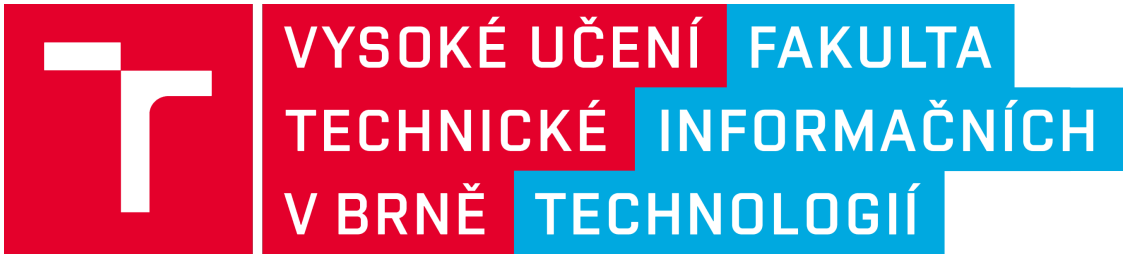


Mikroprocesorové a vestavěné systémy

ESP32: Přístupový terminál

David Chocholatý (xchoch09)

Brno, 16. prosince 2022

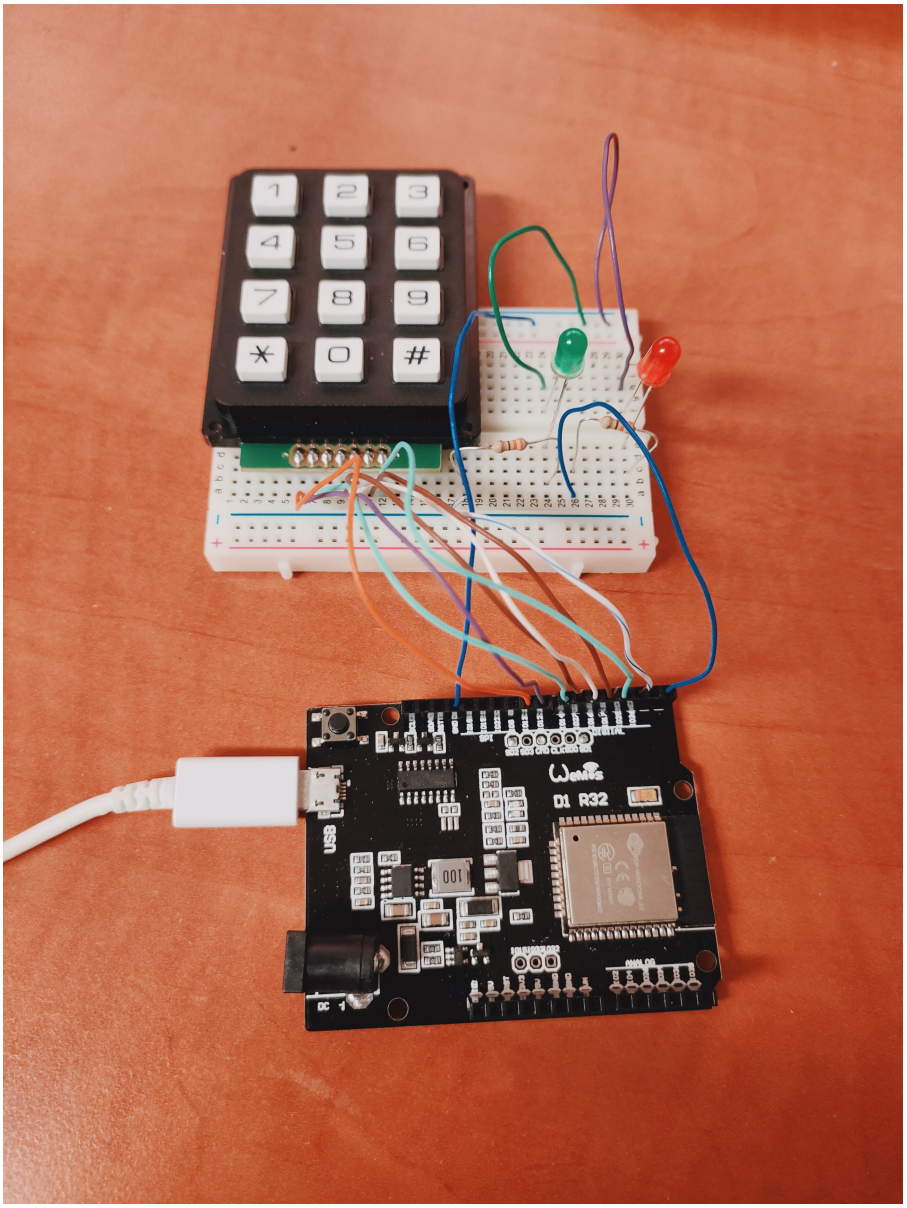
Obsah

1 Úvod

Cílem projektu bylo navrhnout a implementovat přístupový terminál za použití klávesnice a dvou led diod značících stav *otevřeno* či *zavřeno*. Implementace byla vytvořena s použitím vývojové desky ESP32 v jazyce C.

2 Zapojení hardware

Poskytnutý hardware včetně ESP, klávesnice, rezistorů a led diod byl zapojen následovně:



Obrázek 1: Zapojení hardware

Zapojení jednotlivých pinů lze nalézt přímo ve zdrojovém souboru *main.c*, kde zapojení pinů pro implementaci specifikuje následující úsek kódu

```
/*******************************/
/*        LED DIODES       */
/*******************************/

// The pin for the green led.
#define GREEN_LED_PIN GPIO_NUM_26
// The pin for the red led.
#define RED_LED_PIN GPIO_NUM_3

/*******************************/
/*       MATRIX KEYPAD     */
/*******************************/
// The pin for the first row.
#define MATRIX_KP_ROW_1 GPIO_NUM_12
// The pin for the second row.
#define MATRIX_KP_ROW_2 GPIO_NUM_25
// The pin for the third row.
#define MATRIX_KP_ROW_3 GPIO_NUM_17
// The pin for the fourth row.
#define MATRIX_KP_ROW_4 GPIO_NUM_27
// The pin for the first column.
#define MATRIX_KP_COL_1 GPIO_NUM_14
// The pin for the second column.
#define MATRIX_KP_COL_2 GPIO_NUM_13
// The pin for the third column.
#define MATRIX_KP_COL_3 GPIO_NUM_16
```

ze kterého vyplývá zapojení prvků:

- Zelená led dioda - GPIO 26
- Červená led dioda - GPIO 3
