Střední průmyslová škola elektrotechnická Havířov	**PRG**	**Třída: 4B** **Skupina:** 1

Pomodoro stopky

Den: 16.01.2025

Jméno učitele: Božena Ralbovská

Jméno: Vojtěch Lisztwan

Známka: snad jedna

1. Zadání

Realizujte program pomocí RTOS RTX, který bude mít následující vlastnosti:

- Pomodoro je metoda, které funguje s použitím „minutky“ kde se počítají dva časové úseky: práce (typicky 20 minut) a odpočinek (typický 5 minut); po 4 úsecích práce je dlouhý odpočinek (typicky 10 minut). Pro testování použijte 2m - 5s - 10s
- Program tedy bude realizovat pomodoro a zobrazovat čas (odpočet) na prvním řádku ve formátu “#c MM:SS p ”, kde c je aktuální pomodoro (cyklus),p je druh úseku(práce...).
- Zmáčknutím tlačítka uživatelského tlačítka (BTN) se zahájí a ukončí běh programu (reset).
- Pomocí klávesy ‘*’ dojde k přeskočení na další cyklus
- Pomocí klávesnice ve stejný okamžik vložte číslo (zobrazené na druhém řádku), které bude označovat počet udělaných úkolů. (úkoly se pro daný úsek sčítají)
- Aktuální úsek v jednom cyklu zobrazte na LED diodách.
- Počet úseků, jejich trvání a úkoly provedené v úseku i celkový počet udělaných úkolů pošlete sériovým kanálem s ASCII kódováním do PC.
- Požadavky na program:
- Program bude využívat min. 3 tasky.
- V RTX nastavte: strategii Round-robin

2. Teoretický rozbor - analýza

-schémata periferii, vysvětlení RTOS

Struktura kódu

Knihovny a makra:

Zahrnuté knihovny poskytují funkcionalitu pro práci s hardwarem STM32, jako jsou LED, tlačítka, LCD a UART.

Definované časové hodnoty `WORK`, `PAUSE`, `LONGPAUSE` (v sekundách) slouží k řízení jednotlivých úzků cyklu Pomodoro.

Globální proměnné:

Proměnné jako `sec`, `cyklus`, `counterPrace`, `ukoly` slouží ke sledování aktuálního stavu časovače a logiky aplikace.

Semafor `semaphore1` synchronizuje přístup k LCD displeji mezi různými úkoly RTOSu.

Funkce:

led_set a led_set_all: Ovládají stav LED diod.

vypocet: Počítá zbývající čas do konce aktuálního úzku cyklu.

RTOS tasky:

job1: Hlavní logika Pomodoro cyklu (práce, pauza, dlouhá pauza). Zajišťuje aktualizaci displeje a stavu LED.

job2: Zpracovává vstupy z klávesnice pro zadání počtu úkolů během pracovního úzku.

job3: Odesílá shrnutí cyklů a počtu úkolů na UART kanál.

job4: Implementuje jednoduchý časovač, který inkrementuje globální časovou proměnnou `sec`.

job5: Resetuje aplikaci při stisknutí tlačítka USER.

job6: Implementuje přímou změnu cyklu přes klávesnici (stisk `*`).

Hlavní funkce:

setup: Inicializuje hardware, globální proměnné a vytváří úkoly.

main: Spouští RTOS inicializaci.

Analýza funkcionality

1. Pomodoro logika

Pomodoro cyklus se skládá ze tří fáz:

Práce (`WORK` - 120 s)

Krátká pauza (`PAUSE` - 30 s)

Dlouhá pauza (`LONGPAUSE` - 50 s)

Po čtyřech cyklech práce následuje dlouhá pauza a cyklus se resetuje.

Tato logika je řízena v úkolu `job1`, který sleduje aktuální stav a změny mezi fázemi.

2. Synchronizace přes semafory

Synchronizace pomocí semaforu `semaphore1` zabraňuje konfliktům při zapisování na LCD displej z různých úkolů.

3. Rozhraní a vstupy

Klávesnice je použita pro zadávání úkolů a manuální změnu cyklů.

Tlačítko USER slouží k resetu aplikace.

UART poskytuje diagnostický výstup a shrnutí cyklů.

4. LED diody

LED indikuje aktuální fázi cyklu.

