IMP - Přístupový terminál

Autor: Jakub Kratochvíl (xkrato67)

Úvod

Cílem je vytvořit terminál s maticovou klávesnicí připojený na mikrokontrolér ESP32. Mikrokontrolér bude mít dále připojené dvě LED diody, červenou a zelenou, kterými bude indikovat zda přístupový kód byl správný nebo ne. Tento kód bude mít vnitřně uložený v paměti i po vypnutí napájení a bude možné ho nastavit.

Příprava

Součástky

V přípravě na projektu jsem si nejprve připravil všechny potřebné součástky, kterými jsou:

- ESP32 x 1
- Nepájové pole x 1
- Maticová klávesnice x 1
- Červená LED x 1
- Zelená LED x 1
- Rezistor x 2
- Micro USB x 1
- Odizolované drátky x 12

Po přípravě součástek, především odizolování drátku, jsem začal s prototypem projektu, abych vyzkoušel funkcionalitu všech součástek a popřípadě nějaké nefunkční včas vyměnil.

Framework a IDE

Jako vývojový framework jsem si vybral ESP-IDF. Po zkušenostech s oběmi rozšířeními pro VSCode pro vývoj s frameworkem IDF, kterými jsou Platoformio a Espressif IDF, jsem zvolil rozšíření Platformio.

Pokud bych měl porovnat obě vývojová prostředí, přijde mi, že mají velice podobné ovládání, s tím rozdílem, že ESP-IDF má mnohem více pokročilých konfiguračních možností a Platformio je více uživatelsky přívětivé. Pro tento projekt jsem kvůli jednoduchosti vybral rozšíření Platformio.

Implementace

Pro zapnutí debugovacích výpisů lze v souboru main.c nastavit makro DEBUG na 1, případně 0 pro vypnutí.

Hlavní proces

Základem implementace je proces terminal_task, který je spuštěň ve funkci app_main a obsahuje veškerou funkcionalitu projektu.

NVS

Na začátku procesu terminal_task načítám handle k NVS a hledám hodnotu s klíčem “password”, která obsahuje přístupový kód terminálu, jestliže tento klíč s hodnotou neexistuje, tak ho vytvořím a zapíši výchozí heslo “1234”.

Inicializace pinů

Dále inicializuji všechny GPIO piny pro LED diody, maticovou klávesnici a také definuji lokální proměnné potřebné pro běh programu.

GPIO porty pro klávesnici nastavuji následovně:

1. Pro všechny řádky klávesnice:

a. Nastavím pin jako vstupní (GPIO_MODE_INPUT).
b. Použiji pull-up rezistory (GPIO_PULLUP_ONLY), což zajistí, že pin bude mít logickou hodnotu 1, když není stisknuta žádná klávesa.

2. Pro všechny sloupce klávesnice:

a. Nastavím pin jako výstupní (GPIO_MODE_OUTPUT).
b. Nastavím hodnotu na 1, což bude výchozí stav pro sloupce.

Hlavní programová smyčka

Následuje hlavní nekonečná programová smyčka, která odchytává všechny stisky maticové klávesnice a pomocí dvou vnořených for cyklů je zpracovává. Důvod pro zvolení smyčky pro obsluhu klávesnice, namísto přerušení je ten, že obsluha klávesnice je jediná činnost, kterou mikrokontrolér vykonává a nebude tedy bránit v běhu jiným procesům.

Smyčka vypadá následovně:

```
while (1) {
        for (int col = 0; col < COLS; col++) {
                gpio_set_level(keyboard.col_pins[col], 0);
                for (int row = 0; row < ROWS; row++) {
                                c = button_handler(row, col, keyboard.row_pins[row]);
                                check_char(c, pw, buf, my_handle, &state);
                }
                gpio_set_level(keyboard.col_pins[col], 1);
        }
}
```

Kontrola zda stisk tlačítka proběhl je ve funkci button_handler, která při úspěšném odchycení stisku tlačítka vrací znak, který byl na klávsenici stisknut.

