ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra elektrických pohonů a trakce

Bakalářská práce

Běžící text pomocí kitu NUCLEO STM32F303RE

Adam Štěpják
Studijní program: Elektrotechnika, energetika a management
Specializace: Aplikovaná elektrotechnika

Květen 2022
Vedoucí práce: doc. Ing. Jan Bauer, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Štěpják** Jméno: **Adam** Osobní číslo: **491825**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra elektrických pohonů a trakce**

Studijní program: **Elektrotechnika, energetika a management**

Specializace: **Aplikovaná elektrotechnika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Běžící text pomocí kitu NUCLEO STM32F303RE

Název bakalářské práce anglicky:

Runnig Text on NUCLEO STM32F303RE

Pokyny pro vypracování:

1) Seznamte se s kitem ST NUCLEO
2) Navrhněte algorimus pro kódování znaků pomocí SN74HC594
3) Naprogramujte funkci "běžícího textu" do kitu NUCLEO

Seznam doporučené literatury:

[1] datasheet ST NUCLEO
[2] Poupa J. Programovací jayzk C

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Jan Bauer, Ph.D. katedra elektrických pohonů a trakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

doc. Ing. Jan Bauer, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání | Podpis studenta

Poděkování / Prohlášení

Chtěl bych především poděkovat docentu Bauerovi za skvělý profesionální a zároveň lidský přístup při vedení této bakalářské práce. Také bych rád poděkoval svým rodičům, přítelkyni a přátelům, kteří mi byli neodmyslitelnou podporou po celou dobu mého studia.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 20. 5. 2022

.....

Abstrakt / Abstract

Úkolem této bakalářské práce je vytvořit program, který pomocí jednoho sloupce šestnácti LED diod vytvoří čitelný text. Pás LED diod je připevněn na rotoru, který je poháněn pomocí malého stejnosměrného motoru. Program bude nahrán do mikrokontroléru ST Nucleo. V první části této práce bude popsán programovací jazyk, vývojové prostředí, ve kterém je kód napsán a následně budou popsány použité periferie. V druhé části bude blíže rozebrán samotný kód.

Klíčová slova: ST Nucleo, programování, kód, běžící text, F302R8

The task of this bachelor thesis is to create a program that uses a single column of sixteen LEDs to produce readable text. The strip of LEDs is mounted on a rotor which is driven by a small DC motor. The program will be uploaded to the ST Nucleo microcontroller. The first part of this thesis will describe the programming language, the development environment in which the code is written and then the peripherals used will be described. In the second part, the code itself will be discussed in more detail.

Keywords: ST Nucleo, programming, code, running text, F302R8

Title translation: Runnig Text on NUCLEO STM32F303RE

/ Obsah

/ Obrázky

Kapitola 1
Úvod

Mikroprocesory prošly značným vývojem od doby vynálezu prvního mikroprocesoru Intel 4004 roku 1971. Vývoj mikroprocesorů se řídí Mooerovým zákonem, který říká, že počet tranzistorů na čipu se každých 18 měsíců zdvojnásobí [1]. Díky objevování nových materiálů a výrobních procesů bylo možno značně zmenšit tranzistory. Ty se původně vyráběly deseti mikrometrovým výrobním procesem [2]. V dnešní době je možno tranzistory vyrábět pomocí čtyř nanometrové technologie [3]. I díky tomuto pokroku je dnes možné mikroprocesory najít téměř v jakémkoliv elektrickém zařízení. Nalézt je můžeme samozřejmě v zařízeních, jako jsou počítače nebo mobily, ale také v zařízeních, ve kterých to nemusí být na první pohled znát, například v pračkách nebo myčkách.

Cílem této práce je seznámení se s kitem STM32 Nucleo a naprogramování běžícího textu. Program budu psát v programovacím jazyce C ve vývojovém prostředí STM32CubeIDE. K desce je připojen elektromotor, pomocí něhož je otáčena řada LED diod. Elektromotor je napájen ze snižovacího měniče napětí, který je řízen pulzně šířkovou modulací. STM32 Nucleo generuje signál (logické jednička a nuly), který následně posílá do shift registrů. Shift registry SN74HC594 pak rozsvěcují nebo zhasínají, dle příchozího signálu, LED diody. Rotující LED diody pak vytváří čitelný text.

V této práci stručně shrnu základní informace o programovacím jazyku C a jak vypadá vývojové prostředí STM32CubeIDE. Také se zmíním o tom, jak vypadá a z čeho se skládá zařízení použité k vypracování této práce. Blíže se budu věnovat periferiím, které byly použity k fungování samotného kódu. V neposlední řadě budu psát o základních parametrech kitu STM32 Nucleo F302R8 a o parametrech shift registrů SN74HC594. Nakonec se budu podrobně věnovat algoritmu a kódu, který jsem pro tuto práci napsal.

Předmětem této práce nebyl návrh samotného zařízení, ať už se jedná například o desky plošných spojů nebo zapojení jednotlivých periferií. Celé zařízení mi bylo zapůjčeno docentem Bauerem, což mi umožnilo mnohem snadnější a rychlejší ladění celého programu.

Kapitola 2
Úvod do programovacího jazyka C

2.1 Funkce

Funkce jsou základem programovacího jazyka C. Pokud chceme vytvořit spustitelný program musí obsahovat alespoň jednu funkci. Tato funkce se zpravidla nazývá main. Pro správnou deklaraci funkce se používá následující zápis. Na prvním místě je návratový typ funkce. Návratovým typem může být číslo, znak nebo řetězec znaků. Za návratovým typem následuje název funkce. Za názvem funkce se v kulatých závorkách nachází parametry funkce. Po deklaraci funkce následuje tělo funkce ve složených závorkách. Deklarace funkce může vypadat například takto:

- void mojeFunkce(parametrFunkce)

Za zmínku stojí, že funkci nemusíme předávat žádný parametr. Pokud nechceme předat funkci žádný parametr necháme kulaté závorky prázdné. Zvláštním návratovým typem funkce je *void*, který po vykonání funkce nevrací žádnou hodnotu.

Funkce bývají psány ve zdrojových souborech s příponou .c. Ke každému takovému souboru pak zpravidla odpovídá hlavičkový soubor s příponou .h. Hlavičkové soubory se používají k dopředné deklaraci funkcí a jejich proměnných. Hlavičkové soubory je nutné vytvořit, pokud chceme určitou funkci použít ve více zdrojových souborech. Následně je nutné přidat hlavičkový soubor do zdrojového souboru pomocí příkazu include [4], za kterým následuje název hlavičkového souboru i s jeho příponou.

2.2 Hodnoty datových typů

Hodnoty datových typů jsou označovány jako literály nebo také konstanty. Jazyk C používá 6 typů literálů, těmi jsou:

- Celočíselné
- Racionální
- Znakové
- Řetězcové
- Výčtové

Celočíselné literály mohou být reprezentovány datovými typy *char, short, int, long* a *long long*. Pokud chceme, aby tyto datové typy byly neznaménkové, před datový typ je nutné napsat *unsigned*. V této práci nejčastěji používám *char*, protože zpravidla zabírá paměť pouhých 8 bitů, tedy jeden bajt, nebo *short*, který zpravidla zabírá paměť 16 bitů, což jsou 2 bajty.

Racionální literály, též označovány jako reálné se používají pro aproximaci reálných čísel. Bývají implementovány a nejčastěji se řídí normou IEEE-754-1985. Tato čísla jsou složeny ze znaménka, mantisy a exponentu. Typy racionálních literálů jsou *double, float* a *long double*.

Znakové konstanty často zapisujeme pomocí datového typu *char*. Znak zapisujeme do jednoduchých apostrofů například:

- char písmeno = ’A’

V paměti je ale tento znak uložen jako číselná hodnota. Všechny znaky a jejich číselné hodnoty lze dohledat v ASCII tabulkách.

Řetězcové konstanty označujeme jako *string* a zapisují se do dvojitých uvozovek “ “. Ve své podstatě se jedná o pole znakových konstant typu *char*. Zápis vypadá takto:

- char JMENO[] = “TEXT“

Toto pole je automaticky zakončeno znakem \0. Toho využívám později při zjišťování, zda již byly zobrazeny všechny požadované znaky.