- První řádek klávesnice - GPIO 12
- Druhý řádek klávesnice - GPIO 25
- Třetí řádek klávesnice - GPIO 17
- Čtvrtý řádek klávesnice - GPIO 27
- První sloupec klávesnice - GPIO 14
- Druhý sloupec klávesnice - GPIO 13
- Třetí sloupec klávesnice - GPIO 16

3 Návod ke spuštění

Na spuštění implementace přístupového terminálu jsou zapotřebí následující požadavky:

- Aplikace *Visual Studio Code* (dále jen *VS Code*)[3].
- Nainstalovaný plugin *PlatformIO*[4] do aplikace *VS Code*.

Po instalaci všech potřebných požadavků lze přejít k připojení vývojové desky ESP32 k počítači přes USB. Dále v nastavení rozšíření *PlatformIO* lze zjistít číslo portu, na kterém je zařízení připojené, a to v nastavení *Devices*.
Po zkontrolování připojení zařízení dále je možné postupovat dvěma způsoby:

(a) Přeložení a spuštění již vytvořeného projektu.

- Nejprve bude přiložený projekt otevřen v programu *VS Code*. Dále je nutné, aby konfigurace obsahovala správný port, na kterém je připojená vývojová deska ESP32. To se provádí pomocí příkazu *upload_port* v souboru *platformio.ini* nacházejícím se v kořenové složce projektu. Ve výchozím nastavení projektu je vývojová deska očekávána na portu *COM4*.

(b) Vytvoření nového projektu, manuální konfigurace a nahrazení obsahu souboru *main.c* za vytvořenou implementaci.

- Tento způsob spuštění projektu popisuje počáteční vytvoření projektu s popisem všech konfiguračních kroků. Nejprve tedy bude vytvořen nový projekt pomocí rozšíření *PlatformIO*. Jako vývojová deska bude zvolena deska *Espressif ESP32 Dev Module* a framework *Espressif IoT Development Framework*.

