University of Texas at El Paso

Electrical and Computer Engineering Department

EE4178/EE5190 Laboratory for Microprocessors II

LAB 06

WIFI and IOT: Servo Motor Control

Goals:

- Using the provided code, create a soft access point. Use your last name as the ssid and your ID as password.
- Initialize a pwm channel to be able to control a servo motor from a website.
- Edit the `http_server_netconn_serve` so that you can control a servo motor using the buttons in the UI.
- The webpage included in the code is shown in Figure 1.

Bonus:

Create your own HTML page for the server. **+20**

Pre-Lab Questions:

- What is the frequency needed to drive a servo motor?

Written by Hector Mota. Modified by Dr. Erives & Mirza Elahi September 2021

- What are the duty cycles that a servo takes and what angles do this duty cycles are?

- By studying the HTML page, what are the get commands that each button sends?

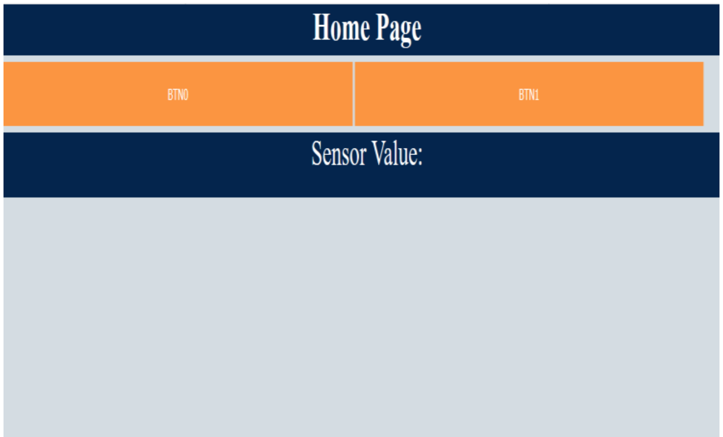

Figure 1. Webpage interface provided with the code.

```
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_event_loop.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "driver/gpio.h"
#include "driver/ledc.h"

#include "lwip/sys.h"
#include "lwip/netdb.h"
#include "lwip/api.h"

const static char http_html_hdr[] = "HTTP/1.1 200 OK\r\nContent-type: text/html\r\n\r\n";
const static char http_html_hdr1[] = "HTTP/1.1 200 OK\r\nContent-type: text/plain\r\n\r\n";
const static char http_index_hml[] = R"=====(<!DOCTYPE html><html> <head> <meta charset = UTF-8
name = "viewport" content = "width = device-width initial-scale = 1.0"> <title>Home Page</title> </head>
<body> <div class="header"> <h1>Home Page</h1> </div> <input class = "btn" id = "btn0" type="button"
value = "BTN0" onclick = "sendRequest()"> <input class = "btn" id = "btn1" type="button" value = "BTN1"
onclick = "sendRequest1()"> <div class="sensorVal"> <p>Sensor Value: </p> <div id="sen"></div> </div>
<style> *{margin:0; padding:0;} body {background-color: #D4DCE2;} .header { width:100%; height:55px;
color: white; background-color: #04254D; padding: 0; text-align:center; } .header h1{ color:white; vertical-
align:center; font-size:42px; } .btn { margin: 0; margin-top: .5%; background-color: #FB9541; width:48%;
border: none; color: white; padding: 25px 38px; text-align: center; text-decoration: none; font-size: 16px; }
.sensorVal { margin: 0; margin-top: .5%; width:100%; height:70px; color: white; background-color:
#04254D; padding: 0; text-align:center; } .sensorVal p{ color:white; vertical-align:center; font-size:38px; }
</style> <script> function sendRequest(){ var http = new XMLHttpRequest(); http.onreadystatechange =
(()=>{ if(http.readyState === 4){ if(http.status === 200){ var res = http.responseText; } } });
http.open("GET", "0", true); http.send(); } function sendRequest1(){ var http = new XMLHttpRequest();
http.onreadystatechange = (()=>{ if(http.readyState === 4){ if(http.status === 200){ var res =
http.responseText; } } }); http.open("GET", "1", true); http.send(); } </script> </body></html>)=====";

#define EXAMPLE_ESP_WIFI_SSID      ""
#define EXAMPLE_ESP_WIFI_PASS      ""
#define EXAMPLE_MAX_STA_CONN       1

static EventGroupHandle_t s_wifi_event_group;

static const char *TAG = "wifi softAP";

static esp_err_t event_handler(void *ctx, system_event_t *event)