Výčtové konstanty se vytváří pomocí datového typu *enum*. Syntaxe pak je *enum* název výčtu a do složených závorek píšeme jednotlivé proměnné a jejich hodnoty. Může tedy vypadat takto:

- enum barvy{
- modrá = 1,
- zelená = 3,
- };

2.3 Řídicí struktury

Řídicí struktury jsou kusy kódu, které určují způsob provedení jednotlivých příkazů. Základními druhy řídicích struktur jsou posloupnosti, větvení a cykly.

Posloupnost znamená, že se postupně provedou příkazy tak, jak jsou v řadě za sebou zapsány. Posloupnost je provedena složeným příkazem nebo blokem příkazů. Ty jsou ohraničeny složenými závorkami.

Větvení umožňuje provést pouze tu část kódu, která splní zadané vstupní podmínky. V jazyce C a mnoha dalších se pro větvení používají příkazy *if*, případně *if else* a příkaz *switch*.

Cyklem se rozumí opakované provádění příkazů nebo bloku kódu tak dlouho, dokud jsou splněny zadané vstupní podmínky. Cykly jsou reprezentovány příkazy *while*, *do while* a *for* [5] .

Kapitola 3
STM32CubeIDE

STM32CubeIDE je integrované vývojové prostředí od firmy STMicroelectronics. Je určena uživatelům, kteří vyvíjejí software pro mikroprocesory a mikrokontroléry STM32 výše zmíněné společnosti. Kód je možno psát v jazyce C nebo C++.

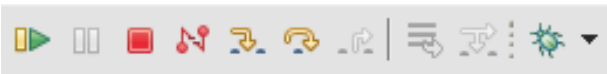
3.1 Založení nového projektu

Pro založení nového projektu můžeme použít klávesovou zkratku alt+shift+N nebo v záložce File najít řádek s názvem New a kliknout na Project. Poté se otevře dialogové okno, ve kterém je třeba zvolit typ nově zakládaného projektu. V tomto případě jsem zvolil STM32 Project. Nyní se otevře okno, kde je potřeba najít typ mikroprocesoru, který budeme programovat. V mém případě je to STM32F302R8. V dalším kroku pojmenujeme nově zakládaný projekt a zvolíme jazyk, ve kterém budeme programovat. Já zvolil jazyk C. Po tomto kroku lze kliknout na tlačítko Finish a STM32CubeIDE automaticky vygeneruje kostru programu speciálně pro zvolený typ mikroprocesoru.

3.2 Ovládání prostředí

V levé části vývojového prostředí se nachází Project Explorer, který obsahuje všechny složky potřebné pro běh programu. Zde jsou nejdůležitější 2 složky. Složka s názvem Inc, do které je nutno vložit hlavičkové soubory použitých funkcí. Druhá složka je Src, ve které se nacházejí zdrojové soubory použitých funkcí.

Při ladění programu jsem používal hlavně následující ikony. První ikona zleva spouští běh programu. Druhou ikonou můžeme běh programu pozastavit. Červený čtvereček ukončí běh programu. Následující tři žluté ikony šipek umožňují krokovat program. Zelená ikona brouka přeloží program a následně jej nahraje do přípravku.

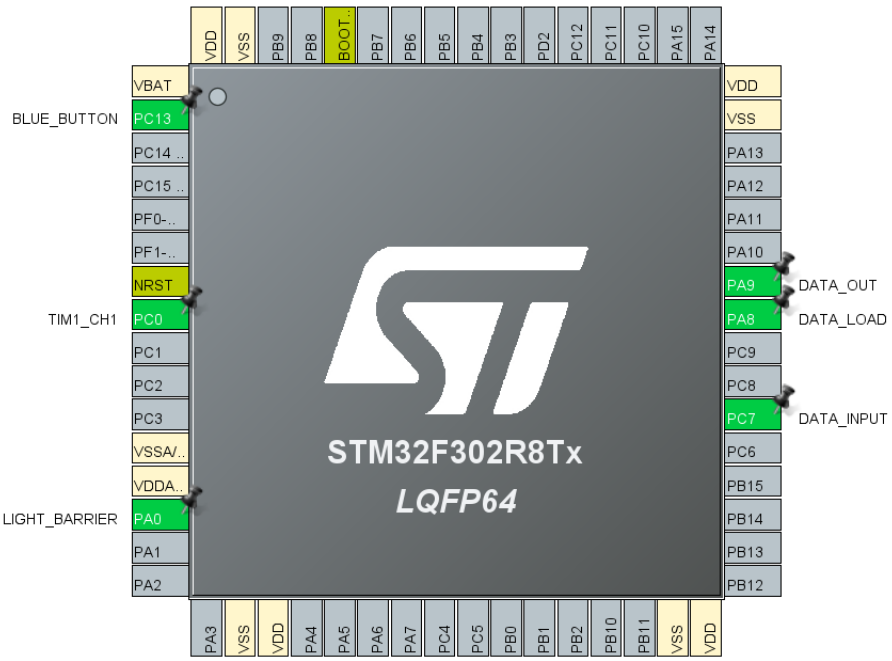


Obrázek 3.1. Ikony hlavních ovládacích funkcí

Mezi velice užitečné ladící funkce patří Console. Ta poskytuje informace o všech varováních či chybách, které se vyskytly při kompilaci nebo běhu programu. Dále bych zmínil možnost zobrazit aktuální stav registrů, pomocí čehož je možno, blíže se podívat, co se v přípravku odehrává, a to zejména pokud něco nefunguje tak, jak má. Tuto funkci je možno zobrazit při běhu programu po kliknutí na záložku Registers. Podobně se lze podívat na hodnoty proměnných, které byly uživatelem vytvořeny. Ty lze nalézt v záložce Variables také při běhu programu.

3.3 STM32CubeMX

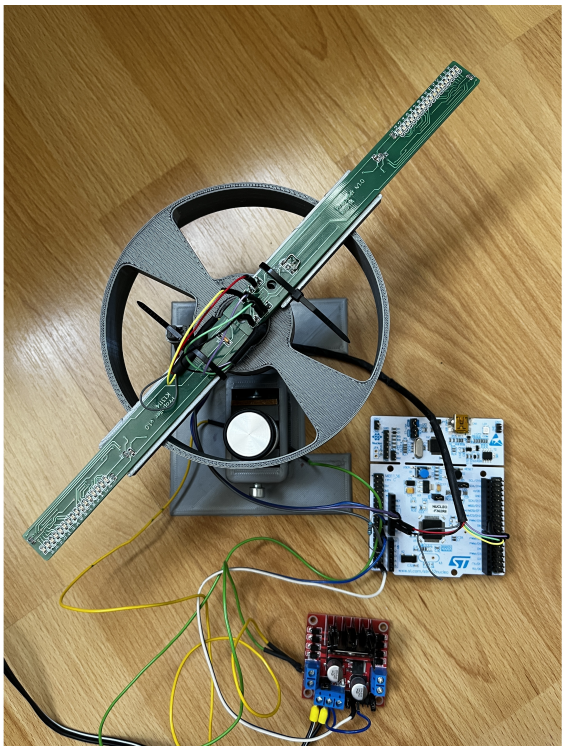
Vývojové prostředí STM32CubeIDE obsahuje také grafický nástroj pro snadnější konfiguraci vstupů a výstupů mikroprocesoru potažmo, celého mikrokontroléru. Při zakládání projektu je automaticky generován soubor s příponou .ioc, ve kterém je možné konfigurovat například již zmíněné vstupy a výstupy nebo čítače pomocí grafického rozhraní. Vstupy a výstupy lze libovolně pojmenovat a později v kódu pomocí těchto názvů k nim i přistupovat. Po konfiguraci libovolného pinu, program automaticky vygeneruje část kódu a nahraje jej tam, kde je to potřeba. Nejčastěji se části kódu přidávají do souboru main.c nebo do souboru stm32f3xx_it.c, kde probíhá obsluha programu, pokud program dostane signál od vnějšího přerušení.



Obrázek 3.2. Grafické prostředí STM32CubeMX

Kapitola 4
Složení zařízení

Zařízení se skládá z několika částí. Hlavní částí je kit ST Nucleo, který je programován a jsou k němu připojeny všechny ostatní periferie. Dále zařízení obsahuje 2 shift registry na jejichž výstupu jsou připojeny LED diody. Zařízení obsahuje také elektromotor, snižovací měnič napětí a světelnou závoru. Nosná konstrukce je vyrobena pomocí 3D tisku. V následujících odstavcích budou stručně popsány výše zmíněné periferie. Jelikož kit ST Nucleo a shift registry jsou hlavním tématem práce, budu se jimi zabývat v samostatných kapitolách.