LED_IN_0 – práce

LED_IN_1 – krátká pauza

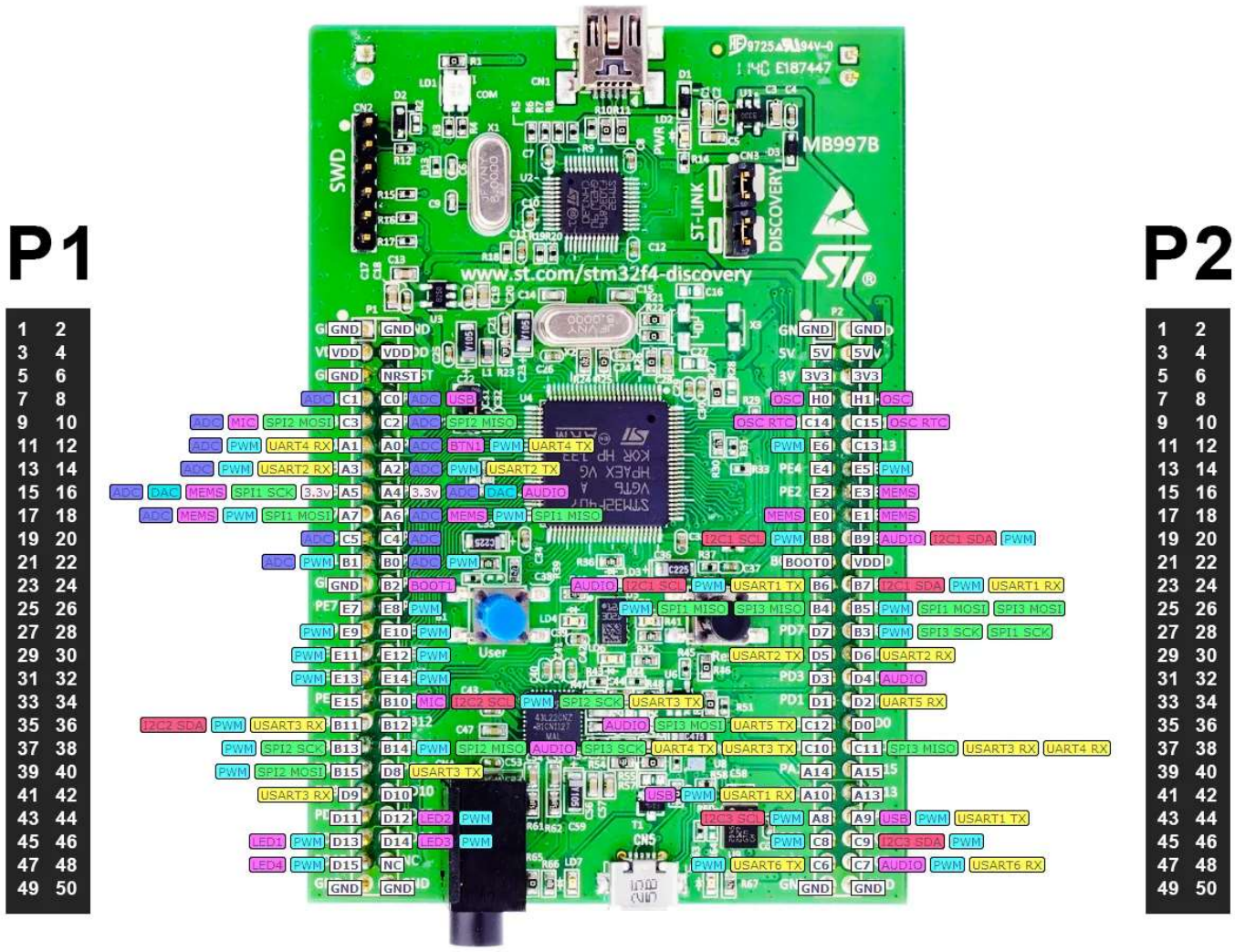
LED_IN_2 – dlouhá pauza

5. Displej

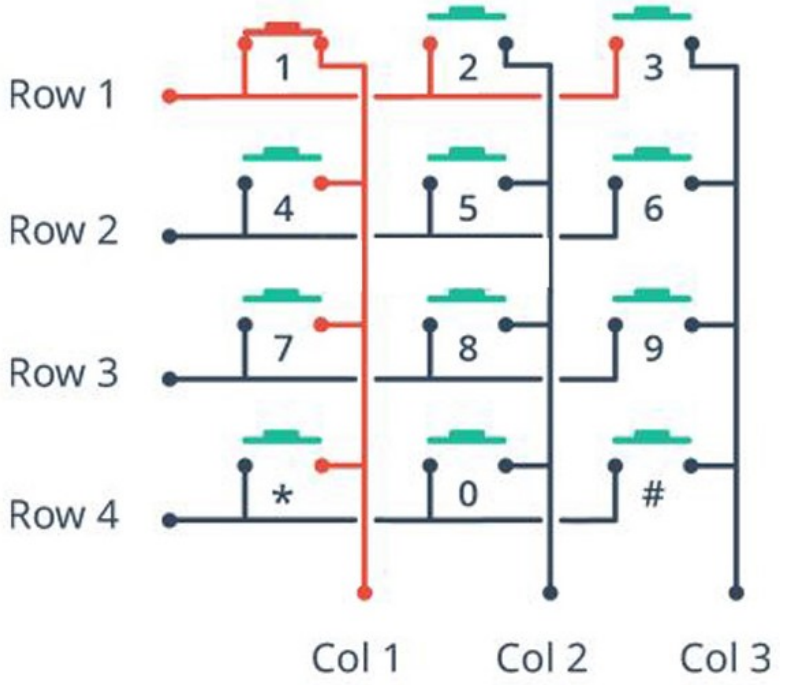
LCD zobrazuje čas zbývající do konce fáze a typ fáze ("prace", "pause", "big pause").