S výstupem této funkce dále pracuje funkce check_char, která zpracuje daný znak a nebo nevykoná nic a program se vrátí zpět do iterace.

Upřednostil jsem zde přehlednost před výkonností, jelikož velká zanořenost kódu už začínala být nepřehledná.

Tudíž nyní má mikrokontrolér režii navíc, kvůli vytváření a rušení rámců zásobníku pro dvě volané funkce, které většinu času nic neobsluhují, ale jelikož vím, že mikrokontrolér zvládá obsluhu bez problému a práce se zasobníkem je velice optimalizovaná, přišla mi přehlednost důležitější.

Kontrola stisku tlačítka

Pro kontrolu tlačítka jsem zvolil standartní přístup, který implementuje kontrolu pomocí dvojitého zpoždění (delay). Kód pro kontrolu vypadá následovně:

```
bool button_pressed(int row_gpio) {
    if (gpio_get_level(row_gpio) == 0) {
        vTaskDelay(30 / portTICK_PERIOD_MS);
        if (gpio_get_level(row_gpio) == 0) {
            vTaskDelay(120 / portTICK_PERIOD_MS);
            return true;
        }
    }
    return false;
}
```

Po prvním stisku tlačítka, program čeká 30 milisekund a následně zkontroluje stisk znova, tímto se odstraní několikanásobné sepnutí vytvořené mechanickými odskoky kontaktu. Po otestování mi hodnota 30 milisekund přišla jako nejlepší, jelikož při nižší hodnotě se občas zaznamenalo více stisknutí a při vyšší hodnotě se zase stalo, že se některé stisky nezaznamenaly.

Druhé zpoždění nastavuje, za jak dlouhou dobu je možné přijmout další stisk tlačítka. Toto se nejvíce projevuje pokud uživatel například tlačítko drží a chceme nějak zamezit velkému množství registrovaných stisknutí.

Funkčnost

Stavy a změna hesla

Program má 3 stavy pro řízení běhu programu:

1. Normal
2. Set1
3. Set2

Stav normal je standardní stav programu, ve kterém čte vstup od uživatele a kontroluje, zda byl zadán správný přístupový kód. Další dva stavy, tedy Set1 a Set2, slouží pro změnu hesla.

Program je ve stavu Set1, dokud při změně hesla uživatel nezadal staré heslo. Pokud zadané heslo bylo správné, program pokračuje do stavu Set2, ve kterém se zadává nové heslo, pokud správné nebylo, program se vrací do stavu normal.

Nastavování hesla je v následujicím formátu: #<staré_heslo>*<nové_heslo>#. Důvodem pro ukončení hesla znakem # je ten, že nové heslo může mít jinou délku než to staré a bez tohoto znaku by nebylo možné rozlišit, kde končí staré heslo a začíná nové.

Heslo má vnitřně definovanou maximální délku, která je 16 znaků. Program kontroluje délku vstupního bufferu a při velikosti 15 znaků nedovoluje další zapsání a vyzívá uživatele k smazání bufferu tlačitkem (*).

Tlačítka a LED

Popis funkcionality tlačítek ve stavu normal:

- **(0-9)** - tlačítka pro zadání hesla
- **(*)** - vymaže vstupní buffer
- **(#)** - potvrdí vstup od uživatele a porovná zadané heslo z bufferu s vnitřně uloženým heslem v paměti

Při zapojení ESP začne svítit červená LED dioda, která tímto značí, že dosud nebylo zadáno přistupové heslo. Po zadání správného hesla se červená LED dioda zhasne a na 2 sekundy začne svítit zelená LED dioda která značí, že bylo zadáno správné heslo. Při zadání špatného hesla se červená LED dioda zhasne na 0,5 sekundy a následně začne znovu svítit, aby nějak indikovala, že heslo bylo špatné. Další LED indikace je při správném nastvení nového hesla, kdy střídavě probliknou obě LED diody.

Ukázka je ve videu níže.

Video s ukázkou

https://nextcloud.fit.vutbr.cz/s/i2dLHNWJjz5iitg