  Jako další krok bude nastaven zvolený port, na kterém je zařízení připojeno také do konfiguračního souboru rozšíření *PlatformIO*. Konkrétněji do souboru *platformio.ini*, nacházejícím se v kořenové složce projektu, bude přidán řádek *upload_port = COM4* pro očekávání připojení vývojové desky na port *COM4*. V konfiguraci projektu ještě bude nastavena monitorovací rychlost 115200 baudů pomocí následujícího řádku: *monitor_speed = 115200*.

  Nyní lze přejít k poslednímu konfiguračnímu kroku, a to k vypnutí sledování procesoru ve stavu IDLE pomocí Watchdog. Pro použití tohoto projektu lze dané sledování zakázat. To bude provedeno pomocí následujícího nastavení v souboru *sdkconfig.esp32dev*:

  CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0=n
  CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1=n

  Nyní již byly všechny konfigurační kroky provedeny a lze nahradit obsah souboru *main.c* nacházejícího se ve složce *src/* za vytvořenou implementaci.

Nyní lze přejít k samotnému překladu a spuštění projektu přístupového terminálu. To lze provést pomocí úlohy *Upload and Monitor* v rozšíření *PlatformIO*.

4 Implementace

Pro vytvoření implementace byla především použita oficiální dokumentace k vývojovým deskám ESP[2]. Dále také pro seznámení s novým zařízením byly použity oficiální příklady od společnosti Espressif[1].

4.1 Konfigurace GPIO pinů

Jako první krok algoritmu po spuštění programu jsou neprve nakonfigurovány GPIO piny, a to nejprve pro obsluhu led diod a poté pro obsluhu klávesnice.

GPIO pro led diody

Piny obsluhující led diody jsou nejprve nastaveny do výchozího stavu pomocí funkce *gpio_reset_pin()*. Dále jsou piny nakonfigurovány pomocí následujícího úseku kódu:

```
// Initialize the config structure.
gpio_config_t io_conf = {
    // Set as input mode.
    .mode = GPIO_MODE_OUTPUT,
    // Disable pull-down mode.
    .pull_down_en = GPIO_PULLDOWN_DISABLE,
    // Enable pull-up mode.
    .pull_up_en = GPIO_PULLUP_DISABLE,
    // Disable interrupt.
    .intr_type = GPIO_INTR_DISABLE
};
```

GPIO piny jsou tedy nastaveny jako výstupní a oba rezistory (pull down a pull up) jsou zakázány. To stejné platí i pro obsluhu přerušení. Následně po nastavení konfigurace je přirazena k daným dvěma pinům obsluhující led diody a poté je port přiřazený k červené led diodě nastaven na hodnotu 1. Pin obsluhující zelenou led diodu je nastaven na hodnotu 0. Celý zmíněný úsek algoritmu lze nalézt ve funkci *configure_leds()*.

GPIO pro klávesnici

Dále jsou nakonfigurovány GPIO piny pro obsluhu klávesnice. Nejprve je vytvořena konfigurace totožná jak pro piny obsluhující řádky klávesnice, tak i pro piny obsluhující sloupce. Následující úsek kódu popisuje zmíněné nastavení:

```
// Initialize the config structure.
gpio_config_t io_conf = {
    // Set as input mode.
    .mode = GPIO_MODE_INPUT_OUTPUT_OD,
    // Disable pull-down mode.
    .pull_down_en = GPIO_PULLDOWN_DISABLE,
    // Enable pull-up mode.
    .pull_up_en = GPIO_PULLUP_ENABLE
};
```

Nyní konfigurace definuje nastavení módu pinu zároveň povolující vstupní i výstupní mód s povolením módu otevřeného kolektoru (open-drain mode). Dále je z rezistorů povolen pouze pull up rezistor. Pull down rezistor je opět zakázán.
Daná konfigurace je přiřazena nejprve pinům pro obsluhu sloupců klávesnice. Pro tyto piny je zakázána obsluha přerušení pomocí příkazu *io_conf.intr_type = GPIO_INTR_DISABLE*. Dále je dokončená konfigurace přiřazena všem pinům pro obsluhu sloupců a jejich výstupní hodnota je nastavena na hodnotu 0.
Piny pro obsluhu řádků klávesnice používají dříve definované nastavení, které bylo použito pro konfiguraci pinů obsluhující sloupce, ovšem nyní je obsluha přerušení povolena, a to pouze při sestupné hraně signálu pomocí příkazu *io_conf.intr_type = GPIO_INTR_NEGEDGE*. Upravená konfigurace je poté přiřazena všem pinům obsluhující řádky klávesnice a jejich výstupní hodnota je nastavena na hodnotu 1. Popsaný úsek algoritmu lze nalézt ve funkci *configure_matrix_keypad()*.