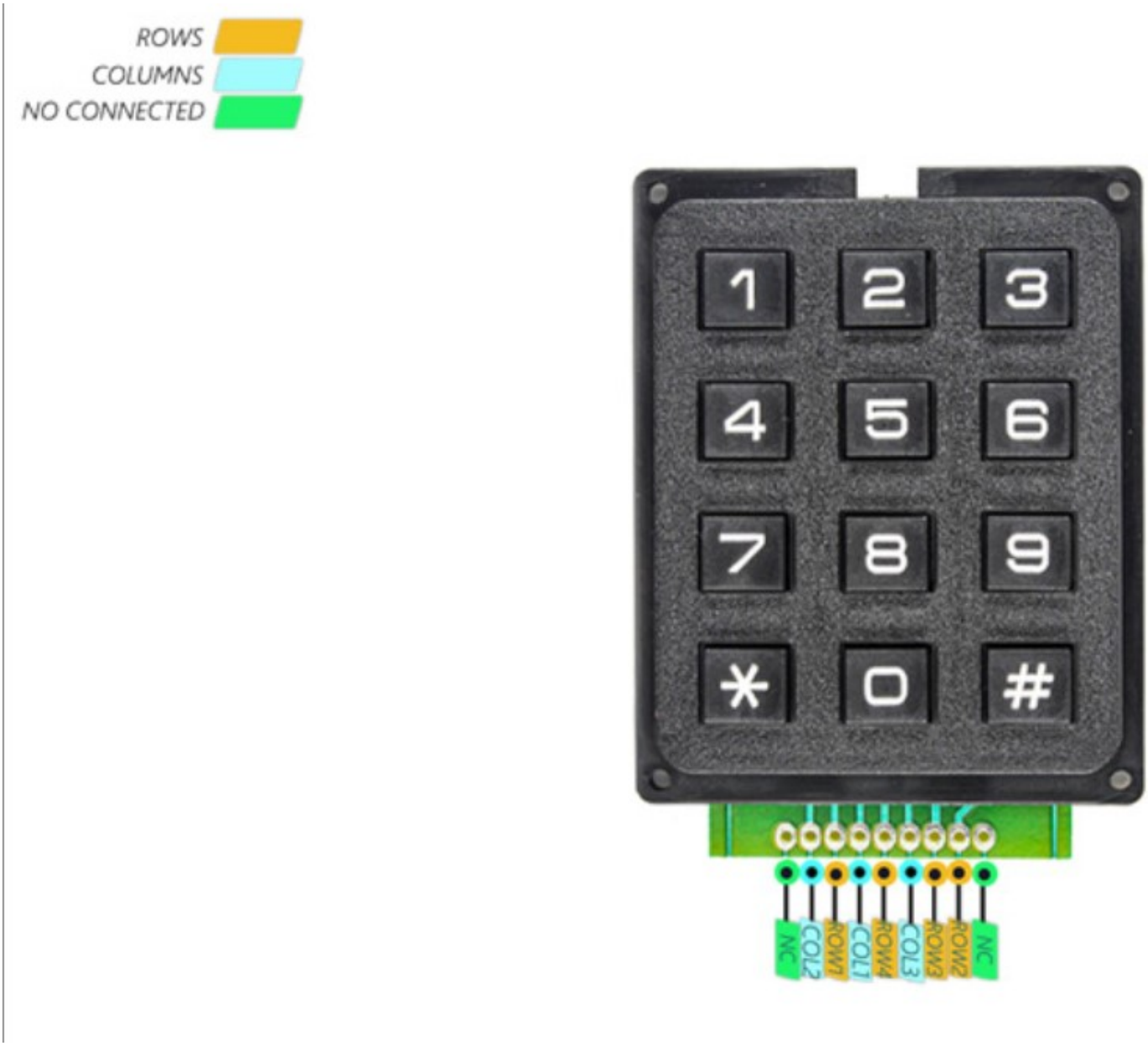
Hardware

STM32F407


P1
P2
www.st.com/stm32f4-discovery
MB997B
SWD
ST-LINK
DISCOVERY
User

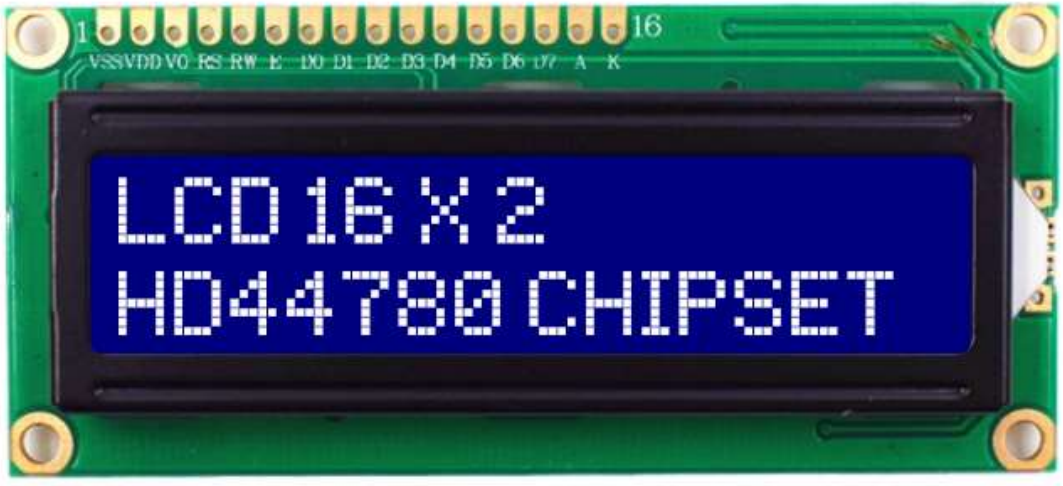
Klávesnice

