ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

Fakulta elektrotechnická
Katedra elektroniky a informačních technologií

BAKALÁŘSKÁ PRÁCE

Elektronická demonstrace třídění

Autor práce: Jan Růžička
Vedoucí práce: Ing. Petr Weissar, Ph.D.

2020

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jan RŮŽIČKA**
Osobní číslo:	**E18B0220P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a telekomunikace**
Téma práce:	**Elektronická demonstrace třídění**
Zadávající katedra:	**Katedra elektroniky a informačních technologií**

Zásady pro vypracování

Navrhněte a realizujte HW pro vizuální demonstraci třídících algoritmů

1. Vizualizaci založte na řadě vícebarevných LED, kdy barva reprezentuje hodnotu
2. Vyberte vhodné algoritmy a popište jejich funkci
3. Vytvořte demo-aplikaci na PC, kde bude možné ověřit funkčnost řešení. Uvažte přenositelnost klíčových částí kódu do mikrokontroléru

Rozsah bakalářské práce: **30 – 40 stran**
Rozsah grafických prací: **podle doporučení vedoucího**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Mikroprocesory a mikropočítače / Jiří Pinker. – 1. vyd. – Praha : BEN – technická literatura, 2004. – 159 s. : il.. – ISBN 80-7300-110-1
2. Yiu, Joseph. The Definitive Guide to ARM? Cortex?-M3 and Cortex?-M4 Processors, Elsevier Science & Technology, 2013

Vedoucí bakalářské práce: **Ing. Petr Weissar, Ph.D.**
Katedra elektroniky a informačních technologií

Datum zadání bakalářské práce: **9. října 2020**
Termín odevzdání bakalářské práce: **27. května 2021**

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Ing. Jiří Hammerbauer, Ph.D.
vedoucí katedry

V Plzni dne 9. října 2020

Abstrakt

Práce se zabývá vizualizací třídících algoritmů pomocí simulace na PC a pomocí ovládání vícebarevných RGB LED. Simulace na PC je psaná ve vývojovém prostředí VisualStudio v jazyce C#. Pro vizualizaci na RGB LED byla vybrána sada NeoPixel Stick od společnosti Adafruit, která je osazena osmicí inteligentních RGB LED diod WS2812. K ovládání sady RGB LED diod byla použita vývojová deska Nucleo STM32 F411RE naprogramována jazykem C ve vývojovém prostředí Atolic TrueStudio. Dále se práce zabývá základním principem použitých algoritmů, a to algoritmů SelectionSort, BubbleSort, InsertionSort, HeapSort a QuickSort.

Klíčová slova

Algoritmus, stabilita třídících algoritmů, SelectionSort, BubbleSort, InsertionSort, HeapSort, QuickSort, NeoPixelStick, inteligentní RGB LED diody WS2812, vývojová deska Nucleo, STM32F411RE

Abstract

Růžička, Jan. Electronic demonstration of sorting [Elektronická demonstrace třídění]. Pilsen, 2021. Bachelor thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Electronics and Information Technologies. Supervisor: Ing. Petr Weissar, Ph.D.

The thesis deals with the visualization of sorting algorithms by using PC simulation and by controlling multi-color RGB LEDs. For visualization on RGB LEDs was chosen the Neopixel stick set from Adafruit. For controlling the set of RGB LEDs was used a Nucleo STM32 F411RE development board programmed in C language in the Atolic TrueStudio development environment. The thesis also deals with the basic principle of used algorithms, namely SelectionSort, BubbleSort, InsertionSort, HeapSort and QuickSort algorithms.

Keywords

Algorithm, stability of sorting algorithms, SelectionSort, BubbleSort, InsertionSort, HeapSort, QuickSort, NeoPixelStick, intelligent RGB LEDs WS2812, development board Nucleo STM32F411RE

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Měcholupech dne 24. května 2021

Jan Růžička

.....