4.2 Zpracování přerušení

Po konfiguraci všech GPIO pinů obsluhujících klávesnici je v totožné funkci *configure_matrix_keypad()* nastaveno zpracování přerušení zaznamenaných pouze na řádcích klávesnice.
Nejprve je obsluha vytvořena pomocí funkce *xTaskCreate* následujícím příkazem:

```
// Start gpio task.
xTaskCreate(matrix_kp_row_isr_callback, "matrix_kp_row_isr_callback", 2048, NULL,
    5, NULL);
```

poté nainstalována služba obsluhy ISR:

```
// Install gpio isr service.
ESP_ERROR_CHECK(gpio_install_isr_service(ESP_INTR_FLAG_DEFAULT));
```

a přidán ovladač ISR pouze pro GPIO piny obsluhující řádky klávesnice:

```
// Hook isr handler for specific gpio pin.
ESP_ERROR_CHECK(gpio_isr_handler_add(MATRIX_KP_ROW_1, gpio_isr_handler, (void*)
    MATRIX_KP_ROW_1));
ESP_ERROR_CHECK(gpio_isr_handler_add(MATRIX_KP_ROW_2, gpio_isr_handler, (void*)
    MATRIX_KP_ROW_2));
ESP_ERROR_CHECK(gpio_isr_handler_add(MATRIX_KP_ROW_3, gpio_isr_handler, (void*)
    MATRIX_KP_ROW_3));
ESP_ERROR_CHECK(gpio_isr_handler_add(MATRIX_KP_ROW_4, gpio_isr_handler, (void*)
    MATRIX_KP_ROW_4));
```

Pro obsluhu přerušení je také vytvořena fronta pro zpracování událostí GPIO z ISR pomocí funkce *xQueueCreate()*.
Samotné přerušení obsluhuje funkce *gpio_isr_handler()*:

```
/*
 * The gpio isr handler for all matrix rows.
 */
static void IRAM_ATTR gpio_isr_handler(void* arg)
{
    uint32_t gpio_num = (uint32_t) arg;
    xQueueSendFromISR(gpio_evt_queue, &gpio_num, NULL);
```

```
}
```

která především vkládá položku do vytvořené fronty pomocí funkce *xQueueSendFromISR()*. Nyní jsou všechny konfigurační úkony dokončeny včetně popisu zpracování přerušení a lze přejít k části algoritmu zpracovávající detekci stisku tlačítka.

4.3 Detekce stisku tlačítka

Detekování a zpracování stisku tlačítka je vytvořeno ve funkci *matrix_kp_row_isr_callback()*, která byla přiřazena k úlohám již ve funkci *xTaskCreate()*. Úloha je implementována jako nekonečná smyčka pomocí cyklu *for(;;)*. Poté je položka přijmuta z fronty pomocí úseku kódu:

```
if(xQueueReceive(gpio_evt_queue, &io_num, portMAX_DELAY) == pdTRUE) {
...
```

Pro otestování hodnoty stisku tlačítka je implementováno čekání pro odeznění zákmitu signálu na začátku stisku tlačítka (anglicky tzv. debounce). Tuto funkcionalitu implementuje stejnojmenná funkce:

```
/*
 * The function for debouncing the matrix keypad button press.
 */
static int debounce(uint32_t io_num)
{
    // Register task to be associated with a watchdog.
    ESP_ERROR_CHECK(esp_task_wdt_add(NULL));

    int previous_level;
    int current_level;

    previous_level = gpio_get_level(io_num);

    for (int i = 0; i < DEBOUNCE_TIME_MS; i++) {
        vTaskDelay(1 / portTICK_PERIOD_MS); // Wait for 1 millisecond.

        current_level = gpio_get_level(io_num);

        if (current_level != previous_level) {
            i = 0; // Start again.
            previous_level = current_level;
        }

        // Explicitly feed the watchdog.
        ESP_ERROR_CHECK(esp_task_wdt_reset());
    }

    // Unsubscribe the task from the watchdog.
    ESP_ERROR_CHECK(esp_task_wdt_delete(NULL));

    return current_level;
}
```

