



Aalto-yliopisto
Sähkötekniikan
korkeakoulu

ELEC-C5070 – Elektroniikkapaja

Loppuraportti IoT-sarastusvalo

Syksy 2021 - Ryhmä numero 16

Justus Ojala
(770084, justus.ojala@aalto.fi, puhelin 044-973 1700)

1 JOHDANTO

Työn tavoitteena oli rakentaa puhelimeen liittyvä sarastusvalo, käytännössä siis herätskello, johon liitettynä valo kirkastuu hiljalleen ennen herätystä. Taustalla tähän on työn tekijän torkutustaipumus.

Ajatus on, että puhelimen herätskello voi halutessa lähettyä ennen herätystä valolle signaalin, joka saa sen kirkastumaan toivotusti. Valon tulisi toimia samalla tavallisena kirkkaussäädetävänä valona.

2 TOTEUTETTU LAITTEISTO

Valo on rakennettu tavallisesta pöytälampusta korvaamalla sen teholelektronikka itse tehdyllä ESP32-ohjatulla. Prototyppissä teholelektronikka on itse valaisimen ulkopuolella, ja sisäinen elektronikka on ohitettu. Valaisinelementti on kytketty suoraan tehonsyöttöön, ja uusi elektronikka tulee teholähteeseen ja lampun syötön väliin.

Elektroniikan prototyppi on rakennettu protolevyn kytkeytyn ESP32-kehitysalustan ympärille. Se ottaa syötön lampun teholähteeltä ja jatkaa sen eteenpäin vaihdettavan esivastuksen ja MOSFETin läpi. ESP32 saa käyttöjännitteensä lineaariregulaattorilta, joka puodottaa jännitteen 11 V -> 5 V.

ESP32 ohjailee MOSFETia ja käyttää sitä PWM-hakkurina. ESP32 itse saa ohjauksensa joko Bluetoothin yli tai potentiometriltä ja mikrokytkimeltä, jotka ajavat virtakytkimen ja kirkkaudensäädon virkoja. Viimeisenä säädetty ohjaus on etusijalla, ja mekaaniset ohjaimet ohittavat ohjelmoidun kirkastumisen.

Laitte vastaanottaa Bluetoothin kautta yksinkertaisia signaaleja, mahdolliset komennot ovat pois, päälle <kirkkaus> ja ohjelmoitu kirkastuminen <hallintapisteet>. Laitte lukee jatkuvasti kytkimen ja potentiometrin asentoa, ja päivittää PWM-aliohjelman ohjausarvon niiden mukaisesti. Jos laite on vastaanottanut viestin, se asettaa kirkkauden sen mukaisesti tai alkaa interpoloimaan ohjauspisteiden välilä. Interpoloatio keskeytyy, jos laite saa muita ohjauksia. Laitteen puolen koodi on liitteenä.



3 MAHDOLLISTA JATKOKEHITYSTÄ

Laitteeseen olisi tarkoitus tehdä kunnollinen piirilevy, joka käyttäisi ESP32-pakettia sellaisenaan kehitysalustan sijasta. Piirilevyn olisi tarkoitus olla riittävän pieni, että sen saa asennettua alkuperäisen paikalle lampun sisään. Tällöin ei tarvitsisi tehdä uutta kotelointia, vain muokata alkuperäistä kytkimen ja potentiometrin käyttämiseksi.

Nykyisellään laitetta ohjataan lähettemällä sille tavuja raakana esimerkiksi Bluetooth-terminaalilta kautta. Tarkoitus olisi tehdä sille puhelinsovellus, joka mieluiten linkittyy vielä puhelimen ensisijaiseen herätyskelloon. Tämä sovellus on vielä työna alla, ja siltä puuttuu esimerkiksi kyky lähetää viestejä Bluetoothin yli sekä kyky ajastaa herätyksiä tai reagoida ulkoisiin. Lisäksi voisi ehkä siirtyä Bluetoothista Matter-verkkoon, jota ei ollut vielä olemassa projektin alussa.