Podpis

Obsah

Seznam obrázků

Seznam symbolů a zkratek

PWM	Pulse Width Modulation (pulzně šířková modulace)
RGB	Red Green Blue (zkratka tří barevných složek)
LED	Light-Emitting Diode (elektroluminiscenční dioda)
GPIO	General-Purpose I/O (univerzální vstupy/výstupy)
RCC	Reset and Clock Control (ovládání resetování a hodin)
USART	Universal synchronous asynchronous receiver transmitter (Univerzální synchronní/asynchronní vysílač/přijímač)
TIM	Timer (časovač)
SPI	Serial Peripheral Interface (periférie sériového rozhraní)
PSC	Prescaler (předdělička)
ARR	Auto-Reload Registr (automaticky obnovovací registr)
GCC	GNU Compiler Collection (sbírka překladačů GNU)

Úvod

Práce zahrnuje dvě části. První část se zaměřuje na demo-aplikaci psanou v jazyce C#. Tato aplikace slouží jako simulace třídění náhodného pole hodnot pomocí vícebarevných LED, kdy každá barva reprezentuje určitou hodnotu. V druhé části provádím přímou aplikaci kódu, tentokrát psanou v jazyce C, na mikroprocesor STM32F411, který ovládá sadu vícebarevných LED. Pro vizualizaci jsem si vybral tyto algoritmy: SelectionSort, BubbleSort, InsertionSort, HeapSort, QuickSort.

Vizualizace třídících algoritmů by mohla posloužit jako pomoc k pochopení principu daných třídících algoritmů, nebo naznačit základy práce s mikroprocesory.

Při rychlém průzkumu jsem našel podobné práce, které se týkali vizualizace třídících algoritmů, nicméně většina prací byla řešena pouze softwarovou vizualizací.

Teoretická část

1 Algoritmus

Pro pochopení některých výrazů, které budou použity u popisu třídících algoritmů, je třeba se ještě zabývat samotným algoritmem.

Jednoduše řečeno, algoritmus je návod k řešení nějakého problému. Na rozdíl od nás jsou počítače stroje, které nemyslí a je proto nutné jim každý úkol dopodrobna popsat. Můžeme se na algoritmus podívat jako na úkol, který lidé dělají již automaticky. Například "vyčisti si zuby" očividně nemůže být algoritmus, ale třeba "vezmi kartáček, otevři pusu, přilož kartáček k puse, kruhovými pohyby přejížděj kartáčkem přes zuby" by se již dalo považovat za pravý algoritmus. V informační technice se samozřejmě neřeší problémy jako čištění zubů. Většinou se řeší problémy jako seřazení prvků podle velikosti, vyhledání prvků a podobné.

Formálněji lze algoritmus charakterizovat jako předpis, který pro různá vstupní data vrátí po konečném počtu kroků správný výsledek. [2]

Algoritmus musí splňovat následující požadavky:

- Elementárnost – algoritmus se skládá z konečného počtu jednoduchých (elementárních kroků)
- Konečnost – algoritmus se nemůže zacyklit a musí vždy skončit v libovolném konečném počtu kroků
- Obecnost – algoritmus vždy musí skončit správným výsledkem a mít tak ošetřené všechny možné vstupy
- Determinovanost – v každém kroku musí jít určit, zda proces skončil a pokud ne, lze jednoznačně říct, jak bude pokračovat. [2]

1.1 Složitost algoritmu

Abychom mohli porovnávat různé algoritmy řešící stejný problém, zavádí se pojem složitost algoritmu. Složitost je jinak řečeno náročnost algoritmu. Čím menší složitost, tím je algoritmus lepší.