Pro odeznění zákmitu lze zmínit, že je použitý o něco jiný typ algorimu, než pouhé čekání po určitou dobu pro další otestování hodnoty tlačítka, po které se předpokládá, že zákmit odezněl. Implementovaný algoritmus nejprve si zaznamená aktuální hodnotu na daném portu. Poté algoritmus přechází do cyklu, který provádí určitý počet iterací. Jelikož v každé iteraci algoritmus provádí čekání pomocí funkce *vTaskDelay()* po dobu jedné milisekundy, počet iterací určuje celkovou dobu čekání (v implementaci je použita hodnota 10 iterací). Ovšem zmíněná funkce neimplementuje pouhé čekání, ale po uplynutí jedné milisekundy opět testuje hodnotu na daném pinu. Pokud hodnota neodpovídá původní hodnotě, ještě probíhá zmíněný zákmit. Z tohoto důvodu je počet provedených iterací opět vynulován a poslední zaznamenaná hodnota na pinu je nastavena jako aktuální. Na tomto základě tedy počet iterací určuje dobu, po kterou je hodnota na pinu neměnná (čekání po dobu jedné milisekundy, které je provedeno v 10 iteracích, kdy v každé iteraci je získaná hodnota totožná s předchozí). Po odeznění zákmitu lze přejít k hlavní logice samotného zpracování stisku tlačítka.
Nejprve je pro GPIO pin, na kterém bylo detekováno přerušení, nastavena výstupní hodnota z hodnoty 1 na 0. Poté je pro všechny piny obsluhující sloupce klávesnice nastavena výstupní hodnota z hodnoty 0 na 1. Danou logiku provádí následující úsek kódu:

```
// Switch the matrix keypad gpio levels to check in which column
// the pressed pad is.
ESP_ERROR_CHECK(gpio_set_level(io_num, 0));
ESP_ERROR_CHECK(gpio_set_level(MATRIX_KP_COL_1, 1));
ESP_ERROR_CHECK(gpio_set_level(MATRIX_KP_COL_2, 1));
ESP_ERROR_CHECK(gpio_set_level(MATRIX_KP_COL_3, 1));
```

Cílem této logiky je po určení řádku klávesnice, na kterém se nachází stisknuté tlačítko, také určit sloupec, ve kterém se tlačítko nachází a tím i definování hodnoty stisknutého tlačítka dle rozložení poskytnuté klávesnice.
Určení sloupce, ve kterém se nachází stisknuté tlačítko se provádí pomocí testování vstupní hodnoty GPIO pinu, který obsluhuje daný sloupec. Například pro první sloupec je daná logika provedena následujícím úsekem kódu:

```
if (gpio_get_level(MATRIX_KP_COL_1) == 0) {
...
```

Poté již implementace využívá konstrukce *switch* pro určení GPIO pinu, který obsluhuje daný řádek, na kterém se nachází stistknuté tlačítko, a na kterém bylo vyvoláno přerušení. Poté po určení správného řádku a sloupce je určena hodnota stisknutého tlačítka.
Jako předposlední krok algoritmus používá testování, zda uživatel nedrží tlačítko stisknuté po delší dobu. Pokud ano, je tato situace ošetřena tak, že pomocí cyklu *while* se provádí čekání, dokud uživatel tlačítko nepustí do výchozí polohy. Danou logiku implementuje následující úsek kódu (například pro první sloupec klávesnice):

```
// Loop here while pressed until the user lets go.
while (gpio_get_level(MATRIX_KP_COL_1) == 0) {
    vTaskDelay(10 / portTICK_PERIOD_MS);
}
...
```

Po detekování a zpracování stisku tlačítka již mohou být změněné výstupní hodnoty navráceny na hodnoty původní. Pro GPIO pin obsluhující řádek, na kterém bylo vyvoláno přerušení, bude

výstupní hodnota navrácena zpět na hodnotu 1 a výstupní hodnota pro všechny piny obsluhující sloupce klávesnice je navrácena na hodnotu 0. Poté již funkce pouze provádí část algoritmu pro zpracování hesla.