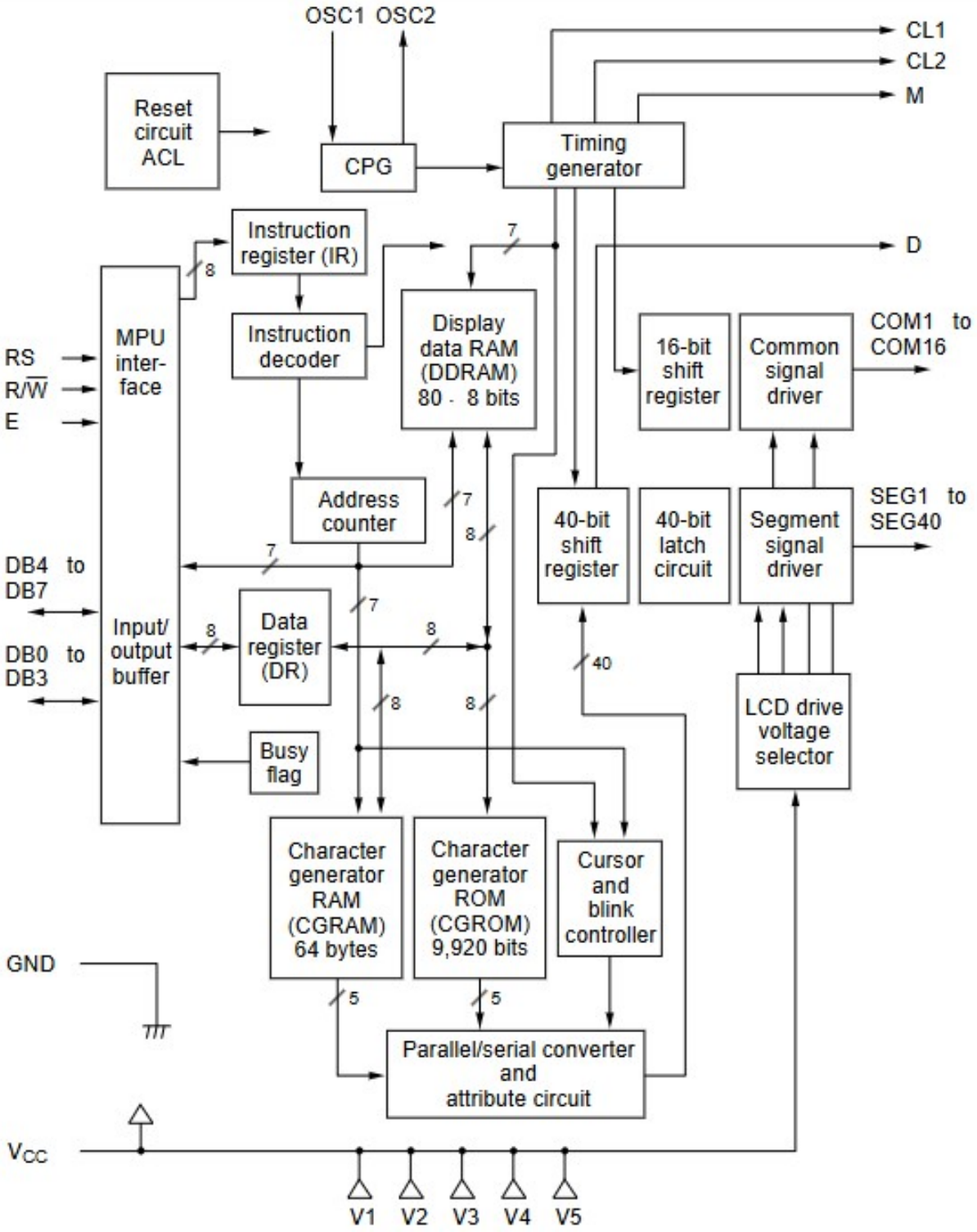

Row 1
Row 2
Row 3
Row 4
1
2
3
4
5
6
7
8
9
*
0
#
Col 1
Col 2
Col 3



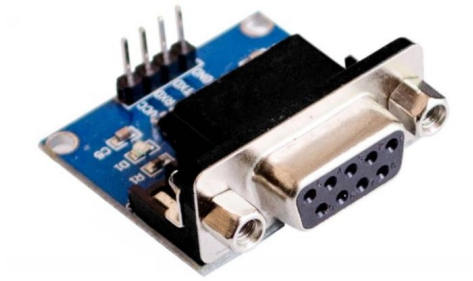
V tomto projektu je použita tato klávesnice, ale verze 4x4