Algoritmická složitost se dá dělit na dvě části, a to na časovou složitost a paměťovou složitost. [1][3]

1.1.1 Paměťová složitost

Paměťová složitost je spotřeba paměti a diskového prostoru v závislosti na vstupních datech. Udává se obvykle jako funkce počtu zpracovaných prvků. Nejlépe jsou na tom algoritmy, které nepotřebují přídavný prostor a jsou schopny pracovat přímo ze zpracovaných dat (těmto algoritmům se také říká, že pracují „na místě“). [2][7]

1.1.2 Časová složitost

Časová složitost algoritmu vyjadřuje závislost času potřebného pro provedení výpočtu na rozsahu (velikosti) vstupních dat. Časová složitost se nemůže vyjadřovat v jednotkách času, protože rychlost provedení nezávisí pouze na určitém algoritmu, ale také na použitém jazyku, procesoru atp. [1][2]

2 Třídící algoritmy

Třídění neboli řazení, je termín, který se používá v informatice pro uspořádávání prvků do posloupnosti. Tato posloupnost má určitou logiku (seřazení dle velikosti, dle abecedy atp.), díky které se s takto setříděnými prvky dále snadněji pracuje a umožnuje provádět různé operace dle daných požadavků. [1][6]

2.1 Stabilita třídicích algoritmů

Stabilita u třídicích algoritmů se vyjadřuje tak, že po roztřídění prvků budou prvky se stejnou hodnotou mít vůči sobě stejnou pozici jako na začátku třídění. Pokud tedy máme například v poli hodnot dvě stejné hodnoty a jedna je více vpravo, tak po roztřídění bude hodnota, co byla více vpravo, stále vpravo.

2.2 Základní rozdělení třídících algoritmů

Algoritmy třídění se dají rozdělit podle různých kritérií.

2.2.1 Dělení podle umístění dat

- Algoritmy vnitřního třídění – data jsou uložena ve vnitřní operační paměti a algoritmus tak má možnost k nim libovolně přistupovat. Počet prvků posloupnosti je znám předem. Dále se budeme zabývat právě těmito algoritmy.
- Algoritmy vnějšího třídění – data jsou uložena na vnějších pamětích (např. soubory na disku) z důvodu velkého množství dat. Počet prvků posloupnosti není znám předem. [5]

2.2.2 Dělení algoritmů vnitřního třídění podle základního principu řazení

- Třídění výběrem – vybírají se prvky z nesetříděné části a řadí se k setříděné části.
- Třídění vkládáním – bereme jednotlivé prvky neseřazené posloupnosti a vkládáme je na správné místo do setříděné části.
- Třídění výměnou – hledáme dvojici prvků, která je ve špatném pořadí a prvky v ní vzájemně vyměníme.
- Třídění slučováním/rozdělováním – celá posloupnost se rozdělí na menší části, které se seřadí. Výsledné seřazené části se poté sloučí takovým způsobem, aby i výsledek byl seřazený.

Většina algoritmů se nedá zařadit pouze do jedné kategorie, ale často se jedná o jejich různou kombinaci.[7]

2.3 Popis vybraných „základních“ třídících algoritmů

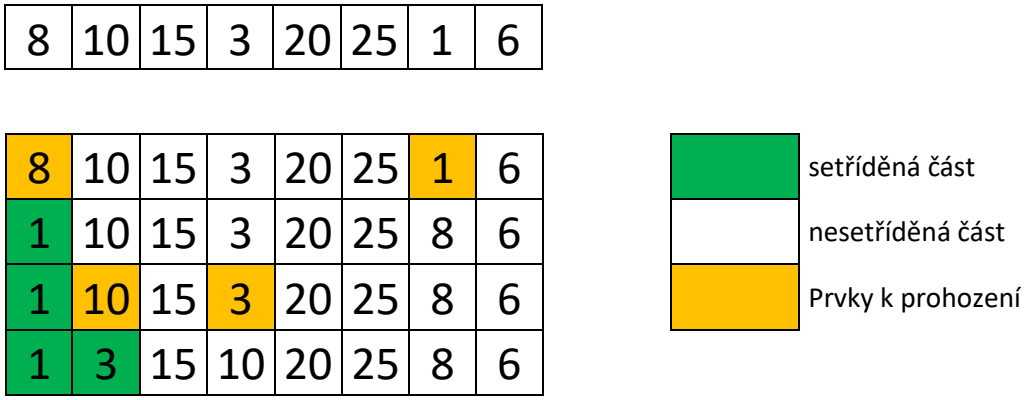
Tyto algoritmy se v praxi spíše nepoužívají, ale pro jejich jednoduchost slouží dobře pro akademický účel.