Obrázek 4.1. Fotografie celé sestavy

4.1 Stejnosměrný elektromotor

Rotor zařízení je poháněn stejnosměrným elektromotorem. Na hřídeli elektromotoru je pogumovaný kotouč, který je umístěn na vnitřní straně rotoru zařízení tak, aby se ho dotýkal. Tím pádem při otáčení hřídele elektromotoru se otáčí pogumovaný kotouč a ten následně otáčí rotorem zařízení.

Pro stejnosměrný motor platí tyto rovnice:

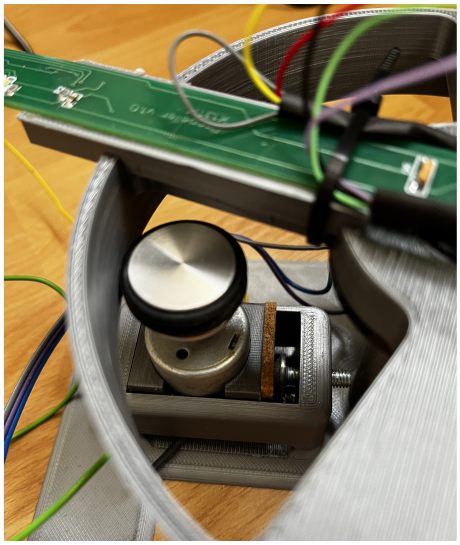
$$U = R_a I + U_i \tag{1}$$

kde U je napájecí napětí, R_a je kotevní odpor, I je napájecí proud a U_i je indukované napětí elektromotoru.

$$U_i = k\phi\Omega \tag{2}$$

kde U_i je indukované napětí elektromotoru, k je konstanta motoru, ϕ je magnetický tok buzení.

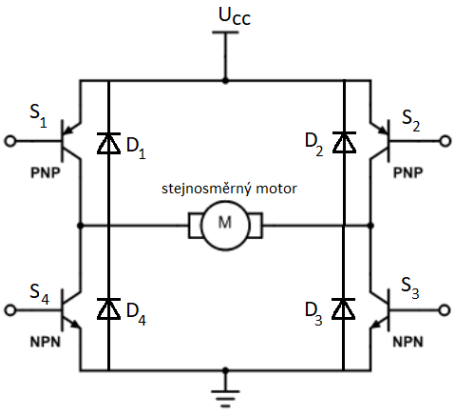
Z první rovnice je možné vidět, že napájecí napětí se rozloží na úbytek napětí na kotvě elektromotoru a na indukované napětí. Z druhé rovnice pak můžeme vidět, že velikost indukovaného napětí je úměrná otáčkám elektromotoru. Tedy dle velikosti napájecího napětí můžeme měnit otáčky.



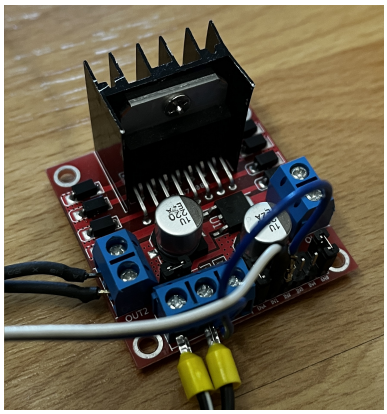
Obrázek 4.2. Fotografie elektromotoru

4.2 Snižovací měnič napětí

Ovládání rychlosti elektromotoru je zajištěno pomocí snižovacího měniče napětí. Měnič je zapojen do H-můstku. H-můstek umožňuje vytvořit na zátěži, v tomto případě na elektromotoru, obě polarity napětí, z toho vyplývá, že dovoluje motoru otáčení na obě strany. Tohoto jsem ovšem v této práci nevyužil, protože pro účely této práce je dostačující otáčení na jednu stranu.



Obrázek 4.3. Obvodové schéma H-můstku



Obrázek 4.4. Fotografie snižovacího měniče L298N

4.3 Světelná závora

Tato periferie slouží ke generování vnějšího přerušení. Dá se říct, že světelná závora zahajuje zobrazování požadovaných znaků. Světelná závora je optoelektronické zařízení složené z vysílače a přijímače [6]. Vysílač bývá často infračervená LED dioda a přijímačem fototranzistor. Samotná závora je vytvořena kusem tvrdého papíru připevněného ze spodu rotoru.

Princip fungování je následující. Fototranzistor detekuje paprsek LED diody. Jakmile jej nedetekuje, znamená to, že paprsek byl přerušen a fototranzistor odešle signál. Tento signál generuje žádost o vnější přerušení.

Obrázek 4.5. Fotografie světelné závory

Kapitola 5
ST Nucleo - F302R8

STM32 mikrokontrolér Nucleo F302R8 obsahuje 64 výstupních pinů. Flash paměť je velká 64 kB a RAM paměť disponuje velikostí 16 kB. Mikrokontrolér může být napájen z hostitelského počítače pomocí USB kabelu nebo externího zdroje. STM32 Nucleo komunikuje a je zároveň napájeno pomocí USB typu A na straně počítače a pomocí mini USB kabelu na straně desky.

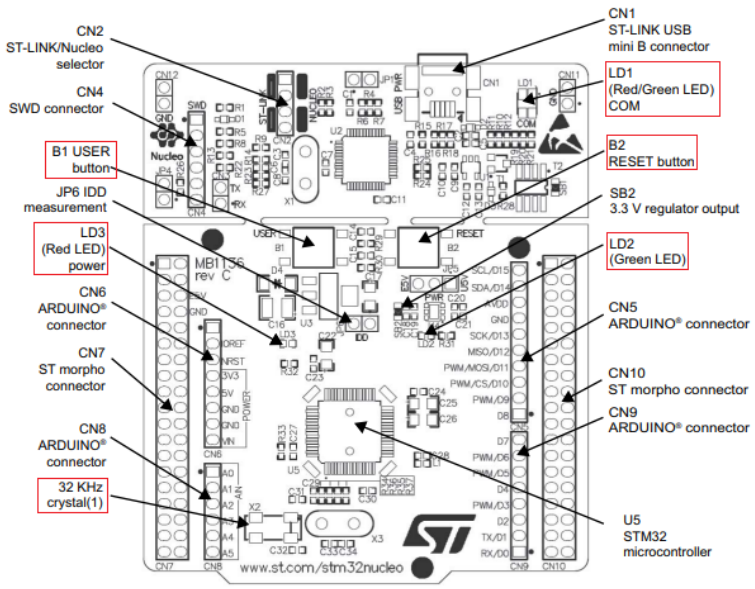
V následujících podkapitolách budou popsány některé periferie kitu ST Nucleo F302R8.

5.1 LED diody

Deska disponuje třemi LED diodami. Led dioda LD1 dle barvy a typu blikání poskytuje informaci o komunikaci s počítačem. LED dioda LD2 je programovatelná dioda nacházející se na pinu PB13. Třetí LED dioda LD3 PWR indikuje, zda je k dispozici 5V zdroj napětí.

5.2 Tlačítka

Deska také obsahuje 2 tlačítka. První tlačítko s názvem B1 USER je programovatelné tlačítko nacházející se na pinu PC13. Toto tlačítko používám k zapnutí nebo vypnutí napájení elektromotoru. Druhé tlačítko B2 RESET funguje jako reset celého mikrokontroléru.