4 MITTAUKSET JA TESTAUS

Laitteen testaamiseksi on kokeiltu kirkkauden muuttamista ja virran kytkemistä sekä etänä puhelimen BT-sarjaterminaalista että mekaanisista ohjaimista. Kirkkautta on arvioitu silmämääräisesti.

5 JOHTOPÄÄTÖKSET

Laite toimii nykyisellään kirkkaussäädettyvänen valaisimena. Etäohjausta ei käytännössä voi pitää toimivana, koska vaikka laite vastaanottaakin viestejä ja reagoi niihin oikein. Ilman kunnollista käyttöliittymää etäohjausta on kuitenkin hyvin kömpelöä käyttää. Myöskin automaatio puuttuu vielä, joten laite ei toimi vielä herätyskellona.

Toiminnalliset puutteet ovat kiinni vain puhelinsovelluksesta, laite itse on niiden osalta valmis. Laitteen omat puutteet liittyvät lähinnä viimestelyyn ja kotelointiin. Muuten laite vastaa vaatimusmäärittelyä.

LIITE 1: TYÖNJAKO JA MITÄ OPITTIIN

Kyseessä oli yksilötyö, joten työnjakoa ei ollut. Opin ESP32-ohjelmoinnista ja ESP-IDF ympäristön käytöstä, ESP32 kommunikaatiostackista, Android-ohjelmoinnista ja jonkin verran (mutten riittävästi) sen Bluetooth-stackista.

MUUT LIITTEET

```
#include <stdio.h>
#include <string.h>
#include "driver/gpio.h"
#include "sdkconfig.h"
#include "driver/ledc.h"
#include "driver/adc.h"
#include "esp_timer.h"
#include "esp_bt.h"
#include "esp_spp_api.h"
#include "esp_bt_device.h"
#include "esp_bt_main.h"
#include "esp_gap_bt_api.h"
#include "esp_log.h"
#include "nvs.h"
#include "nvs_flash.h"

#define mos_pin 14
#define button 26
#define NUM_READINGS 100

#define SPP_TAG "IOT_LAMP"

int received = 0;
uint8_t* rdata = NULL;

//Struct for control points to control the brightness of the lamp during
wakeup
typedef struct {
    float brightness;    //Brightness in range [0, 1]
    int time;           //Time from start in minutes
}cpoint;

//SPP callback function for BT communication
static void esp_spp_cb(esp_spp_cb_event_t event, esp_spp_cb_param_t *param)
{
    switch (event) {
    case ESP_SPP_INIT_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_INIT_EVT");
        esp_spp_start_srv(ESP_SPP_SEC_NONE, ESP_SPP_ROLE_SLAVE, 0,
"SPP_SERVER");
    }
```

```

        break;
    case ESP_SPP_DISCOVERY_COMP_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_DISCOVERY_COMP_EVT");
        break;
    case ESP_SPP_OPEN_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_OPEN_EVT");
        break;
    case ESP_SPP_CLOSE_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_CLOSE_EVT");
        break;
    case ESP_SPP_START_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_START_EVT");
        esp_bt_dev_set_device_name("ESP_LAMP");
        esp_bt_gap_set_scan_mode(ESP_BT_CONNECTABLE,
ESP_BT_GENERAL_DISCOVERABLE);
        break;
    case ESP_SPP_CL_INIT_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_CL_INIT_EVT");
        break;
    case ESP_SPP_DATA_IND_EVT:
        rdata = (uint8_t*) malloc(param->data_ind.len);
        memcpy(rdata, param->data_ind.data, param->data_ind.len);
        received = param->data_ind.len;
        break;
    case ESP_SPP_CONG_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_CONG_EVT");
        break;
    case ESP_SPP_WRITE_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_WRITE_EVT");
        break;
    case ESP_SPP_SRV_OPEN_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_SRV_OPEN_EVT");
        break;
    case ESP_SPP_SRV_STOP_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_SRV_STOP_EVT");
        break;
    case ESP_SPP_UNINIT_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_UNINIT_EVT");
        break;
    default:
        break;
    }
}

void esp_bt_gap_cb(esp_bt_gap_cb_event_t event, esp_bt_gap_cb_param_t *param)
{
    switch (event) {
    case ESP_BT_GAP_AUTH_CMPL_EVT:
        if (param->auth_cmpl.stat == ESP_BT_STATUS_SUCCESS) {
            ESP_LOGI(SPP_TAG, "authentication success: %s", param-
>auth_cmpl.device_name);

