned long interval = 40000;  //the value is a number of milliseconds
```

```
//blink function
void blink()
{
  digitalWrite(internalLed ,HIGH);
delay(100);
digitalWrite(internalLed ,LOW);
delay(100);
}

//play song function
void play_song(int num_notes, int melody[], int noteDurations[], int tempo) {
  // step through and play all of the notes
  for (int i=0; i<num_notes; i++) {
    int duration = 0;
    // calculate the duration of each note
    if (noteDurations[i] > 0) {
      duration = tempo / noteDurations[i];
    }
    // if it's a negative number, means dotted note
    // increases the duration by half for dotted notes
    else if (noteDurations[i] < 0) {
      duration = tempo / abs(noteDurations[i]) * 1.5;
    }

   // tone(SPEAKER_PIN, melody[i], duration);
     ledcWriteTone(0,melody[i]);
     delay(duration);



    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = duration * 1.20;
    delay(pauseBetweenNotes);
    // stop the tone playing:

    //noTone(SPEAKER_PIN);
         ledcWriteTone(0,0);
  }
}

// callback function that will be executed when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  blink();
  memcpy(&myData, incomingData, sizeof(myData));
```

```
  Serial.print("Bytes received: ");
  Serial.println(len);
  Serial.print("Bool buzzer: ");
  Serial.println(myData.buzzer);
    Serial.print("Bool timer: ");
  Serial.println(myData.timer);
  Serial.println();
}

 //switch function to play a certain song
 bool switchfunction(int num)
 {
   switch (num) {
        case 0:
          play_song(haircutLength,haircut,haircutDurations,haircutTempo);
          break;
        case 1:
          play_song(marioLength,mario,marioDurations,marioTempo);
          break;
        case 2:
          play_song(miiLength,mii,miiDurations,miiTempo);
          break;
        case 3:
          play_song(hpLength,hp,hpDurations,hpTempo);
          break;
        case 4:
          play_song(takeonmeLength,takeonme,takeonmeDurations,takeonmeTempo);
          break;
        default:
          play_song(miiLength,mii,miiDurations,miiTempo);
          break;
}
 }
void setup() {

  xTaskCreatePinnedToCore(
core0, // name of the function that will run on specific core
"task1", // name of the task
1000, // size of the stack for task1
NULL, // parameters for task1
1, // priority 1 is the highest
&task1 , // this is the handle we have created
0 ); // core zero
xTaskCreatePinnedToCore(
core1, // name of the function that will run on specific core
```

```
"task2", // name of the task
1000, // size of the stack for task1
NULL, // parameters for task1
1, // priority 1 is the highest
&task2 , // this is the handle we have created
1 ); // core zero

// this is for the buzzer
   ledcSetup(0,1E5,12);
  ledcAttachPin(OUTPUT_PIN,SPEAKER_PIN);  // attach pin 25 to 0.

// set them to false intially
  myData.buzzer = false;
  myData.timer = false;
  Serial.begin(115200);

  pinMode(internalLed , OUTPUT);
  pinMode(externalLed1 , OUTPUT);
  pinMode(externalLed2 , OUTPUT);

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Once ESPNow is successfully Init, we will register for recv CB to
  // get recv packer info
  esp_now_register_recv_cb(OnDataRecv);
  prevMillis = millis();  //initial start time
}

void core0(void *pvParameter)
{

while(1)
{
  bool decision = true;
  int song_choice = 0;
 if(myData.buzzer)
      {// if buzzer was set to true , make buzzer go off
      if(decision)
```

```
      {
        song_choice = random(5);
        decision = false;
      }
           switchfunction(song_choice);

        digitalWrite(externalLed1,1);
      }
      else
      { // if buzzer was set to false, make buzzer turn off
          digitalWrite(externalLed1,0);
          decision = true;
      }
// Serial.print("In core0 I am running on core : ");
// Serial.println(xPortGetCoreID());
// delay(1000);

}

}
 void core1(void *pvParameter)
{
  prevMillis = millis();
  while(1)
  {
    currentMillis = millis();
      if(myData.timer)
      {// if timer was turned on then we are timing this and turning buzzer off a
fter that time
        digitalWrite(externalLed2,1);

        if (currentMillis - prevMillis >= interval)
          {
            myData.buzzer = false;  // turn off buzzer and timer
            myData.timer = false;
            prevMillis = currentMillis;
          }
      }
      else
      { // if timer is not on then dont do anything.
          digitalWrite(externalLed2,0);
          prevMillis = currentMillis;
      }