4.4 Práce s kódem (heslem)

Definice

Pro práci s kódem (heslem) je definována struktura *password*:

```
/*******************************/
/*         PASSWORD        */
/*******************************/
typedef struct password {
    char *password_reference;
    char *password_data;
    uint32_t password_length;
    uint32_t password_position;
    bool change_password_auth_state;
    bool change_password_auth_flag;
    bool change_password_write_state;
} password_t;
```

Hodnoty jednotlivých proměnných struktury jsou nastaveny pomocí následujícího úseku kódu:

```
/*******************************/
/*         PASSWORD        */
/*******************************/

// Default configuration for the password handling structure.
#define PASSWORD_DEFAULT_CONFIG() \
{                               \
    .change_password_auth_state = false, \
    .change_password_auth_flag = false, \
    .change_password_write_state = false, \
    .password_length = 4,       \
    .password_position = 0,     \
    .password_reference = (char[]) {'1', '2', '3', '4', '\0'}, \
    .password_data = (char[]) {'x', 'x', 'x', 'x', '\0'} \
}
```

Délka kódu, neboli hesla, je pevně nastavena na délku 4 znaků. Jako výchozí heslo je nastaveno heslo *1234*. Dále je proměnná obsahující ukazatel na pole, ve kterém budou uloženy znaky zadané uživatelem na klávesnici, inicializována tak, že každý znak je ve výchozím nastavení zastoupen znakem *x*, který není obsažen na poskytnuté klávesnici (výchozí zadané heslo je tedy heslo *xxxx*). Struktura také definuje tři proměnné pro práci s heslem, a to proměnné

- *change_password_auth_state*
- *change_password_auth_flag*
- *change_password_write_state*

První proměnná (*change_password_auth_state*) slouží pro nastavení stavu systému, kdy uživatel stiskl tlačítko na klávesnici se znakem *. V takovém případě uživatel začíná sekvenci kroků, kdy chce změnit současné heslo za heslo nové. Pokud po zadání znaku * zadá správné heslo, při kontrole zadaného hesla implementace nespouští vizuální signalizaci pro zadání správného hesla, ovšem přepíná systém do druhého stavu, a to do stavu kdy je nastaven příznak identifikující správné zadání původního hesla při změně hesla (proměnná *change_password_auth_flag*). Poslední proměnná (*change_password_write_state*) slouží pro přepnutí systému do stavu, kdy uživatel chce měnit aktuální heslo za nové, správně zadal původní heslo a po zadání tohoto hesla bylo stisknuto tlačítko se znakem #, za kterým nyní bude následovat heslo nové. Právě tento stav zadávání nového hesla reprezentuje zmíněná proměnná.

Zaznamenání znaku kódu (hesla)

Po detekci a zpracování stisku tlačítka na klávesnici již lze identifikovat hodnotu stisknutého tlačítka. Pro každý znak kromě znaků * a # je znak zaznamenán do pole obsahujícího zadané heslo a je aktualizována aktuální pozice ve zmíněném poli. Danou logiku provádí úsek kódu:

```
password.password_data[password.password_position] = value;
password.password_position++;
...
```

Obsluha znaků * a # při práci s heslem je implementována specificky dle požadovaného chování a zohledňující všechny okrajové případy, které budou dále zmíněny.

Ověření správnosti kódu (hesla)