```

```

        esp_log_buffer_hex(SPP_TAG, param->auth_cmpl.bda,
ESP_BD_ADDR_LEN);
    } else {
        ESP_LOGE(SPP_TAG, "authentication failed, status:%d", param-
>auth_cmpl.stat);
    }
    break;
}
case ESP_BT_GAP_PIN_REQ_EVT:{
    ESP_LOGI(SPP_TAG, "ESP_BT_GAP_PIN_REQ_EVT min_16_digit:%d", param-
>pin_req.min_16_digit);
    if (param->pin_req.min_16_digit) {
        ESP_LOGI(SPP_TAG, "Input pin code: 0000 0000 0000 0000");
        esp_bt_pin_code_t pin_code = {0};
        esp_bt_gap_pin_reply(param->pin_req.bda, true, 16, pin_code);
    } else {
        ESP_LOGI(SPP_TAG, "Input pin code: 1234");
        esp_bt_pin_code_t pin_code;
        pin_code[0] = '1';
        pin_code[1] = '2';
        pin_code[2] = '3';
        pin_code[3] = '4';
        esp_bt_gap_pin_reply(param->pin_req.bda, true, 4, pin_code);
    }
    break;
}
#endif (CONFIG_BT_SSP_ENABLED == true)
case ESP_BT_GAP_CFM_REQ_EVT:
    ESP_LOGI(SPP_TAG, "ESP_BT_GAP_CFM_REQ_EVT Please compare the numeric
value: %d", (int) param->cfm_req.num_val);
    esp_bt_gap_ssp_confirm_reply(param->cfm_req.bda, true);
    break;
case ESP_BT_GAP_KEY_NOTIF_EVT:
    ESP_LOGI(SPP_TAG, "ESP_BT_GAP_KEY_NOTIF_EVT passkey:%d", (int) param-
>key_notif.passkey);
    break;
case ESP_BT_GAP_KEY_REQ_EVT:
    ESP_LOGI(SPP_TAG, "ESP_BT_GAP_KEY_REQ_EVT Please enter passkey!");
    break;
#endif
case ESP_BT_GAP_MODE_CHG_EVT:
    ESP_LOGI(SPP_TAG, "ESP_BT_GAP_MODE_CHG_EVT mode:%d", (int) param-
>mode_chg.mode);
    break;
default: {
    ESP_LOGI(SPP_TAG, "event: %d", (int) event);
    break;
}
}

```

```

    return;
}

void app_main(void){
    ESP_LOGI(SPP_TAG, "Main entry");
    //Initialize bluetooth
    esp_err_t ret = nvs_flash_init();
    ESP_LOGI(SPP_TAG, "Nvs Init comp");
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK( ret );

    ESP_ERROR_CHECK(esp_bt_controller_mem_release(ESP_BT_MODE_BLE));

    esp_bt_controller_config_t bt_cfg = BT_CONTROLLER_INIT_CONFIG_DEFAULT();
    if ((ret = esp_bt_controller_init(&bt_cfg)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s initialize controller failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bt_controller_enable(ESP_BT_MODE_CLASSIC_BT)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s enable controller failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bluetooth_init()) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s initialize bluetooth failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bluetooth_enable()) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s enable bluetooth failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bt_gap_register_callback(esp_bt_gap_cb)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s gap register failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_spp_register_callback(esp_spp_cb)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s spp register failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }
}

