Project Integration
Research Applied Computer Science

Jochem Arends
495637
Group 1

Academic Year: 2023-2024

Contents

Abbreviations

ACS	Applied Computer Science.
AIoT	Artificial Intelligence of Things.

Glossary

Bluetooth	A short-range wireless interconnection of mobile phones, computers, and other electronic devices.
WiFi	A wireless networking technology that uses radio waves to provide high-speed Internet access.

References

Espressif. (n.d.). *About Espressif.* https://www.espressif.com/en/company/about-espressif.

1 Introduction

As part of Project Integration, Applied Computer Science (ACS) students need to conduct research about the ESP32 microcontroller. Upon completion of this research, a set of assignments have to be completed. First, some introductory assignments have to be done in order to familiarize ourselves with concepts that are important for the project. At the end of the research, each student works on a final assignment that goes more in depth into a specific topic.

2 The ESP32

The ESP32 is a microcontroller developed by Espressif. The ESP32 has integrated WiFi and Bluetooth modules.

3 Espressif

Espressif Systems is a multinational semiconductor company that focusses among other things on developing wireless communication and Artificial Intelligence of Things (AIoT). (Espressif, n.d.)

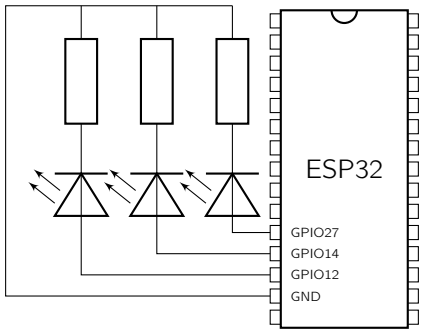
4 Lilygo

5 Assignment 1

Show that the ESP32 can turn three different LEDs on and off seperately using an internal loop with delays.

5.1 Circuit

The circuit for this assignment is quite simple. Just connect three LEDs to the ESP32, put these LEDs in series with a resistor, and connect them to a common ground.



5.2 Toolchain

For this assignment, I have used the ESP-IDF toolchain. This assignment was done using the ESP-IDF toolchain.

5.3 Software

```
#include <array>
#include <chrono>
#include <ranges>
#include <thread>

#include "driver/gpio.h"
#include "rom/gpio.h"

extern "C" void app_main() {
    const std::array<gpio_num_t, 3> led_pins{
        GPIO_NUM_12,
        GPIO_NUM_14,
        GPIO_NUM_27,
    };

    // configure the pins for output
    for (auto pin : led_pins) {
        gpio_pad_select_gpio(pin);
        gpio_reset_pin(pin);
        gpio_set_direction(pin, GPIO_MODE_OUTPUT);
    }

    // blink individual LEDs indefinitely
    for (auto pin : std::views::join(std::views::repeat(led_pins))) {
        gpio_set_level(pin, 1);
        std::this_thread::sleep_for(std::chrono::seconds{1});
        gpio_set_level(pin, 0);
    }
}
```

6 Assignment 2

6.1 Software

```
#include <iostream>
#include <map>
#include <ranges>

#include "driver/gpio.h"
#include "driver/uart.h"
#include "esp_vfs_dev.h"
#include "rom/gpio.h"

void toggle(gpio_num_t pin) {
    int level = gpio_get_level(pin) ^ 1;
    gpio_set_level(pin, level);
}

extern "C" void app_main() {
    // configure stdin to use blocking mode
    setvbuf(stdin, nullptr, _IONBF, 0);
    constexpr auto uart_num = CONFIG_ESP_CONSOLE_UART_NUM;
    uart_driver_install(static_cast<uart_port_t>(uart_num), 256, 0, 0, nullptr, 0);
    esp_vfs_dev_uart_use_driver(uart_num);

    const std::map<char, gpio_num_t> led_pins{
        {'0', GPIO_NUM_12},
        {'1', GPIO_NUM_14},
        {'2', GPIO_NUM_27},
    };

    // configure the pins for input and output
    for (auto pin : led_pins | std::views::values) {
        gpio_pad_select_gpio(pin);
        gpio_reset_pin(pin);
        gpio_set_direction(pin, GPIO_MODE_INPUT_OUTPUT);
    }

    for (char ch : std::views::istream<char>(std::cin)) {
        if (auto it = led_pins.find(ch); it != led_pins.end()) {
            toggle(it->second);
        }
    }
}
```