LCD 16x4



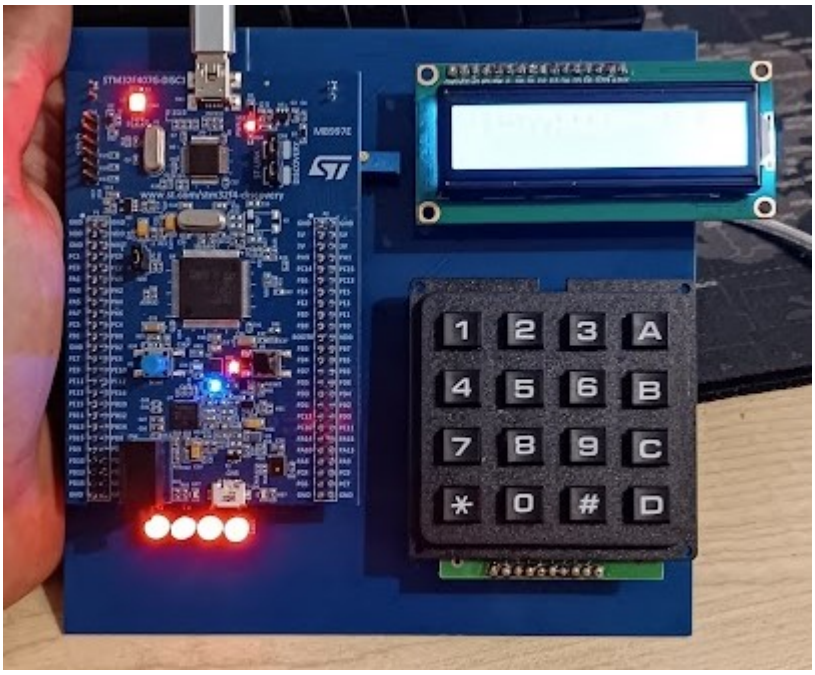


RSR232 na TTL převodník



DESKA-vlastní výroby

Deska je zjednodušení pro zapojení.



3. Postup - popis vlastních funkcí

led_set(pin_t pin, int value)

Nastavuje stav konkrétní LED diody podle zadaného pinu. Funkce rozlišuje mezi interními a externími LED diodami. Pro externí LED je logika inverzní (0 zapnuto, 1 vypnuto).

led_set_all(enum pin leds[], int value)

Nastavuje stav všech LED diod ze zadaného pole `leds` na hodnotu `value`. Používá funkci `led_set()` k ovládání jednotlivých LED diod.

opak(pin_t ledka)

Přepíná aktuální stav konkrétní LED diody zadané parametrem `ledka`.

vypocet()

Vypočítává zbývající čas v aktuálním cyklu. Výsledek převádí na minuty a sekundy a formátuje do textového řetězce uloženého v proměnné `buff`.

__task void job1()

Hlavní funkce časovače. Sleduje průběh pracovního cyklu, krátké pauzy a dlouhé pauzy. Podle aktuálního cyklu nastavuje zbývající čas, typ cyklu (`type`) a ovládá LED diody. Zobrazuje informace na LCD.

__task void job2()

Zpracovává vstupy z klávesnice. Umožňuje zadávat počet úkolů pro jednotlivé pracovní bloky. Hodnoty jsou ukládány do pole `ukoly`.

__task void job3()

Po dokončení všech čtyř pracovních bloků odesílá výsledky (součet úkolů a rozdělení podle bloků) přes sériový kanál UART. Udržuje pořadí pracovních cyklů pomocí proměnné `counterCtveric`.

__task void job4()

Implementuje jednoduchý časovač, který každou sekundu inkrementuje hodnotu proměnné `sec`. Slouží jako základ pro měření času.

__task void job5()

Sleduje stav uživatelského tlačítka (`USER_BUTTON`). Po stisku tlačítka resetuje všechny parametry, jako je čas, cyklus a pole úkolů. Navíc bliká všemi LED diodami jako indikace resetu.

__task void job6()

Zpracovává speciální vstupy z klávesnice, konkrétně znak `*`. Umožňuje ručně přepnout mezi cykly (práce, krátká pauza, dlouhá pauza).