```

```

}

if ((ret = esp_spp_init(ESP_SPP_MODE_CB)) != ESP_OK) {
    ESP_LOGE(SPP_TAG, "%s spp init failed: %s\n", __func__,
esp_err_to_name(ret));
    return;
}

#endif

/*
 * Set default parameters for Legacy Pairing
 * Use variable pin, input pin code when pairing
 */
esp_bt_pin_type_t pin_type = ESP_BT_PIN_TYPE_VARIABLE;
esp_bt_pin_code_t pin_code;
esp_bt_gap_set_pin(pin_type, 0, pin_code);

//Initialize LED PWM timer
ledc_timer_config_t mos_timer = {
    .speed_mode      = LEDC_LOW_SPEED_MODE,
    .timer_num       = LEDC_TIMER_0,
    .duty_resolution = LEDC_TIMER_13_BIT,
    .freq_hz         = 5000,
    .clk_cfg         = LEDC_AUTO_CLK
};
ESP_ERROR_CHECK(ledc_timer_config(&mos_timer));

//Initialize LED PWM channel
ledc_channel_config_t mos_channel = {
    .speed_mode      = LEDC_LOW_SPEED_MODE,
    .channel         = LEDC_CHANNEL_0,
    .timer_sel       = LEDC_TIMER_0,
    .intr_type       = LEDC_INTR_DISABLE,
    .gpio_num        = mos_pin,
    .duty            = 0, // Set duty to 0%
    .hpoint          = 0
};
ESP_ERROR_CHECK(ledc_channel_config(&mos_channel));

//configure button pin
gpio_reset_pin(button);
gpio_set_direction(button, GPIO_MODE_INPUT);
gpio_set_pull_mode(button, GPIO_PULLDOWN_ONLY);

//Configure ADC
adc1_config_channel_atten(ADC1_CHANNEL_4, 3); //Potentiometer on pin 32

```

```

ESP_LOGI(SPP_TAG, )

//Current and previous potentiometer reading
int pots[NUM_READINGS]; uint poti = 0;
int avg_pot = 0; int prev_avg = 0;

//Button values
int button_read = 0; uint64_t bstates = 0; unsigned int state = 0;

//Current brightness and whether or not the lamp is on
float brightness = 0.0f; int on = 0;

//Values for wakeup brightness control
int bezier = 0; //Whether or not the brightness curve is a bezier curve
int autom = 0; //Whether or not the brightness is currently determined
automatically
cpoint* control_points = NULL; //The brightness curve control points

//Values for linear interpolation
float k = 0; //The slope of the current piece
float b0 = 0; //The vertical offset of the piece
float t0 = 0; //The horizontal offset of the piece
uint pt = 0; //Current point
uint points = 0; //Number of points
float t1 = 0; //The stop time of the current piece

float tStart = 0; //Start time

while(1) {
    //Take button reading with debouncing
    if(gpio_get_level(button)){
        bstates |= 1 << state;
    }else bstates &= ~(1 << state);
    state = (state + 1) % 64;

    //Check the button reading and toggle lamp/override automatic control
if high
    if(!button_read && !(~bstates)){
        button_read = 1;
        if(autom == false) on = !on;
        else autom = false;
    }else if(button_read && !bstates){
        button_read = 0;
    }

    //Check the potentiometer value with smoothing, update brightness and
override automatic
    //control if altered sufficiently
    int pot = adc1_get_raw(ADC1_CHANNEL_4);
    avg_pot += pot * 0.001 - pots[poti] * 0.001;
    pots[poti] = 1;
}