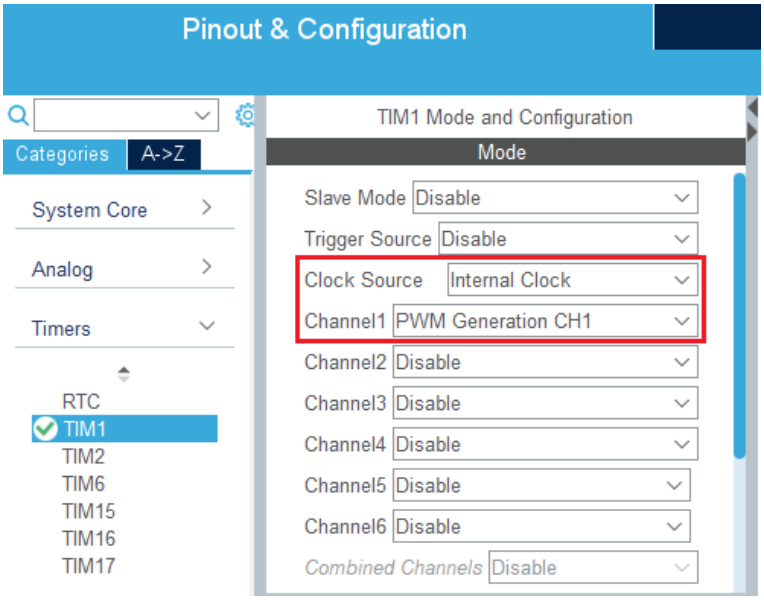
Obrázek 5.1. Rozložení periférií na desce [7]

5.3 Oscilátory

K dispozici jsou dva krystalické oscilátory, které mohou být použity jako hodinový signál. První vysokorychlostní externí oscilátor s frekvencí až 72 MHz a druhý vnitřní oscilátor s frekvencí 32 kHz. Oba tyto oscilátory mohou být použity jako čítače. Frekvence čítání může být následně zpomalena vhodným nastavením předděličky.

5.4 Konfigurace čítače a pulsně šířkové modulace

Pro správné nastavení pulsně šířkové modulace je potřeba nejdříve zkonfigurovat čítač. Pro tuto potřebu je možno použít jeden ze šesti dostupných čítačů. Já jsem použil čítač TIM1, který jsem nastavil na pin PC0. Pro správné fungování čítače je vhodné jej zkonfigurovat ve vývojovém prostředí STM32CubeMX. V záložce Pinout & Configuration je nutné kliknout na TIM1 a podívat se do části s označením Mode. Následně je potřeba vybrat na řádku Clock Source - Internal Clock a na řádku Channel1 zvolit PWM Generation CH1.

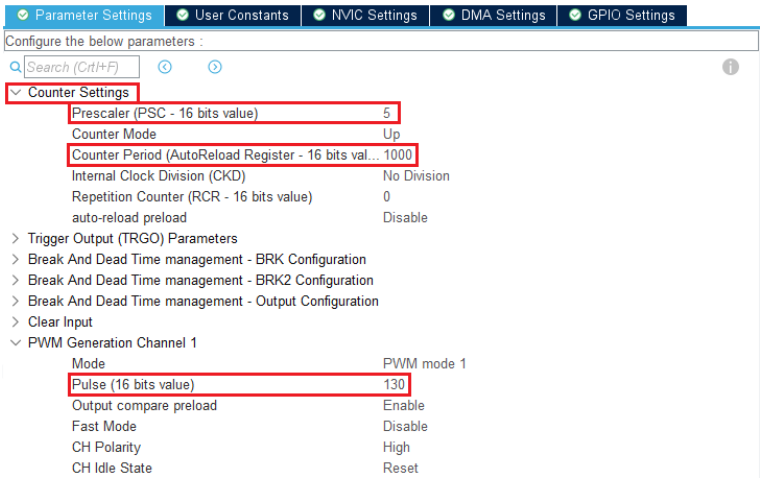


Obrázek 5.2. Konfigurace čítače TIM1

Nyní je možno zkonfigurovat samotnou pulsně šířkovou modulaci. To lze v tomto případě udělat dvěma způsoby.

První způsob je konfigurace ve vývojovém prostředí STM32CubeMX. V tomto případě je postup stejný jako v případě konfigurace čítače TIM1 s tím rozdílem, že místo části Mode zamíříme do části Configuration. V této části musíme kliknout na záložku s názvem Parameter Settings. V podsložce Counter Settings je třeba nastavit Prescaler v závislosti na tom, jak rychle chceme, aby čítač zvyšoval svoji hodnotu. V mém případě jsem zvolil hodnotu 5, což znamená, že čítač zvyšuje svoji hodnotu pětkrát pomaleji, než je frekvence oscilátoru. Dále je potřeba nastavit periodu, do kolika bude čítač počítat. Tu jsem nastavil na hodnotu 1000. Nyní se můžeme posunout úplně dolů do sekce PWM Generation Channel 1, kde musíme nastavit řádek Pulse. Tento řádek určuje šířku pulsu, přičemž 0 znamená, že se negeneruje žádný puls, tj. napětí je 0 V, a 1000 znamená, že generujeme jeden dlouhý, nekonečný puls a napětí je rovno jeho vstupní hodnotě. V tomto případě používám 9V zdroj napětí, tedy napětí by bylo rovno 9 V.

Aby se LED diody neotáčely příliš rychle a text nebyl hodně roztažený, nebo naopak se LED diody točily příliš pomalu a text byl stlačený, tak se mi osvědčila hodnota 130. Bohužel nyní by PWM ještě nefungovala, neboť je potřeba explicitně zapnout čítač. Ten nelze ve vývojovém prostředí STM32CubeMX zapnout. Zapnout jej můžeme opět pomocí dvou způsobů. První je příkaz HAL_TIM_PWM_START(&htim1,TIM_CHANNEL1) jehož parametry jsou název čítače, který chceme zapnout a na jakém kanále se čítač nachází. Druhý způsob zapnutí je příkazem TIM1->CR1=1. Ten zapne bit, který přímo ovládá čítač a čítač proto začíná počítat.

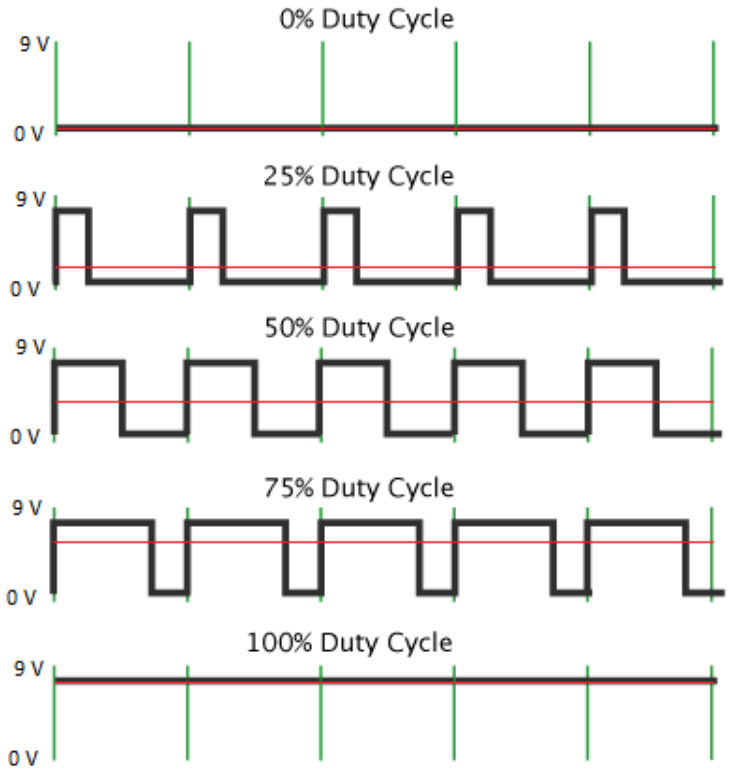


Obrázek 5.3. Konfigurace PWM pomocí STM32CubeMX

Druhou metodou je konfigurace pomocí přístupu na jednotlivé bity ovládající nastavení PWM a čítače za pomoci série příkazů [8]:

- TIM1->BDTR=TIM_BDTR_MOE;
  - zajišťuje správný chod PWM po vygenerování pulsu
- TIM1->PSC=5;
  - nastaví hodnotu předděličky na požadované číslo
- TIM1->CCER=1;
  - zajišťuje reakci na náběžnou hranu čítače
- TIM1->ARR=1000;
  - nastaví čítání do 1000
- TIM1->CCR1=130;
  - nastavuje šířku pulsu
- TIM1->CR1=1;
  - spustí čítač

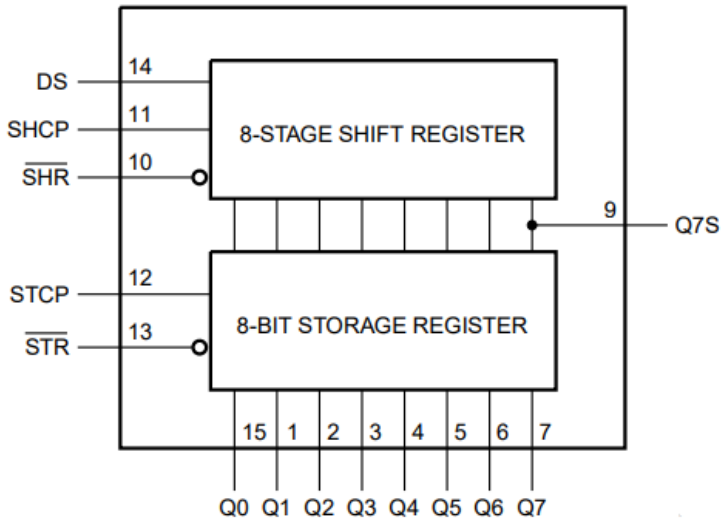
Na následujícím obrázku lze vidět, jak se s měnící se střídou PWM mění i střední hodnota napětí.