Ověření správnosti kódu provádí následující úsek kódu:

```
// The length of the entered password is the same as the length
// of the checked password.
if (password.password_position == password.password_length) {
    password.password_position = 0;

    // Check if the user tries to set a new password.
    if (password.change_password_write_state) {
        // The user entered the new password which should be set
        // as a new one. So, this state is still valid and
        // the new password will be set.

        password.change_password_write_state = false;

        // Set the new password
        strcpy(password.password_reference, password.password_data);

        ESP_LOGI(TAG, "password has been successfully changed");

        password_changed_signalization();

    } else {
        // Test if the entered password is correct.
```

```
        if (strcmp(password.password_reference, password.password_data) == 0) {
            // The entered password is correct.

            // Check if the user wants to set the new password.
            if (password.change_password_auth_state) {
                // Successful password check when changing password.
                password.change_password_auth_state = false;
                password.change_password_auth_flag = true;

                ESP_LOGI(TAG, "successful authentication when changing the
                    password");
            } else {
                // Otherwise, the user just to want to enter.
                ESP_LOGI(TAG, "passwords match");

                access_allowed_signalization();
            }

        } else {
            // The entered password is incorrect.

            // Check if the user wants to set the new password (auth state).
            if (password.change_password_auth_state) {
                // Delete the auth state.
                password.change_password_auth_state = false;
            }

            // The entered password is incorrect.
            ESP_LOGI(TAG, "passwords are different");

            access_denied_signalization();
        }
    }
}
```

Testování správnosti zadaného hesla je porovnáváno pouze v případě, že všechny požadované znaky kódu, neboli hesla, byly zadány a zároveň se systém nenachází ve stavu, kdy bylo zadáno nové heslo, které nyní má být heslem referenčním (heslo je změněno a je spuštěna příslušná vizuální signalizace). V případě zadání všech znaků kódu (hesla) je pozice v zadaném heslu nastavena opět na výchozí hodnotu 0 značící začátek pole obsahujícího zadané heslo. Poté lze přejít k porovnání zadaného hesla a požadovaného hesla pomocí funkce *strcmp()*. Nyní lze odděleně popsat následující dvě situace:

(a) Zadané heslo je správné

- Pokud zadané heslo je správné, může se systém nacházet ve dvou stavech. Pokud se systém nachází ve výchozím stavu, je spuštěna příslušná vizuální signalizace značící zadání správného kódu (hesla). Pokud ne, nachází se ve stavu změny hesla, kdy uživatel splnil první část sekvence kroků, a to zadání původního, a v tomto případě

správného hesla. Pro tuto situaci je systém přepnut do stavu, kdy je nastaven příznak značící správně zadané heslo.

(b) Zadané heslo není správné

- Pokud zadané heslo není správné, je případný stav systému značící zadávání původního hesla při změně hesla vynulován a v každém případě je spuštěna příslušná vizuální signalizace značící nevalidní kód (heslo).

4.5 Implementované okrajové případy

Implementace podporuje také následující okrajové případy (viz přiložené video).

- Zadání nesprávného původního hesla při změně kódu.
- Na pozici znaku # při změně hesla se nachází jiný znak.
- Nové heslo obsahuje znak * nebo #.
- Nové heslo je totožné s původním heslem (heslo je nahrazeno heslem totožným).
- Ve výchozím stavu znak # slouží pro vymazání již zadané části hesla, pokud již nebylo zadané celé heslo.

4.6 Vizuální signalizace

Implementace podporuje vizuální signalizaci pro tři následující situace (a jim implementované funkce):

- *access_allowed_signalization()* - signalizace správného hesla (přístup povolen)
- *access_denied_signalization()* - signalizace nesprávného hesla (přístup odmítnut) nebo nesprávné práce s klávesnicí
- *password_changed_signalization()* - signalizace úspěšné změny hesla

4.7 Watchdog

Při práci s vývojovou deskou ESP32 byla také nutná práce z tzv. *Watchdog*. Tato část je provedena nejprve v konfiguraci a poté i v samotném kódu. Při konfiguraci je vypnuto sledování procesoru ve stavu IDLE pomocí následujícího nastavení v souboru *sdkconfig.esp32dev*:

CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0=n

CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1=n

Pro záměr tohoto projektu lze tuto kontrolu vypnout a vynechat tuto v jiných případěch vhodnou funkcionalitu.

Dále je pracováno s *Watchdog* i v samotné implementaci. Nejprve je pomocí funkce *esp_task_wdt_init()* inicializován *Task Watchdog Timer (TWDT)*. Při konfiguraci je časový limit nastaven na hodnotu 2 sekundy a je zakázáno vyvolání tzv. paniky (*panic handler*). Poté je na začátku každé funkce (mimo funkcí *gpio_isr_handler()* a *matrix_kp_row_isr_callback()*) registrována úloha k *Watchdog* pomocí příkazu *esp_task_wdt_add(NULL)*, případně v průběhu funkce je *Watchdog* explicitně periodicky informován o validním průběhu práce systému pomocí funkce *esp_task_wdt_reset()*, aby bylo zabráněno vypršení časového limitu. Na konci funkce je každá úloha odregistrována od *Watchdog* pomocí příkazu *esp_task_wdt_delete(NULL)*.