2.3.1 SelectionSort

Jak již český název *třídění přímým výběrem* naznačuje, algoritmus pracuje na principu třídění výběrem. Tříděná posloupnost se rozdělí na dvě části, a to na setříděnou a nesetříděnou část. Základní myšlenka je tedy naleznutí nejmenšího prvku, který se zařadí na začátek pole (případně největšího prvku, který se zařadí na konec). Tento prvek je již zařazený do seřazené části a v dalších krocích s ním nepočítáme.

Algoritmus je nestabilní a pomalý.[5][6][7]

Průběh algoritmu:



Obrázek 1. Ukázka zařazení prvních dvou prvků pomocí třídícího algoritmu SelectionSort.

Na obrázku (Obrázek 1) můžeme vidět průběh zařazení prvních dvou prvků. Nejprve se projede cyklem nesetříděná část od prvního do posledního prvku a najde se nejmenší prvek. Tento prvek (zde číslo 1) se zařadí na první místo nesetříděné části a prohodí se s prvkem, který je na této pozici (zde číslo 8). Tento prvek je již v setříděné části a dále s ním nepočítáme. Další cyklus opakuje stejný průběh a najde tak další nejmenší prvek v nesetříděné části (číslo 3) a prohodí ho s číslem na první pozici nesetříděné části (číslo 10). Máme tak zařazené první dva prvky a pokračujeme, dokud se nedostaneme na předposlední prvek. Poslední prvek již bude logicky zařazen správně, čímž ušetříme jeden krok.

1	10	15	3	20	25	8	6
1	3	15	10	20	25	8	6
1	3	6	10	20	25	8	15
1	3	6	8	20	25	10	15
1	3	6	8	10	25	20	15
1	3	6	8	10	15	20	25
1	3	6	8	10	15	20	25

setříděná část
nesetříděná část

Obrázek 2. Ukázka celého průběhu třídícího algoritmu SelectionSort.

2.3.2 BubbleSort

BubbleSort je česky znám jako *přímé třídění výměnou,* a je tak jasné, že pracuje na principu třídění výměnou. Tento algoritmus nemá vlastně žádné dobré vlastnosti oproti ostatním, ale je zajímavý svým průběhem, který se podobá přírodním jevům. Díky této zajímavosti se hodí, a často se používá, k akademickým účelům.

Algoritmus je stabilní, ale velmi pomalý. [5][6][7]

Průběh algoritmu:

8	10	15	3	20

8	10	15	3	20
8	10	15	3	20
8	10	15	3	20
8	10	3	15	20
8	10	3	15	20
8	10	15	3	20
8	10	15	3	20
8	10	15	3	20
8	10	15	3	20
8	10	3	15	20
8	10	3	15	20

setříděná část
nesetříděná část
porovnává se

Obrázek 3. Ukázka zařazení prvních dvou prvků pomocí algoritmu BubbleSort

Na obrázku (Obrázek 3) můžeme znovu vidět zařazení dvou prvních prvků (z důvodu pomalého průběhu algoritmu jsem zmenšil pole prvků pouze na 5 hodnot). Pole prvků procházíme zleva doprava a porovnáváme dva sousední prvky. Pokud je první prvek větší než následující, prohodí se mezi sebou a pokračuje se v porovnávání s dalším prvkem. Takto se "probublá" největší prvek až na konec a pokračuje se zase od začátku. Se zařazeným prvkem se již nepočítá. Poslední prvek je stejně jako u předchozího algoritmu zařazen.