Obrázek 5.4. Změna střední hodnoty napětí s měnící se střídou (převzato z [9])

Kapitola 6
Shift registry 74HC594

Pro ovládání všech šestnácti LED diod jsou použity dva shift registry s výstupními registry typu 74HC594. Každý shift registr má 8 bitů a jsou zapojeny v sérii. Bity jsou posouvány od nejnižšího k nejvyššímu. DS pin (14) funguje jako vstupní datový pin. SHCP pin (11) funguje jako hodinový vstup pro shift registr, tedy s každým hodinovým impulsem je načtena hodnota, která se nachází na DS pinu. STCP pin (12) je hodinový signál pro výstupní registr. Pokud je na tomto pinu logická jednička data uložená ve výstupním registru jsou odeslána na výstup, kterým jsou LED diody.



Obrázek 6.1. Funkční diagram shift registru [10]

Pro pohodlné ovládání shift registru jsem vytvořil následující 4 funkce. Tyto funkce se nacházejí v souboru *shiftRegisterControl.c*.

6.1 Funkce turnOn

Jak název této funkce napovídá, tato funkce pošle na datový vstup shift registru logickou jedničku, která později zapne LED diodu. V informatice je obvyklé používat pro zapnuté objekty hodnotu 1 a pro vypnuté hodnotu 0. Proto zde píši, že do shift registru posílám jedničku, ačkoliv příkaz GPIO_PIN_RESET v podstatě do shift registru pošle 0, která způsobí uzemnění příslušného pinu a rozsvícení LED diody.

Funkce je typu *void*, protože není zapotřebí, aby funkce vracela jakoukoliv hodnotu. Funkce vypadá takto:

```
11 //Tato funkce načítá do shift registru 1, která rozsvítí LED diodu
12 void turnOn(){
13     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_RESET);
14     //vygeneruje na pinu PC7 logickou jedničku a pošle ji na DS pin shift registru
15     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
16     //vygeneruje náběžnou hranu na pinu PA8, kterou pošle na SHCP pin shift registru
17     getPause();
18     //funkce pro zpoždění, aby vznikl dostatečně dlouhý signál
19     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
20     //vygeneruje sestupnou hranu na pinu PA8, kterou pošle na SHCP pin shift registru
21     getPause();
22 }
```

Obrázek 6.2. Funkce turnOn

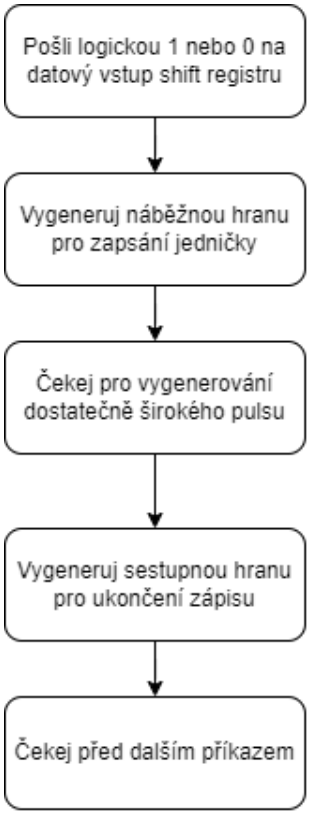
6.2 Funkce turnOff

Obdobně jako u funkce *turnOn* je i zde z názvu patrné co tato funkce dělá. Tato funkce pošle na datový vstup shift registru logickou nulu, která později zhasne LED diodu. Funkce vypadá následovně:

```
24 //Tato funkce načítá do shift registru 0, která zhasne LED diodu
25 void turnOff(){
26     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_SET);
27     //vygeneruje na pinu PC7 logickou nulu a pošle ji na DS pin shift registru
28     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
29     //vygeneruje náběžnou hranu na pinu PA8, kterou pošle na SHCP pin shift registru
30     getPause();
31     //funkce pro zpoždění, aby vznikl dostatečně dlouhý signál
32     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
33     //vygeneruje sestupnou hranu na pinu PA8, kterou pošle na SHCP pin shift registru
34     getPause();
35 }
```

Obrázek 6.3. Funkce turnOff

Princip funkcí *turnOn* a *turnOff* je vysvětlen na následujícím vývojovém diagramu:



Obrázek 6.4. Vývojový diagram funkcí turnOn a turnOff

6.3 Funkce turnOffAll

Tato funkce slouží k zápisu šestnácti logických nul za sebou do shift registrů. Tuto funkci jsem vytvořil, abych pro zhasnutí všech diod najednou nemusel šestnáctkrát volat funkci *turnOff*.

6.4 Funkce pushDataOut

Toto je poslední funkce, která slouží k ovládání shift registrů. Tato funkce má za úkol poslat hodinový signál na pin STCP. Tento vygenerovaný signál způsobí zobrazení obsahu shift registrů na jejich výstupy. Na výstupy shift registrů jsou připojeny LED diody. Pokud je na určitém místě shift registru zapsána logická jednička, příslušná LED dioda se rozsvítí, a naopak pokud je na daném místě v shift registru logická nula, příslušná LED dioda zhasne.

```
108 //Tato funkce pošle do shift registru signál, který odešle obsah shift registrů na jeho výstup
109 void pushDataOut(){
110     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
111     //vygeneruje náběžnou hranu na pinu PA9, kterou pošle na STCP pin shift registru
112     getPause();
113     //funkce pro zpoždění, aby vznikl dostatečně dlouhý signál
114     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
115     //vygeneruje sestupnou hranu na pinu PA9, kterou pošle na STCP pin shift registru
116     getPause();
117 }
```

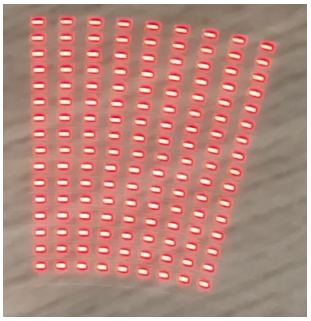
Obrázek 6.5. Funkce pushDataOut

Kapitola 7
Program - běžící text

V této kapitole podrobně popíšu princip fungování celého program, všechny vytvořené funkce potřebné pro chod programu a celkové fungování programu.

7.1 Soubor alphabet.c

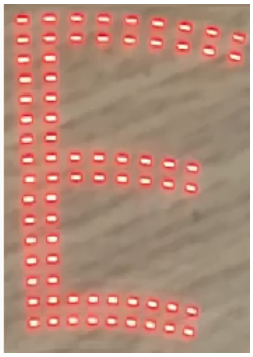
K tomuto zdrojovému souboru patří hlavičkový soubor alphabet.h, ve kterém jsou definovaná pole pro jednotlivé znaky. Následující znaky je možné zobrazit: všechna velká písmena (bez háčků), čísla od nuly do desítky, +, -, =, x (krát), :, /, _, ., !, ?, kulaté závorky, hranaté závorky, špičaté závorky a #. Pokud je zadán znak, který není v souboru alphabet.c rozsvítí se všechny diody. Vzniká tak svítící obdélník, ve kterém by se nacházel zadaný znak.