{
  switch(event->event_id) {
    case SYSTEM_EVENT_AP_STACONNECTED:
    ESP_LOGI(TAG, "station:"MACSTR" join, AID=%d",
     MAC2STR(event->event_info.sta_connected.mac),
     event->event_info.sta_connected.aid);
    break;
    case SYSTEM_EVENT_AP_STADISCONNECTED:
    ESP_LOGI(TAG, "station:"MACSTR"leave, AID=%d",
     MAC2STR(event->event_info.sta_disconnected.mac),
     event->event_info.sta_disconnected.aid);
    break;
    default:
    break;
  }
  return ESP_OK;
}

void wifi_init_softap()
{
  s_wifi_event_group = xEventGroupCreate();

  tcpip_adapter_init();
  ESP_ERROR_CHECK(esp_event_loop_init(event_handler, NULL));

  wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
  ESP_ERROR_CHECK(esp_wifi_init(&cfg));
  wifi_config_t wifi_config = {
    .ap = {
      .ssid = EXAMPLE_ESP_WIFI_SSID,
      .ssid_len = strlen(EXAMPLE_ESP_WIFI_SSID),
      .password = EXAMPLE_ESP_WIFI_PASS,
      .max_connection = EXAMPLE_MAX_STA_CONN,
      .authmode = WIFI_AUTH_WPA_WPA2_PSK
    },
  };
  if (strlen(EXAMPLE_ESP_WIFI_PASS) == 0) {
    wifi_config.ap.authmode = WIFI_AUTH_OPEN;
  }

  ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
  ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_AP, &wifi_config));
  ESP_ERROR_CHECK(esp_wifi_start());

  ESP_LOGI(TAG, "wifi_init_softap finished.SSID:%s password:%s", EXAMPLE_ESP_WIFI_SSID,
EXAMPLE_ESP_WIFI_PASS);
}


static void http_server_netconn_serve(struct netconn *conn)
{
  struct netbuf *inbuf;
  char *buf;
  u16_t buflen;
  err_t err;

  /* Read the data from the port, blocking if nothing yet there.
   We assume the request (the part we care about) is in one netbuf */
  err = netconn_recv(conn, &inbuf);

  if (err == ERR_OK) {
    netbuf_data(inbuf, (void**)&buf, &buflen);

    /* Is this an HTTP GET command? (only check the first 5 chars, since
    there are other formats for GET, and we're keeping it very simple )*/
    if (buflen>=5 &&
      buf[0]=='G' &&
      buf[1]=='E' &&
      buf[2]=='T' &&
      buf[3]==' ' &&
      buf[4]=='/' ) {
      printf("%c\n", buf[5]);
      /* Send the HTML header
             * subtract 1 from the size, since we dont send the \0 in the string
             * NETCONN_NOCOPY: our data is const static, so no need to copy it
       */

    //command from btn0 = '0' command from btn1 = '1'
    if(buf[5]=='0'){
    }
    if(buf[5]=='1'){
    }
    else{
      netconn_write(conn, http_html_hdr, sizeof(http_html_hdr)-1, NETCONN_NOCOPY);
      netconn_write(conn, http_index_hml, sizeof(http_index_hml)-1, NETCONN_NOCOPY);
    }
  }

}
netconn_close(conn);
```

```
netbuf_delete(inbuf);
}

static void http_server(void *pvParameters)
{
  struct netconn *conn, *newconn;
  err_t err;
  conn = netconn_new(NETCONN_TCP);
  netconn_bind(conn, NULL, 80);
  netconn_listen(conn);
  do {
   err = netconn_accept(conn, &newconn);
   if (err == ERR_OK) {
     http_server_netconn_serve(newconn);
     netconn_delete(newconn);
   }
 } while(err == ERR_OK);
 netconn_close(conn);
 netconn_delete(conn);
}

void setUpPWM()
{

}

void app_main()
{
  nvs_flash_init();
  wifi_init_softap();
  setUpPWM();
  xTaskCreate(&http_server, "http_server", 2048, NULL, 5, NULL);
}
```

Listing 1. Program template for Lab 6.

Written by Hector Mota. Modified by Dr. Erives & Mirza Elahi September 2021