8	10	15	3	20
8	10	3	15	20
8	3	10	15	20
3	8	10	15	20

setříděná část
nesetříděná část

Obrázek 4. Ukázka celého průběhu třídícího algoritmu BubbleSort

2.3.3 InsertionSort

InserionSort je česky znám jako *třídění přímým vkládáním.* Stejně jako například u algoritmu SelectionSort se i tady pole rozdělí na setříděnou a nesetříděnou část. Je velmi účinný na malých nebo již předtříděných polích, a proto je to jeden z mála jednoduchých třídících algoritmů, který můžeme potkat i v praxi. Algoritmus je stabilní a rychlý na malých polích. [5][6][7]

Průběh algoritmu:

8	1	3	10	25	20	6	15

8	1	3	10	25	20	6	15
8	1	3	10	25	20	6	15
1	8	3	10	25	20	6	15
1	8	3	10	25	20	6	15
1	8	3	10	25	20	6	15
1	3	8	10	25	20	6	15
1	3	8	10	25	20	6	15

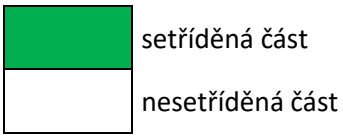
setříděná část
nesetříděná část
X uloženo v paměti

Obrázek 5. Ukázka zařazení prvních tří prvků pomocí algoritmu InsertionSort

Na obrázku (Obrázek 5) vidíme princip zařazení prvních tří prvků. První prvek (číslo 8) se považuje za již setříděný a začíná se od druhého prvku (číslo 1). Tento prvek se uloží do paměti. Prvek, který je v paměti, se posouvá doleva dokud nenarazí na menší prvek nebo začátek pole.

Tímto máme dva seřazené prvky a uložíme do paměti třetí prvek (číslo 3). Znovu posouváme prvek doleva, ale tentokrát narazí na menší prvek a vloží se před něj. Máme zařazené první tři prvky a pokračujeme až do posledního prvku, protože na rozdíl od předchozích algoritmů poslední prvek nemusí být zařazen správně.

8	3	1	10	25	20	6	15
3	8	1	10	25	20	6	15
1	3	8	10	25	20	6	15
1	3	8	10	25	20	6	15
1	3	8	10	25	20	6	15
1	3	8	10	20	25	6	15
1	3	6	8	10	20	25	15
1	3	6	8	10	15	20	25



Obrázek 6. Ukázka průběhu celého třídícího algoritmu InsertionSort

2.4 Popis vybraných „profesionálních“ třídících algoritmů

Tyto algoritmy se již objevují v praxi, ale jsou značně složitější než předchozí příklady.

2.4.1 HeapSort

HeapSort neboli řazení haldou je jeden z nejvyspělejší algoritmů založený na principu třídění. Algoritmus je založen na odtrhávání extrému (podobně jako SelectionSort), který přesune na konec pole. Na rozdíl od SelectionSort se ale nemusí projíždět celá nesetříděná část pole a algoritmus je proto mnohem rychlejší. Z tříděného pole se nejdříve vytvoří tzv. Halda (anglicky Heap), ze které dále složíme již setříděné pole. [5][6]