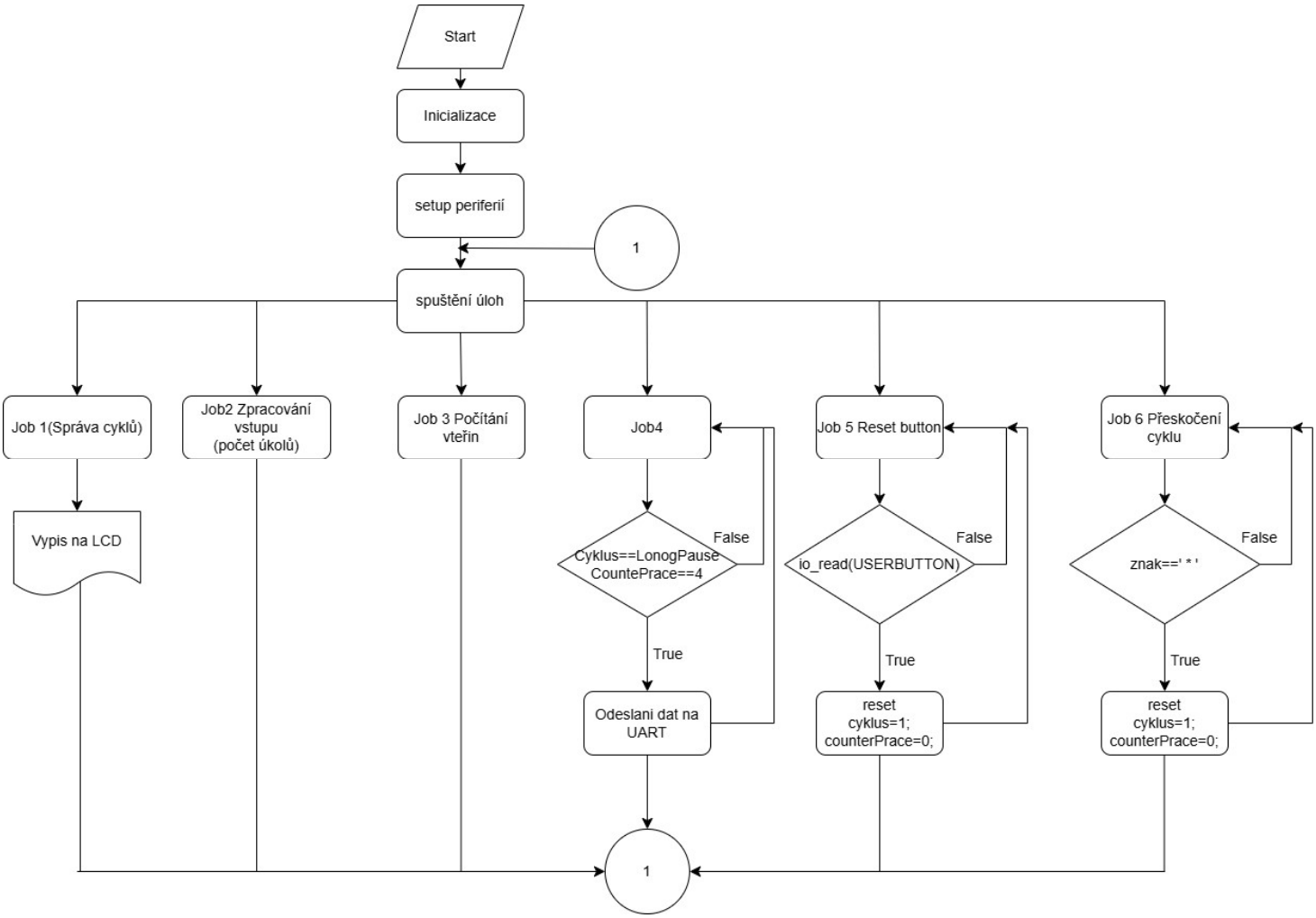
__task void setup()

Inicializuje všechny potřebné periferie a zdroje, včetně LCD, tlačítek, UART, klávesnice a LED diod. Nastavuje semafor `semaphore1` pro synchronizaci. Vytváří jednotlivé úlohy (tasks) a následně ukončuje úlohu `setup`.

int main()

Spouští operační systém a inicializační úlohu `setup`. Po spuštění systému jsou zajištěny všechny potřebné funkce prostřednictvím vytvořených úloh.