    //Serial.print("In core1 I am running on core : ");
```

```
    //Serial.println(xPortGetCoreID());

  }
}

void loop() {
// do nothing here..
}
```

// **********INSIDE UNIT/ Receiver code Above **********

// ******************OUTSIDE UNIT/ Transmitter code Below **************************

```
#include <Arduino.h>

#include "esp_camera.h"
#include <WiFi.h>
#include "esp_timer.h"
#include "img_converters.h"
#include "fb_gfx.h"
#include "soc/soc.h"             // disable brownout problems
#include "soc/rtc_cntl_reg.h"    // disable brownout problems
#include "esp_http_server.h"

//-------------------- esp now
#include <esp_now.h>
#include <WiFi.h>

// address of inside ESP32
uint8_t broadcastAddress[] = {0x7C, 0x9E, 0xBD, 0x47, 0xCB, 0x98};


// data structure that we are sending via esp-now
typedef struct struct_message {
  bool buzzer;
  bool timer;
} struct_message;

struct_message myData; // instance of that structure

// this function will send via esp-now to the inside esp32
void send()
{
   esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(
myData));

  if (result == ESP_OK) {
    Serial.println("Sent with success");
  }
  else {
    Serial.println("Error sending the data");
  }
}

// callback when data is sent
```

```
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery
Fail");
}

//----------------------- esp now

// Replace with network credentials that you want
const char* ssid = "ESP32CAM_STREAM";
const char* password = "123456789";

//******* for Wake up *********
const int wakepin = 2;
const int led_flash = 4;

#define EXT_WAKEUP_PIN_BITMASK 0x0004  //  2^2 for GPIO 2

RTC_DATA_ATTR int bootCount = 0;   // this will store the bootcount through the w
ake up and sleep cycles
unsigned long prevMillis;  //some global variables available anywhere in the prog
ram
unsigned long currentMillis;
const unsigned long interval = 60000;  //the value is a number of milliseconds
bool startTimer = false;  // control for timing
void gotoSleep();


#define PART_BOUNDARY "123456789000000000000987654321"
#define CAMERA_MODEL_AI_THINKER
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WITHOUT_PSRAM
//#define CAMERA_MODEL_M5STACK_PSRAM_B
//#define CAMERA_MODEL_WROVER_KIT

#if defined(CAMERA_MODEL_AI_THINKER)
  #define PWDN_GPIO_NUM     32
  #define RESET_GPIO_NUM    -1
  #define XCLK_GPIO_NUM      0
  #define SIOD_GPIO_NUM     26
  #define SIOC_GPIO_NUM     27

  #define Y9_GPIO_NUM       35
  #define Y8_GPIO_NUM       34
  #define Y7_GPIO_NUM       39
```

```
  #define Y6_GPIO_NUM       36
  #define Y5_GPIO_NUM       21
  #define Y4_GPIO_NUM       19
  #define Y3_GPIO_NUM       18
  #define Y2_GPIO_NUM        5
  #define VSYNC_GPIO_NUM    25
  #define HREF_GPIO_NUM     23
  #define PCLK_GPIO_NUM     22
#endif

// HTML, Server , Stream setup below -----------
static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-
replace;boundary=" PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-
Length: %u\r\n\r\n";

httpd_handle_t camera_httpd = NULL;
httpd_handle_t stream_httpd = NULL;

// html that will be used on local servers "website"
static const char PROGMEM INDEX_HTML[] = R"rawliteral(
<html>
  <head>
    <title>T-DETECTOR</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
      body { font-family: Arial; text-align: center; margin:0px auto; padding-
top: 30px;}
      table { margin-left: auto; margin-right: auto; }
      td { padding: 8 px; }
      .button {
        background-color: #2f4468;
        border: none;
        color: white;
        padding: 10px 20px;
        text-align: center;
        text-decoration: none;
        display: inline-block;
        font-size: 18px;
        margin: 6px 3px;
        cursor: pointer;
        -webkit-touch-callout: none;
        -webkit-user-select: none;
        -khtml-user-select: none;
```