```

```

        poti = (poti + 1) % NUM_READINGS;
        if(abs(avg_pot - prev_avg) > 100){
            prev_avg = avg_pot;
            brightness = (float) avg_pot / 4095.0f;
            if(brightness < 0.0) brightness = 0.0;
            if(brightness > 1.0) brightness = 1.0;
            autom = false;
        }

        if(received){
            ESP_LOGI(SPP_TAG, "Command received (%i bytes)", (int) received);
            switch(rdata[0]){
                case 0: //Turn off
                    on = 0;
                    autom = 0;
                    ESP_LOGI(SPP_TAG, "Turning off");
                    break;
                case 1: //Turn on
                    on = 1;
                    brightness = (float) rdata[1] / 255.0;
                    autom = 0;
                    ESP_LOGI(SPP_TAG, "Turning on; brightness %f",
brightness);
                    break;
                case 2: //Set to brighten automatically (linear)
                    bezier = 0;
                    on = 1;
                    autom = 1;
                    pt = 0;
                    points = (received - 1) / 2;
                    ESP_LOGI(SPP_TAG, "Setting up linear interpolation with %i
points", (int) points);
                    if(control_points) free(control_points);
                    control_points = (cpoint*) malloc(points *
sizeof(cpoint));
                    if(!control_points){
                        ESP_LOGI(SPP_TAG, "Memory allocation for cpoints
failed.");
                        break;
                    }
                    for(uint i = 0; i < points; i++){
                        cpoint point = {(float) rdata[i*2 + 1] / 255.0,
rdata[i*2 + 2]};
                        control_points[i] = point;
                        ESP_LOGI(SPP_TAG, "Point %i: brightness %f, time %i",
i, point.brightness, point.time);
                    }
                    break;
                case 3: //Set to brighten automatically (Bezier - for now
interpreted as linear)
                    bezier = 1;
                    on = 1;
            }
        }
    }
}

```

```

        autom = 1;
        pt = 0;
        points = (received - 1) / 2;
        ESP_LOGI(SPP_TAG, "Setting up Bezier interpolation with %i
points", (int) points);
        if(control_points) free(control_points);
        control_points = (cpoint*) malloc(points *
sizeof(cpoint));
        if(!control_points){
            ESP_LOGI(SPP_TAG, "Memory allocation for cpoints
failed.");
            break;
        }
        for(uint i = 0; i < points; i++){
            cpoint point = {(float) rdata[i*2 + 1] / 255.0,
rdata[i*2 + 2]};;
            control_points[i] = point;
            ESP_LOGI(SPP_TAG, "Point %i: brightness %f, time %i",
i, point.brightness, point.time);
        }
        break;
    default: break;
}
received = 0;
free(rdata);
rdata = NULL;
}

//Automatic brightness control for wakeup
if(autom){
    float t = (float) esp_timer_get_time() / 1000000.0f - tStart;
    if(0 /*bezier*/){
        //Only linear interpolation for now
    }else{
        if(t >= t1){
            pt++;
            if(pt < points){
                t0 = t1;
                t1 = 60*control_points[pt + 1].time;
                b0 = control_points[pt].brightness;
                if(t1 > t0){
                    k = (control_points[pt + 1].brightness - b0)/(t1 -
t0);
                }else if(t1 == t0){
                    brightness = control_points[pt + 1].brightness;
                }else{
                    free(control_points);
                    control_points = NULL;
                    autom = false;
                }
            }else{
                free(control_points);
            }
        }
    }
}

```

```

        control_points = NULL;
        autom = false;
    }
} else brightness = b0 + (t - t0)*k;
}

//Set the PWM value based on the brightness and whether the lamp is on
if(on){
    ESP_ERROR_CHECK(ledc_set_duty(LEDC_LOW_SPEED_MODE, LEDC_CHANNEL_0,
brightness * 8191.0f));
    ESP_ERROR_CHECK(ledc_update_duty(LEDC_LOW_SPEED_MODE,
LEDC_CHANNEL_0));
} else{
    ESP_ERROR_CHECK(ledc_set_duty(LEDC_LOW_SPEED_MODE, LEDC_CHANNEL_0,
0));
    ESP_ERROR_CHECK(ledc_update_duty(LEDC_LOW_SPEED_MODE,
LEDC_CHANNEL_0));
}
}
}
```