4.8 Ověření validity řešení

Po dokončení implementace bylo řešení důkladě testováno jak na jednotlivé případy užití, které zmiňuje oficiální zadání, tak i na případy užití definované autorem, které zadání rozšiřují. Během práce se systémem se systém nedostal do nežádoucího stavu či neprováděl jiné, než požadované úkony. Na základě důkladného otestování a demostrace funkčnosti řešení pomocí videa lze tímto podpořit a konstatovat validitu implementovaného řešení.

5 Autoevaluace

Na základě vypracování projektu lze provést autoevaluaci pro následující kritéria hodnocení.

5.1 Přístup (E)

Přístup k projektu lze hodnotit pozitivně. Projekt byl řešen s dostatečným časovým předstihem. Nejprve autor projektu si pečlivě nastudoval podstatné informace pro práci s vývojovou deskou ESP32 a vyzkoušel pár testovacích a ukázkových oficiálních příkladu pro otestování funkčnosti zařízení a seznámení se s vývojem na dané platformě. Autor při implementaci pracoval i nad rámec oficiálního zadání, implementoval podporu i pro okrajové případy a navrhnul vhodné způsoby signalizace pro jednotlivé situace, které mohou při práci se systémem nastat.

5.2 Funkčnost (F)

Implementace projektu splňuje veškerou požadovanou funkcionalitu, kterou, jak již bylo řečeno, rozšiřuje pro ošetření a podporu okrajových případů. Při práci se systémem během testování nenastala situace, kdy by se systém dostal do nepožadovaného stavu či pracoval nesprávně. Na základě testování lze tímto podpořit validitu implementovaného řešení.

5.3 Kvalita (Q)

Autor projektu implementoval různé způsoby signalizace pomocí dvou led diod (červené a zelené) nad rámec oficiálního zadání. Tímto byla zvýšení především uživatelská přívětivost systému. Dále byl projekt implementován s důrazem na kvalitu a čistotu zdrojového kódu a veškerá funkcionalita je popsána i přiloženými odpovídajícími komentáři. Již struktura dokumentace napovídá o preciznosti dekompozice projektu na jednotlivé podproblémy a dílčí úkony, které jsou implementovaným algoritmem prováděny.

5.4 Prezentace (P)

Po dokončení implementace projektu bylo autorem vytvořeno prezentační komentované video, kde ukazuje i se slovním popisem práci se systémem a jednotlivé implementované a podporované situace.

5.5 Dokumentace (D)

Tato dokumentace obsahuje jak úvod do problematiky, tak také definování cíle projektu. Dále je popsáno zapojení poskytnutého hardware a konfigurace projektu nutná pro správné spuštění implementace. Dokumentace je důkladně rozdělena do kapitol dle jednotlivých podproblémů, kde v každé kapitole je problém následně důkladně vysvětlen a popsán. Vytvořená ukázka také obsahuje části kódu ke slovnímu popisu pro lepší názornost řešení. Dokumentace také zmiňuje výčet implementovaných okrajových řešeních nad rámec oficiálního zadání projektu a popisuje ověření validity implementovaného řešení. Na závěr autor projektu provádí finální autoevaluaci.

Použitá literatura

[1] Espressif: Examples. [online], rev. prosinec 2021, [vid. 2022-12-14]. Dostupné z: https://github.com/espressif/esp-idf/tree/edd815af2e7d541b25ff3a74c59a7fe0612df42d/examples

[2] Espressif: *ESP-IDF Programming Guide*. rev. 2022, [vid. 2022-11-30]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html

[3] Microsoft: Visual Studio Code. [online], rev. prosinec 2021, [vid. 2022-12-14]. Dostupné z: https://code.visualstudio.com/

[4] PlatformIO Labs: Platform IO. [online], rev. prosinec 2021, [vid. 2022-12-14]. Dostupné z: https://platformio.org/