```
        -moz-user-select: none;
        -ms-user-select: none;
        user-select: none;
        -webkit-tap-highlight-color: rgba(0,0,0,0);
      }
      img {  width: auto ;
        max-width: 100% ;
        height: auto ;
      }
    </style>
  </head>
  <body>
    <h1>T-DETECTOR</h1>
    <img src="" id="photo" >
    <table>
      <tr><td colspan="3" align="center"><button class="button" onmousedown="togg
leCheckbox('top');" ontouchstart="toggleCheckbox('top');">KEEP STREAM</button></t
d></tr>
      <tr><td colspan="3" align="center"><button class="button" onmousedown="togg
leCheckbox('middle');" ontouchstart="toggleCheckbox('middle');">STOP BUZZER</butt
on></td></tr>
      <tr><td colspan="3" align="center"><button class="button" onmousedown="togg
leCheckbox('bottom');" ontouchstart="toggleCheckbox('bottom');">GO TO SLEEP</butt
on></td></tr>
    </table>
   <script>
   function toggleCheckbox(x) {
     var xhr = new XMLHttpRequest();
     xhr.open("GET", "/action?go=" + x, true);
     xhr.send();
   }
   window.onload = document.getElementById("photo").src = window.location.href.sl
ice(0, -1) + ":81/stream";
  </script>
  </body>
</html>
)rawliteral";

static esp_err_t index_handler(httpd_req_t *req){
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, (const char *)INDEX_HTML, strlen(INDEX_HTML));
}

static esp_err_t stream_handler(httpd_req_t *req){
  camera_fb_t * fb = NULL;
```

```
  esp_err_t res = ESP_OK;
  size_t _jpg_buf_len = 0;
  uint8_t * _jpg_buf = NULL;
  char * part_buf[64];

  res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
  if(res != ESP_OK){
    return res;
  }

  while(true){
    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera capture failed");
      res = ESP_FAIL;
    } else {
      if(fb->width > 400){
        if(fb->format != PIXFORMAT_JPEG){
          bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
          esp_camera_fb_return(fb);
          fb = NULL;
          if(!jpeg_converted){
            Serial.println("JPEG compression failed");
            res = ESP_FAIL;
          }
        } else {
          _jpg_buf_len = fb->len;
          _jpg_buf = fb->buf;
        }
      }
    }
    if(res == ESP_OK){
      size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART, _jpg_buf_len);
      res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
    }
    if(res == ESP_OK){
      res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
    }
    if(res == ESP_OK){
      res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(_STREAM_BOUNDARY)
);
    }
    if(fb){
      esp_camera_fb_return(fb);
      fb = NULL;
```

```
      _jpg_buf = NULL;
    } else if(_jpg_buf){
      free(_jpg_buf);
      _jpg_buf = NULL;
    }
    if(res != ESP_OK){
      break;
    }
    //Serial.printf("MJPG: %uB\n",(uint32_t)(_jpg_buf_len));
  }
  return res;
}

static esp_err_t cmd_handler(httpd_req_t *req){
  char*  buf;
  size_t buf_len;
  char variable[32] = {0,};

  buf_len = httpd_req_get_url_query_len(req) + 1;
  if (buf_len > 1) {
    buf = (char*)malloc(buf_len);
    if(!buf){
      httpd_resp_send_500(req);
      return ESP_FAIL;
    }
    if (httpd_req_get_url_query_str(req, buf, buf_len) == ESP_OK) {
      if (httpd_query_key_value(buf, "go", variable, sizeof(variable)) == ESP_OK)
 {
      } else {
        free(buf);
        httpd_resp_send_404(req);
        return ESP_FAIL;
      }
    } else {
      free(buf);
      httpd_resp_send_404(req);
      return ESP_FAIL;
    }
    free(buf);
  } else {
    httpd_resp_send_404(req);
    return ESP_FAIL;
  }

  sensor_t * s= esp_camera_sensor_get();
```

```
  int res = 0;

  if(!strcmp(variable, "top")) {
        // do something when KEEP STREAM is pressed
        // tell other esp32 to stop its timer
      myData.timer = false;
      send();
      delay(1000);
      startTimer=false;  // stop the timer so stream continues until manually sto
pped
  }
  else if(!strcmp(variable, "middle")) {
      // do something when Stop buzzer is pressed
      // tell the other esp32 to stop the buzzer
    myData.buzzer = false;
    send();
    delay(1000);

  }
  else if(!strcmp(variable, "bottom")) {
  // do something when GO SLEEP is pressed
  // tell other esp32 to go to sleep : 11
     myData.buzzer=false;
     myData.timer =false;
     send();
     gotoSleep();

  }
  else {
    res = -1;
  }

  if(res){
    return httpd_resp_send_500(req);
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, NULL, 0);
}

void startCameraServer(){
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  config.server_port = 80;
  httpd_uri_t index_uri = {
```