4. Vývojový diagram



5. program + blokový komentář

```
/**
 * @file    pomodoroStopky.c
 * @author <Vojtech Lisztwan>
 *
 * Tento soubor implementuje Pomodoro timer pomoci mikrokontroleru STM32.
 * Obsahuje funkce pro pracovni intervaly, kratke a dlouhe prestavky,
 * pocitani ukolu a interakci s uzivatelem pomoci tlacitek a LCD displeje.
 */
```

```c
#include "stm32_kit.h"
#include "stm32_kit/lcd.h"
#include "string.h"
#include "stdio.h"
#include "stm32_kit/button.h"
#include <rtl.h>
#include <stdbool.h>
#include "stm32_kit/uart.h"
#include "stm32_kit/keypad.h"
#include "stm32_kit/led.h"
#include "stm32_kit/pin.h"

// Definice konstant pro casovani Pomodoro
#define WORK 120       // Pracovni interval v sekundach
#define PAUSE 30       // Kratka prestavka v sekundach
#define LONGPAUSE 50   // Dlouha prestavka v sekundach

// Mapovani klavesnice pro vstup od uzivatele
static uint8_t KBD_MAP[KEYPAD_ROWS][KEYPAD_COLS] = {
    '1', '2', '3', 'A',
    '4', '5', '6', 'B',
    '7', '8', '9', 'C',
    '*', '0', '#', 'D'
};

// Definice pinu pro LED diody
enum pin ledky[] = {
    LED_IN_0,
    LED_IN_1,
    LED_IN_2,
    LED_IN_3,
```

```
    LED_EX_0,
    LED_EX_1,
    LED_EX_2,
    LED_EX_3,
    P_INVALID
};

// Funkce pro nastaveni stavu jedne LED diody
void led_set(pin_t pin, int value) {
    int on = 1;
    if (io_port(pin) == io_port(LED_EX_0)) {
        on = 0; // Reverze logiky pro externi LED
    }
    io_set(pin, on ? value : !value);
}

// Funkce pro nastaveni stavu vsech LED v poli
void led_set_all(enum pin leds[], int value) {
    for (int i = 0; leds[i] != P_INVALID; i++) {
        led_set(leds[i], value);
    }
}

// Funkce pro prepinani stavu LED diody
void opak(pin_t ledka) {
    led_set(ledka, !io_get(ledka));
}

// ID jednotlivych uzlu ukolu
OS_TID id_task1, id_task2, id_task3, id_task4, id_task5, id_task6;
```

```
// Semaphore pro pristup k LCD
OS_SEM semaphore1;

// Promenne pro sledovani stavu Pomodoro cyklu a ukolu
int counterCtveric = 1; // Pocet dokonceni Pomodoro cyklu
int counterPrace = 1;  // Pocet dokonceni pracovnich intervalu v ramci cyklu
int cyklus = 1;        // Typ aktualniho cyklu (1: prace, 2: kratka prestavka, 3: dlouha prestavka)
int sec = 0;           // Globalni casove pocitadlo v sekundach
int prev = 1;          // Predchozi typ cyklu
char stav[10];         // Popis stavu
int sekundy_start, minuty = 0; // Startovaci cas a zbyvajici minuty
int sekundy_end = WORK;        // Konecny cas aktualniho intervalu
int sekundy_do_konce;          // Zbyvajici sekundy do konce intervalu
char buff[17];                 // Buffer pro zobrazeni na LCD
char type[10] = "prace";       // Typ aktualniho cyklu
int ukoly[4] = {0, 0, 0, 0};   // Pole pro ulozeni poctu ukolu

// Funkce pro vypocet zbyvajiciho casu a formatovani textu
void vypocet() {
    sekundy_do_konce = sekundy_end - sec;
    minuty = (sekundy_do_konce - (sekundy_do_konce % 60)) / 60;
    sekundy_do_konce = sekundy_do_konce % 60;
    snprintf(buff, 17, "%d %02d:%02d ", counterPrace, minuty, sekundy_do_konce);
}

/**
 * Ukol pro spravu pracovniho intervalu a prestavek.
 */
__task void job1() {
    for (;;) {
        if (prev != cyklus) { // Kontrola zmeny cyklu
```

```
if (cyklus == 1) counterPrace++; // Zvyseni poctu pracovnich intervalu
if (counterPrace > 4) { // Reset po dokonceni cyklu
    for (int i = 0; i < 4; i++) {
        ukoly[i] = 0;
    }
    counterPrace = 1;
}
prev = cyklus;
sekundy_start = sec;
minuty = 0;

// Vycisteni LCD pro novy cyklus
os_sem_wait(semaphore1, 0xffff);
LCD_set(LCD_CLR);
os_sem_send(semaphore1);

// Nastaveni delky intervalu a typu cyklu
switch (cyklus) {
    case 1:
        sekundy_end = sec + WORK;
        strcpy(type, " prace");
        break;
    case 2:
        sekundy_end = sec + PAUSE;
        strcpy(type, " pause");
        break;
    case 3:
        sekundy_end = sec + LONGPAUSE;
        strcpy(type, " big pause");
        break;
}
```

```
            // Aktualizace LED indikatoru
            led_set_all(ledky, 0);
            led_set(ledky[cyklus - 1], 1);
        }

        delay_ms(300);
        vypocet(); // Vypocet zbyvajiciho casu
        strcat(buff, type);

        // Zobrazeni zbyvajiciho casu a typu cyklu na LCD
        os_sem_wait(&semaphore1, 0xffff);
        LCD_set(LCD_LINE1);
        LCD_print(buff);
        LCD_set(LCD_LINE2);
        os_sem_send(&semaphore1);

        // Prechod na dalsi cyklus po skonceni intervalu
        if (sekundy_end < sec) {
            if (cyklus == 1) {
                if (counterPrace == 4) cyklus = 3;
                else cyklus = 2;
            } else {
                cyklus = 1;
            }
        }
    }
}

/**
 * Ukol pro zpracovani vstupu od uzivatele pomoci klavesnice.
 */
```

```
__task void job2() {
    for (;;) {
        uint16_t znak;
        char text[8];
        znak = KBD_read();
        if (znak) {
            if (znak >= '0' && znak <= '9') {
                ukoly[counterPrace - 1] = ukoly[counterPrace - 1] * 10 + (znak - '0');
            }
        }
        os_sem_wait(semaphore1, 0xffff);
        snprintf(text, 8, " %d ", ukoly[counterPrace - 1]);
        LCD_set(LCD_LINE2);
        LCD_print("pocet ukolu:");
        LCD_print(text);
        os_sem_send(semaphore1);
        delay_ms(30);
    }
}


/**
 * Ukol pro odeslani shrnutych dat na UART kanal.
 */
__task void job3() {
    for (;;) {
        int sum = 0;
        char text[50];
        if (counterPrace == 4 && cyklus == 3) {
            for (int i = 0; i < 4; i++) {
                sum += ukoly[i];
            }
```

```
            snprintf(text, 50, "%d: %d 1-%d;2-%d;3-%d;4-%d\n\r", counterCtveric, sum,
ukoly[0], ukoly[1], ukoly[2], ukoly[3]);
            os_sem_wait(semaphore1, 0xffff);
            UART_write(text, 50);
            os_sem_send(semaphore1);
            counterCtveric++;
            while (cyklus == 3) {
                // Cekani na zmenu cyklu
            }
        }
    }
}

/**
 * Ukol pro inkrementaci globalniho casoveho pocitadla.
 */
__task void job4() {
    for (;;) {
        sec++;
        delay_ms(800);
    }
}

/**
 * Ukol pro ovladani tlacitka uzivatele pro reset timeru.
 */
__task void job5() {
    for (;;) {
        if (io_read(USER_BUTTON)) {
            os_sem_wait(&semaphore1, 0xffff);
            sec = 0;
            cyklus = 1;
```

```
            counterPrace = 0;
            prev = 0;
            for (int i = 0; i < 4; i++) {
                ukoly[i] = 0;
            }
            counterCtveric = 1;
            led_set_all(ledky, 1);
            delay_ms(250);
            led_set_all(ledky, 0);
            delay_ms(250);
            os_sem_send(&semaphore1);
        }
    }
}

/**
 * Ukol pro zpracovani specialniho vstupu z klavesnice pro manualni zmenu cyklu.
 */
__task void job6() {
    for (;;) {
        uint8_t znak = KBD_read();
        if (znak == '*') {
            if (cyklus == 1) {
                if (counterPrace == 4) cyklus = 3;
                else cyklus = 2;
            } else {
                cyklus = 1;
            }
        }
        delay_ms(200);
    }
}
```

```
/**
 * Inicializacni ukol pro nastaveni periferii a vytvoreni ostatnich ukolu.
 */
__task void setup() {
    LCD_setup();
    BTN_setup();
    UART_setup();
    KBD_setup();
    LED_setup();

    for (int i = 0; i < 4; i++) {
        ukoly[i] = 0;
    }

    os_sem_init(&semaphore1, 0);
    os_sem_send(&semaphore1);

    LCD_set(LCD_CUR_NO_BLINK);
    LCD_set(LCD_CUR_OFF);

    id_task1 = os_tsk_create(job1, 0);
    id_task2 = os_tsk_create(job2, 0);
    id_task3 = os_tsk_create(job3, 0);
    id_task4 = os_tsk_create(job4, 0);
    id_task5 = os_tsk_create(job5, 0);
    id_task6 = os_tsk_create(job6, 0);
    os_tsk_delete_self();
}

/**
```

```
 * Hlavni funkce pro spusteni RTOS a inicializacniho ukolu.
 */
int main() {
    os_sys_init(setup);
}
```