7 Assignment 4

7.1 Software

```
#include <chrono>
#include <cstdint>
#include <memory>
#include <thread>
#include <type_traits>

#include "driver/gpio.h"
#include "esp_event.h"
#include "esp_http_server.h"
#include "esp_log.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "freertos/task.h"
#include "nvs_flash.h"
#include "rom/gpio.h"

namespace server {
    esp_err_t post_handler(httpd_req_t* req) {
        auto level = gpio_get_level(GPIO_NUM_12) ^ 1;
        gpio_set_level(GPIO_NUM_12, level);

        const char *resp_str = "POST request received";
        httpd_resp_send(req, resp_str, strlen(resp_str));

        return ESP_OK;
    }

    struct deleter {
        void operator()(httpd_handle_t server) {
            if (server) {
                httpd_stop(server);
            }
        }
    };

    using handle = std::unique_ptr<std::remove_pointer_t<httpd_handle_t>, deleter>;

    handle make() {
        static const httpd_uri_t uri_post{
            .uri      = "/toggle",
```

```
            .method   = HTTP_POST,
            .handler  = post_handler,
            .user_ctx = nullptr,
        };

        httpd_config_t config = HTTPD_DEFAULT_CONFIG();
        httpd_handle_t server = nullptr;
        if (httpd_start(&server, &config) == ESP_OK) {
            httpd_register_uri_handler(server, &uri_post);
        }

        return handle{server};
    }
}

namespace wifi {
    namespace {
        EventGroupHandle_t event_group{};
        constexpr auto connected_bit = BIT0;
        constexpr auto fail_bit = BIT1;
    }

    void event_handler(void* arg, esp_event_base_t event_base, std::int32_t event_id, void*
        if (event_base == WIFI_EVENT) {
            if (event_id == WIFI_EVENT_STA_START) {
                esp_wifi_connect();
            }
            else if (event_id == WIFI_EVENT_STA_DISCONNECTED) {
                xEventGroupSetBits(event_group, fail_bit);
            }
        }
        else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
            ESP_LOGI("SERVER", "IP:" IPSTR, IP2STR(&event->ip_info.ip));
            xEventGroupSetBits(event_group, connected_bit);
        }
    }

    void init_sta() {
        event_group = xEventGroupCreate();

        ESP_ERROR_CHECK(esp_netif_init());
        ESP_ERROR_CHECK(esp_event_loop_create_default());
        esp_netif_create_default_wifi_sta();
```

```
        wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
        ESP_ERROR_CHECK(esp_wifi_init(&cfg));

        esp_event_handler_instance_t any_id{}, got_ip{};
        ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT, ESP_EVENT_ANY_ID, ev
        ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT, IP_EVENT_STA_GOT_IP, e

        wifi_config_t wifi_config{
            .sta{
                .ssid = "TP-LINK_3745",
                .password = "tilligte"
            },
        };

        ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
        ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config));
        ESP_ERROR_CHECK(esp_wifi_start());

        xEventGroupWaitBits(event_group, connected_bit | fail_bit, pdFALSE, pdFALSE, portMAX
    }
}

extern "C" void app_main() {
    esp_err_t ret = nvs_flash_init();
    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);

    gpio_pad_select_gpio(GPIO_NUM_12);
    gpio_reset_pin(GPIO_NUM_12);
    gpio_set_direction(GPIO_NUM_12, GPIO_MODE_INPUT_OUTPUT);

    wifi::init_sta();
    auto server = server::make();
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds{1});
    }
}
```