Obrázek 7.1. Zadaný znak není v souboru alphabet.c

V tomto zdrojovém souboru se nachází dvourozměrná pole obsahující nuly a jedničky. Pole jsou devět sloupců široká a šestnáct řádků vysoká. Jedničky v poli reprezentují LED diody, které jsou rozsvíceny a nuly reprezentují zhasnuté LED diody. Na následujícím obrázku je příklad pole pro písmeno E.

```
95  //písmeno E
96  char letterE[16][9]={
97          {1,1,1,1,1,1,1,1,1},
98          {1,1,1,1,1,1,1,1,1},
99          {1,1,0,0,0,0,0,0,0},
100         {1,1,0,0,0,0,0,0,0},
101         {1,1,0,0,0,0,0,0,0},
102         {1,1,0,0,0,0,0,0,0},
103         {1,1,0,0,0,0,0,0,0},
104         {1,1,1,1,1,1,1,1,0},
105         {1,1,1,1,1,1,1,1,0},
106         {1,1,0,0,0,0,0,0,0},
107         {1,1,0,0,0,0,0,0,0},
108         {1,1,0,0,0,0,0,0,0},
109         {1,1,0,0,0,0,0,0,0},
110         {1,1,0,0,0,0,0,0,0},
111         {1,1,1,1,1,1,1,1,1},
112         {1,1,1,1,1,1,1,1,1},
113 };
```



Obrázek 7.2. Pole definující písmeno E s ukázkou

7.2 Soubor writtingLetters.c

Tento zdrojový soubor obsahuje pět funkcí. Z těchto pěti funkcí, tři slouží k vytvoření různě velkých pauz, dále je zde funkce sloužící k tisku písmen. Poslední funkce má za úkol přiřadit k zadanému znaku příslušný znak z knihovny znaků *alphabet.c*.

7.2.1 Funkce getSpace

Funkce *getSpace* datového typu *void* je prázdný *for* cyklus, který má za úkol vytvořit mezeru mezi písmeny. Zavoláním této funkce procesor inkrementuje inicializátor a nemůže provádět nic jiného. Po skončení *for* cyklu program pokračuje v běhu. Nastavením podmínky $i < x$, kde x je libovolné celé číslo, lze nastavit velikost mezery mezi písmeny. Já jsem experimentálně za x zvolil číslo 800. Tato velikost mezery mi přišla proporcionálně vhodná.

```
12  void getSpace(){
13      for(int i=0; i<800; i++){
14          //vytvoří mezeru mezi písmeny
15      }
16  }
```

Obrázek 7.3. Kód funkce getSpace

7.2.2 Funkce getPauseLong

Funkce *getPauseLong* je stejně jako funkce *getSpace* datového typu *void* a také je to prázdný *for* cyklus. Ovšem tato funkce určuje, jak dlouho bude každý sloupec LED diod svítit. Zde jsem zvolil, aby *for* cyklus provedl 20 opakování (0 - 19). Toto je poměrně důležitá hodnota, protože se podílí na tom, zda budou písmena široká nebo úzká.

7.2.3 Funkce getPause

I tato funkce je datového typu *void* a opět je to prázdný *for* cyklus. Tuto funkci jsem vytvořil a používám ji ve zdrojovém souboru *shiftRegisterControl.c*. Zde slouží k vytvoření dostatečně dlouhého pulsu pro zápis a čtení dat shift registrů.

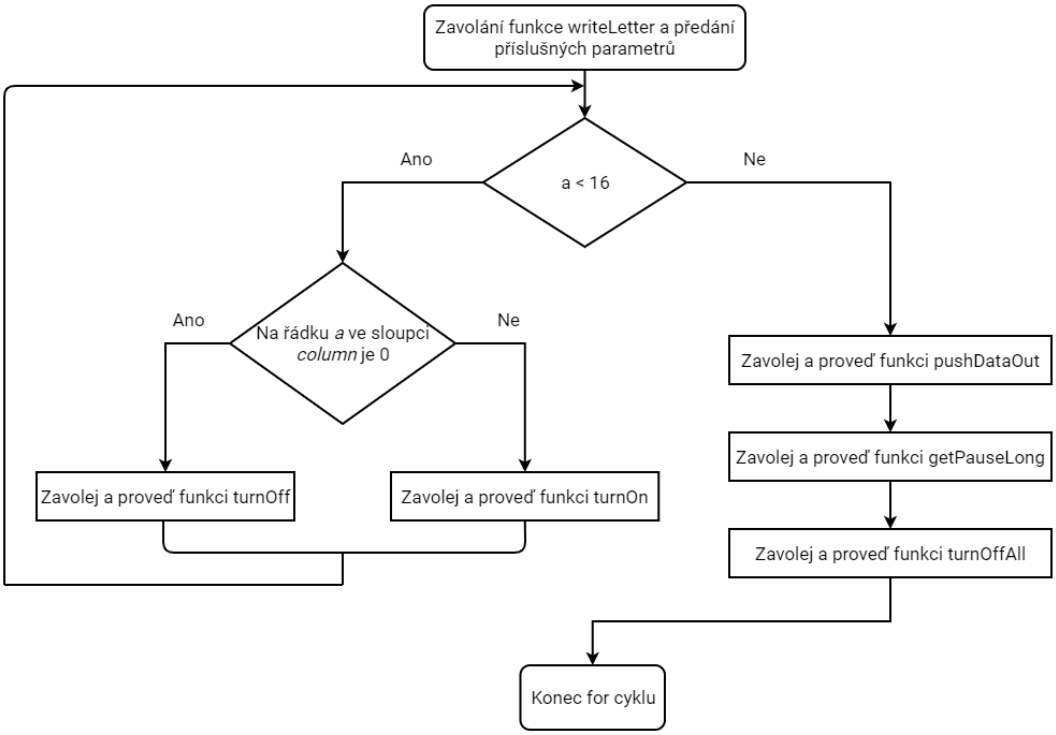
7.2.4 Funkce writeLetter

Tato funkce je datového typu *void* a obsahuje dva parametry. Prvním parametrem je proměnná datového typu *short* s názvem *column*. Tato proměnná určuje, který sloupec požadovaného znaku se má zobrazit. Druhým parametrem je dvourozměrné pole znaku, který chceme zobrazit.

```
36  void writeLetter(short column, char letter[16][9]){
37      for (int a=0; a<16; a++){
38          if(letter[a][column]==0){
39              turnOff();
40          }
41          else{
42              turnOn();
43          }
44      }
45      pushDataOut();
46      getPauseLong();
47      turnOffAll();
48  }
```

Obrázek 7.4. Kód funkce writeLetter

Samotný princip fungování této funkce je poměrně prostý a je vysvětlen na následujícím vývojovém diagramu.



Obrázek 7.5. Vývojový diagram funkce writeLetter

7.2.5 Funkce getArray

Tato funkce již není datovým typem *void*, ale je to ukazatel na ukazatele *char*. Jelikož jazyk C neumožňuje vnořené funkce je zapotřebí ukázat na místo v paměti, kde se funkce nachází. Parametrem této funkce je proměnná *letter* datového typu *char*. Funkce obsahuje řídící strukturu *switch*. Tato funkce má za úkol ke každému znaku, který je zadán na příslušném řádku ve zdrojovém souboru *main.c* přiřadit příslušný znak z knihovny znaků *alphabet.c*. Například pokud bude požadavek na zobrazení písmena A, tento znak se stane parametrem funkce *getArray* a porovnává se s ostatními *case* případy. Jakmile najde shodu, *switch* se přeruší a vrátí pole příslušného znaku. Pokud není nalezena shoda proměnné *letter* a *case* zobrazí se defaultní znak error.

```
55  char ** getArray(char letter){
56      switch(letter){
57      case 'A':
58          return (char ** )letterA;
59          break;
60
61      case 'B':
62          return (char ** )letterB;
63          break;
64
65      case 'C':
66          return (char ** )letterC;
67          break;
```

Obrázek 7.6. Část kódu funkce getArray

7.3 Soubor main.c

Kostra zdrojového souboru *main.c* je automaticky generována za pomocí vývojového prostředí STM IDE. Zde je zapotřebí definovat proměnné, které chceme používat globálně. V tomto souboru se také zadávají znaky, které se mají následně zobrazit. Znaky jsou zadány jako pole znaků tedy *string*. Toto pole je automaticky ukončeno znakem \0. Toho následně využívám při zjišťování, zda už byly zobrazeny všechny znaky. V tomto souboru je možné zkonfigurovat jednotlivé bity pulsně šířkové modulace. Hlavní částí tohoto programu je nekonečná *while* smyčka. Tato *while* smyčka zajišťuje neustálý běh programu.

```
45  char word[]="AHOJ";              //Zde uloženy znaky které budou zobrazeny
46  unsigned char columnCounter=0;   //Určuje který sloupec znaku je  zobrazován
47  unsigned char letterCounter=0;   //Určuje kolikátý znak je zobrazován
48  unsigned char pwmState=0;        //Určuje stav PWM on/off
```

Obrázek 7.7. Použité globální proměnné

7.4 Soubor stm32f3xx_it.c

Kostra tohoto zdrojového souboru je také automaticky generována vývojovým prostředím STM IDE. V tomto souboru jsou definovány vnější přerušení. Pokud tedy dojde ke vnějšímu přerušení, program vyskočí ze smyčky *while* v souboru main.c a přejde se do příslušné části kódu v souboru *stm32f3xx_it.c*. Já používám dvě přerušení. První přerušení je generováno stiskem tlačítka B1 USER button. Druhé přerušení je generováno světelnou závorou. Ta je připojena na pin PA0.