6. Zhodnocení

Projekt Pomodoro byl úspěšně realizován na platformě STM32 za použití RTOS (Real-Time Operating System). Cílem bylo vytvořit systém pro řízení pracovních intervalů, krátkých a dlouhých přestávek, s využitím uživatelského rozhraní zahrnujícího LCD displej, klávesnici, tlačítka a LED diody.

Díky desce vlastní výroby bylo jednodužší periferie propojit dohromady a nemusel jsem mít strach z povytaženého vodiče.

Klíčové vlastnosti projektu

Modularita:
Implementace jednotlivých úkolů (tasks) byla rozdělena do modulů, které umožňují efektivní řízení pracovních cyklů (práce, pauza, dlouhá pauza), uživatelský vstup, časování a zobrazování na LCD displeji.

Interaktivita:
Projekt zahrnuje vstupy od uživatele prostřednictvím klávesnice a tlačítek. Uživatel může zadávat počet úkolů, resetovat časovač nebo manuálně přepínat cykly.

Použití periferií:
Byly využity různé periférie dostupné na STM32, jako je LCD displej pro vizualizaci času a stavu, LED diody pro indikaci aktuálního cyklu, klávesnice pro zadávání údajů a UART pro komunikaci se sériovým terminálem.

Real-Time Management:
Díky použití RTOS je zajištěn plynulý chod jednotlivých úkolů, včetně inkrementace času, zpracování vstupu a aktualizace displeje. Semafory byly využity k synchronizaci přístupu ke sdíleným prostředkům, jako je LCD displej.

Silné stránky projektu

- **Efektivní využití RTOS**: Rozdělení funkcionality do úkolů zlepšuje čitelnost a správu kódu.
- **Robustní správa času**: Systém správně počítá zbývající čas a umožňuje uživateli sledovat průběh cyklů.
- **Přehledný kód**: Projekt je dobře komentovaný (nyní bez diakritiky), což usnadňuje orientaci v kódu i jeho případné rozšíření.

Možnosti vylepšení

1. **Optimalizace času delay**: Některé části kódu (např. delay_ms) mohou být optimalizovány, aby systém reagoval rychleji na uživatelské vstupy.

2. **Přesné časování:** Časování (počítání sekund) je v tomto případě docela dost nepřesné. Pro větší přesnost by bylo vhodnější použít jiný interní časovač(třeba TIM4) a trochu jiný způsob počítání. Třeba použití Handleru pro přičtení sekundy do čítače.
3. **Energetická efektivita**: Při delší nečinnosti by bylo vhodné implementovat úsporný režim pro minimalizaci spotřeby energie.
4. **Rozšíření funkcí**: Například možnost uložit historii pracovních intervalů nebo grafické rozhraní na LCD.
5. **Lepší validace vstupů**: Přidat funkce pro kontrolu a validaci dat zadávaných uživatelem.

Závěr

Projekt splnil všechny stanovené cíle. Pomodoro timer je plně funkční a demonstruje efektivní použití RTOS pro řízení více úkolů. Systém je rozšiřitelný a poskytuje dobrý základ pro budoucí úpravy nebo implementace dalších funkcí.