```
    .uri       = "/",
    .method    = HTTP_GET,
    .handler   = index_handler,
    .user_ctx  = NULL
  };

  httpd_uri_t cmd_uri = {
    .uri       = "/action",
    .method    = HTTP_GET,
    .handler   = cmd_handler,
    .user_ctx  = NULL
  };
  httpd_uri_t stream_uri = {
    .uri       = "/stream",
    .method    = HTTP_GET,
    .handler   = stream_handler,
    .user_ctx  = NULL
  };
  if (httpd_start(&camera_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(camera_httpd, &index_uri);
    httpd_register_uri_handler(camera_httpd, &cmd_uri);
  }
  config.server_port += 1;
  config.ctrl_port += 1;
  if (httpd_start(&stream_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &stream_uri);
  }
}

// HTML, server , Stream setup above -----------

//function that will make esp-cam board go to sleep
void gotoSleep() {
  Serial.println("Deep sleep enabled");
  esp_sleep_enable_ext1_wakeup(EXT_WAKEUP_PIN_BITMASK, ESP_EXT1_WAKEUP_ANY_HIGH);
 //esp_sleep_enable_ext0_wakeup(GPIO_NUM_12,1); //1 = High, 0 = Low
  esp_deep_sleep_start();
}

void setup() {
    pinMode(wakepin, INPUT);
    pinMode(led_flash, OUTPUT);

  WiFi.mode(WIFI_STA);
```

```
  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
    esp_now_register_send_cb(OnDataSent);

// Register peer
  esp_now_peer_info_t peerInfo;
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }

  //------------
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector

  Serial.begin(115200);
  Serial.setDebugOutput(false);

 // camera configuration setup
   camera_config_t camconfig;
  camconfig.ledc_channel = LEDC_CHANNEL_0;
  camconfig.ledc_timer = LEDC_TIMER_0;
  camconfig.pin_d0 = Y2_GPIO_NUM;
  camconfig.pin_d1 = Y3_GPIO_NUM;
  camconfig.pin_d2 = Y4_GPIO_NUM;
  camconfig.pin_d3 = Y5_GPIO_NUM;
  camconfig.pin_d4 = Y6_GPIO_NUM;
  camconfig.pin_d5 = Y7_GPIO_NUM;
  camconfig.pin_d6 = Y8_GPIO_NUM;
  camconfig.pin_d7 = Y9_GPIO_NUM;
  camconfig.pin_xclk = XCLK_GPIO_NUM;
  camconfig.pin_pclk = PCLK_GPIO_NUM;
  camconfig.pin_vsync = VSYNC_GPIO_NUM;
  camconfig.pin_href = HREF_GPIO_NUM;
  camconfig.pin_sscb_sda = SIOD_GPIO_NUM;
  camconfig.pin_sscb_scl = SIOC_GPIO_NUM;
  camconfig.pin_pwdn = PWDN_GPIO_NUM;
  camconfig.pin_reset = RESET_GPIO_NUM;
```

```
  camconfig.xclk_freq_hz = 20000000;
  camconfig.pixel_format = PIXFORMAT_JPEG;

  if(psramFound()){
    camconfig.frame_size = FRAMESIZE_VGA;
    camconfig.jpeg_quality = 10;
    camconfig.fb_count = 2;
  } else {
    camconfig.frame_size = FRAMESIZE_SVGA;
    camconfig.jpeg_quality = 12;
    camconfig.fb_count = 1;
  }

  // Camera init
  esp_err_t err = esp_camera_init(&camconfig);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
  // Wi-Fi connection
  WiFi.softAP(ssid, password);

bootCount++; // increment bootcount
prevMillis = millis();  //initial start time
  startTimer = false;
  Serial.print("bootcount");
  Serial.println(bootCount);
  if(bootCount >= 2)
  {// this is the case when the board wakes up
  myData.buzzer = true;
  myData.timer = true;
  send(); // turn on the buzzer and start the timer for inside ESP32
    //start timer that will force going to sleep after some time.
  startTimer = true;

  sensor_t * s= esp_camera_sensor_get(); // this flips the stream 180 degrees

  // mirror effect
  //s->set_hmirror(s, 1);          // 0 = disable , 1 = enable
  s->set_vflip(s, 1);
  startCameraServer();
Serial.println("started server");
  }
  else
  {
```

```
    gotoSleep(); // go to sleep if everything was reset
  }

  Serial.println(WiFi.localIP());

  // Start streaming web server
  //startCameraServer();
}

void loop() {

if(startTimer)
{ // if we started the timer then the board will go to sleep after predetermined
time : 60 seconds
   currentMillis = millis();
  if (currentMillis - prevMillis >= interval)
  {
    gotoSleep();
    prevMillis = currentMillis;
  }
}

}
```

// **********OUTSIDE UNIT/ Transmitter code Above **********