Přerušení od tlačítka B1 USER používám k zapínání a vypínání pulsně šířkové modulace. Na počátku je proměnná *pwmState=0*. Pokud tedy stisknu tlačítko podmínka se vyhodnotí kladně, nastavím proměnnou *pwmState=1* a nastavím šířku generovaného pulsu PWM na požadovanou hodnotu. Při dalším stisknutí je podmínka vyhodnocena záporně a provede se druhá část příkazu, kde se nastaví *pwmState=0* a šířka pulsu na 0.

```
239     if(pwmState == 0){
240         pwmState = 1;
241         TIM1->CCR1=150;
242     }
243     else{
244         pwmState = 0;
245         TIM1->CCR1=0;
246     }
```

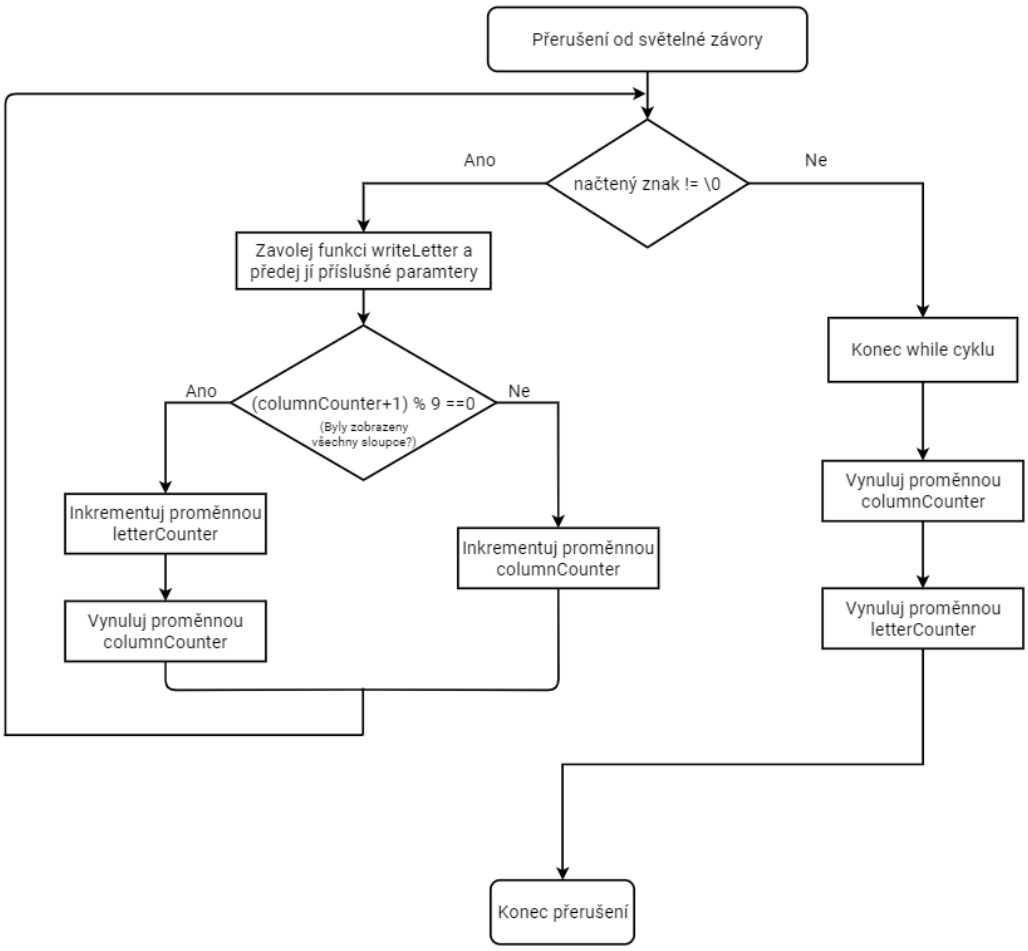
Obrázek 7.8. Zapínání a vypínání PWM

Ve druhém přerušení generovaném světelnou závorou se provádí část kódu, která má za následek zobrazení znaku nebo skupiny znaků. Kód je složen z jedné *while* smyčky a *if else* příkazu. Podmínka *while* smyčky kontroluje, zda se již zobrazily všechny znaky, zatímco *if else* hlídá, jestli byly zobrazeny všechny sloupce příslušného znaku.

```
213     while(word[letterCounter]!='\0'){
214       writeLetter(columnCounter, getArray(word[letterCounter]));
215       if((columnCounter+1) % 9 == 0){
216         letterCounter++;
217         columnCounter=0;
218         getSpace();
219       }
220       else{
221         columnCounter++;
222       }
223     }
224     columnCounter=0;
225     letterCounter=0;
```

Obrázek 7.9. Kód prováděný při přerušení od světelné závory

Princip fungování je vysvětlen na následujícím vývojovém diagramu.



Obrázek 7.10. Vývojový diagram kódu při přerušení od světelné závory

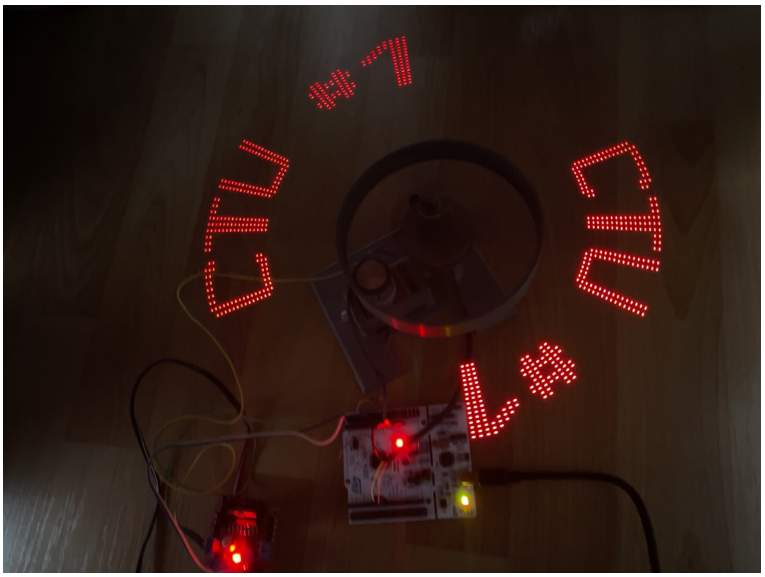
V tomto souboru používám proměnné, které jsou definovány v souboru *main.c*. Pro přístup k těmto proměnným slouží příkaz *extern* za kterým následuje datový typ a název proměnné.

```
47  extern char word[];          //Zde uloženy znaky které budou zobrazeny
48  extern char columnCounter;   //Určuje který sloupec znaku je zobrazován
49  extern char letterCounter;   //Určuje kolikátý znak je zobrazován
50  extern char pwmState;        //Určuje stav PWM on/off
```

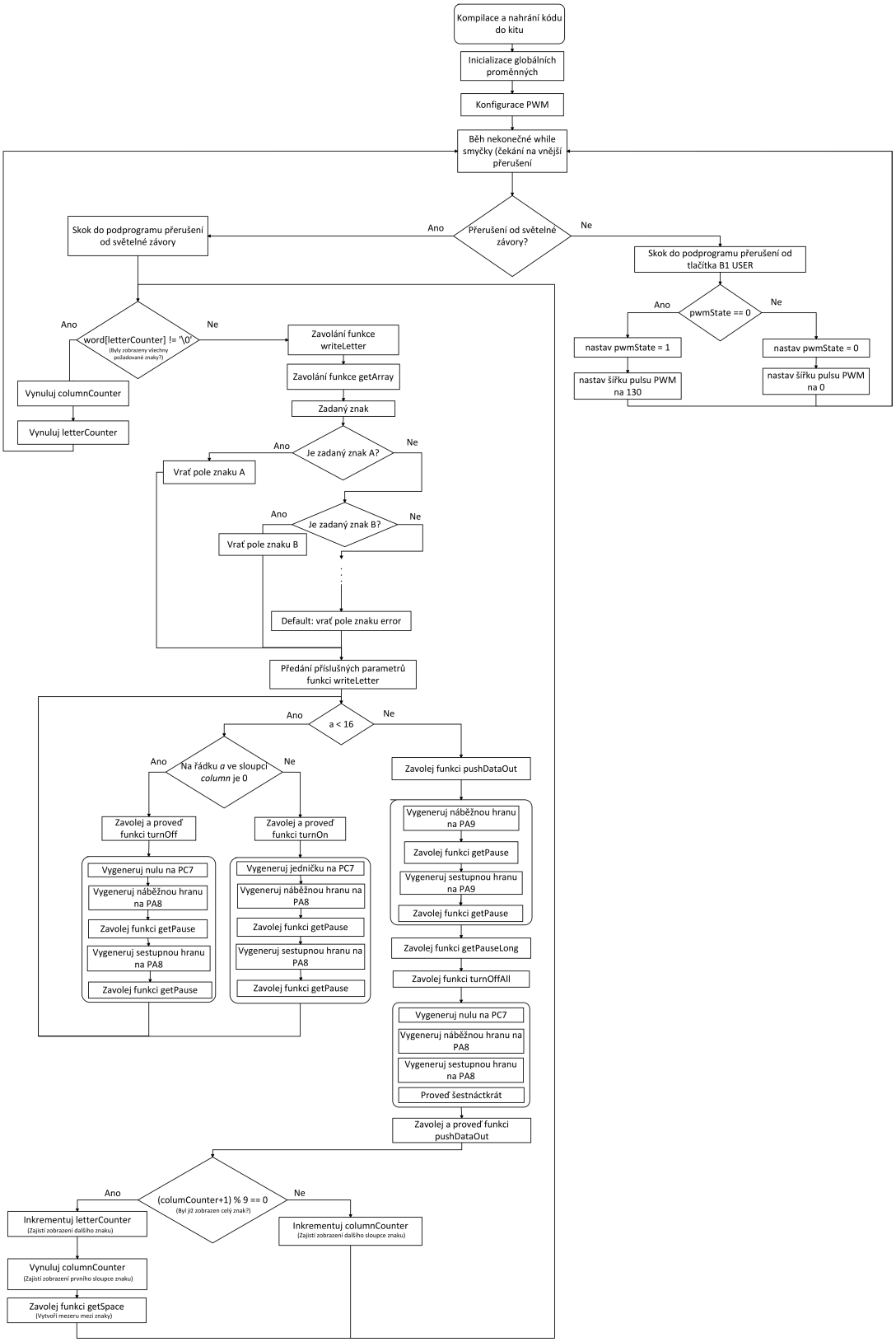
Obrázek 7.11. Přístup ke globálním proměnným

7.5 Fungování celého programu

Nyní stručně popíšu fungování celého programu. Podrobnější funkce bude vysvětlena pomocí vývojového diagramu. První je nutná kompilace a nahrání programu do kitu ST Nucelo. Po nahrání a spuštění programu je nutné zapnout elektromotor pomocí tlačítka B1 USER. Po tomto kroku nastává rozběh rotoru zařízení. Jakmile se zařízení roztočí a světelná závora vyšle signál o přerušení, přechází se z nekonečné *while* smyčky v main.c do podprogramu přerušení. Nyní je zavolána funkce *writeLetter*, která volá 2 parametry. A to sice kolikátý sloupec se má zobrazit a funkci *getArray*, která vrací pole požadovaného znaku. Následně funkce *writeLetter* projde prvky prvního sloupce pole a vyhodnotí, zda do shift registrů pošle jedničku nebo nulu (tzn. rozsvítí nebo zhasne LED diodu). Po ukončení funkce *writeLetter* se program vrací do přerušení, kde pomocí *if* podmínky zjistí, jestli byl zobrazen celý znak nebo jen jeho část. Pokud podmínka vyhodnotí, že byl zobrazen celý znak, inkrementuje se proměnná určující, kolikátý znak má být zobrazen. Jakmile je vyhodnoceno, že byly zobrazeny všechny požadované znaky, vynulují se proměnné určující, kolikátý sloupec znaku má být zobrazen a kolikátý znak má být zobrazen. Nyní se program vrátí zpět do nekonečné *while* smyčky v *main.c* a čeká na další přerušení od světelné závory či tlačítka B1 USER.



Obrázek 7.12. Ukázka funkčnosti programu



Obrázek 7.13. Vývojový diagram celého programu

Kapitola 8
Závěr

Cílem této bakalářské práce bylo vytvořit program pro kit ST Nucleo, který by umožnil zobrazit požadované znaky, a to pouze za pomoci pásu šestnácti LED diod. Program jsem psal v programovacím jazyce C, protože je to stále populární jazyk, který se v praxi používá v mnoho odvětvích. Pro napsání správně fungujícího programu bylo nejprve nutné nastudovat základy programovacího jazyka C. Těmto základům se stručně věnuji ve druhé kapitole této bakalářské práce.

Dalším důležitým krokem bylo zorientování se v samotném vývojovém prostředím STM32CubeIDE od společnosti STMicroelectronics. Jeho základní ovládání a některé funkce jsou popsány ve třetí kapitole.

Důležitou součástí bylo seznámení se s celým zařízením. A to ať už s principem fungování jednotlivých perfiérií, jako je například světelná závora, nebo nejpodstatnější částí celého zařízení, kterou je mikrokontrolér ST Nucleo F302R8. Pro ovládání všech šestnácti LED diod jsou použity 2 sériově zapojené shift registry SN74HC594. Tyto informace jsou obsaženy v kapitolách 4, 5 a 6.

V sedmé kapitole popisuji všechny funkce, které jsem vytvořil pro splnění zadání této práce. Tato kapitola mimo jiné obsahuje vývojový diagram celého programu a fotografii, jak vypadá běžící text ve skutečnosti.

Zkratky

PWM	Pulse width modulation (pulsně šířková modulace)
LED	Light-emitting diode (elektroluminiscenční dioda)
DC	Direct current (stejnosměrný proud)
3D	Three-dimensional (trojrozměrný)
DS	Serial data input (sériový datový vstup)
SHCP	Shift register clock input (hodinový vstup shift registru)
STCP	Storage register clock input (hodinový vstup výstupního registru)

Literatura

[1] Wikipedie. *Moorův zákon*. 2021.
https://cs.wikipedia.org/wiki/Moor%C5%AFv_z%C3%A1kon.

[2] Wikipedie. *Intel 4004*. 2021.
https://cs.wikipedia.org/wiki/Intel_4004.

[3] Lukáš Václavík. *Samsung rozjel 4nm výrobu.* 2021.
https://www.msn.com/cs-cz/zpravy/v%C4%9Bda-a-technika/samsung-rozjel-4nm-v%C3%BDrobu-na-p%C5%99%C3%AD%C5%A1t%C3%AD-rok-chyst%C3%A1-3nm-%C4%8Dipy-v-roce-2025-pak-2nm/ar-AAPe6fs.

[4] Wikipedia. *Hlavičkový soubor*. 2021.
https://cs.wikipedia.org/wiki/Hlavi%C4%8Dkov%C3%BD_soubor.

[5] Stanislav Vítek. *3. Základní řídicí struktury*. 2020.
https://cw.fel.cvut.cz/b191/_media/courses/b0b99prpa/prpa-lec03.pdf.

[6] the free encyclopedia Wikipedia. *Light curtain*. 2021.
https://en.wikipedia.org/wiki/Light_curtain.

[7] STMicroelectronics. *UM1724 User manual*. 2020.
https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf.

[8] STMicroelectronics. *AN4776 Application note*. 2019.
https://www.st.com/resource/en/application_note/dm00236305-general purpose-timer-cookbook-for-stm32-microcontrollers-stmicroelectronics.pdf.

[9] Timothy Hirzel. *PWM*. 2018.
https://www.arduino.cc/en/Tutorial/Foundations/PWM.

[10] Nexperia B.V. *74HC594; 74HCT594 8-bit shift register with output register*. 2021.
https://www.nexperia.com/products/analog-logic-ics/i-o-expansion-logic/shift-registers/series/74HC594-74HCT594.html.