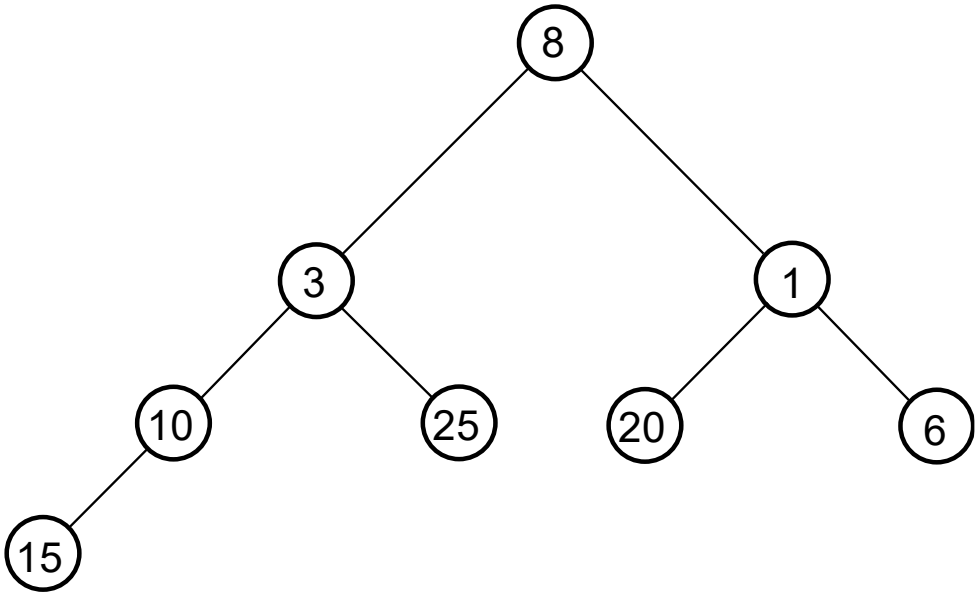
2.4.1.1 Halda

Jedná se o datovou strukturu, která se používá pro efektivní nalezení minimálního nebo maximálního prvku v konstantním čase. Halda musí splňovat tyto podmínky:

- všechna ”patra“ haldy kromě posledního jsou plně obsazeny prvky – každý otec má dva syny
- poslední patro se zaplňuje z leva
- oba synové jsou vždy menší nebo rovni otci. [5][8]

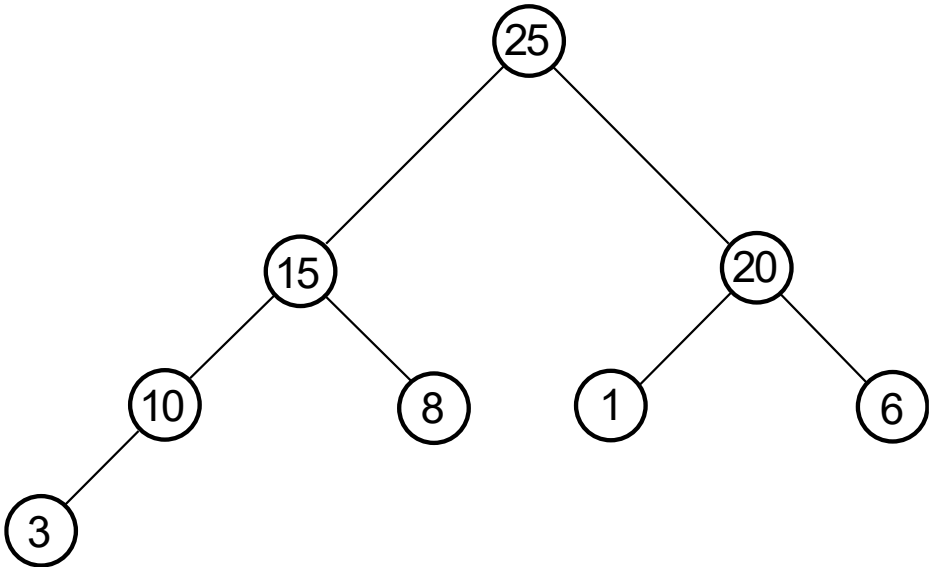
Sestrojení haldy z pole:

8	3	1	10	25	20	6	15



Obrázek 7. Prvotní "halda" utvořená z pole

Při prvním pohledu na obrázek (Obrázek 7) je zřejmé, že tento strom nesplňuje hlavní pravidlo haldy a to, že otec je větší než syn. Nejedná se proto ještě o haldu. Tento strom se utvoří jako prvotní a musí se seřadit dle pravidel haldy. Porovnávají se tedy postupně prvky od kořene (začátek stromu) dolů a pokud je otec menší než syn, tak se prohodí. Tento cyklus se opakuje, dokud není halda správně seřazena.



Obrázek 8. Hotová halda utvořená z pole

Průběh algoritmu:

Algoritmus s utvořenou haldou nepracuje jako se speciální stromovou strukturou, ale pracuje přímo s polem, na který se pouze dívá jako na haldu. Na obrázku (Obrázek 8) jsme si sestrojili haldu z nesetříděného pole. Tato halda by v poli vypadala takto:

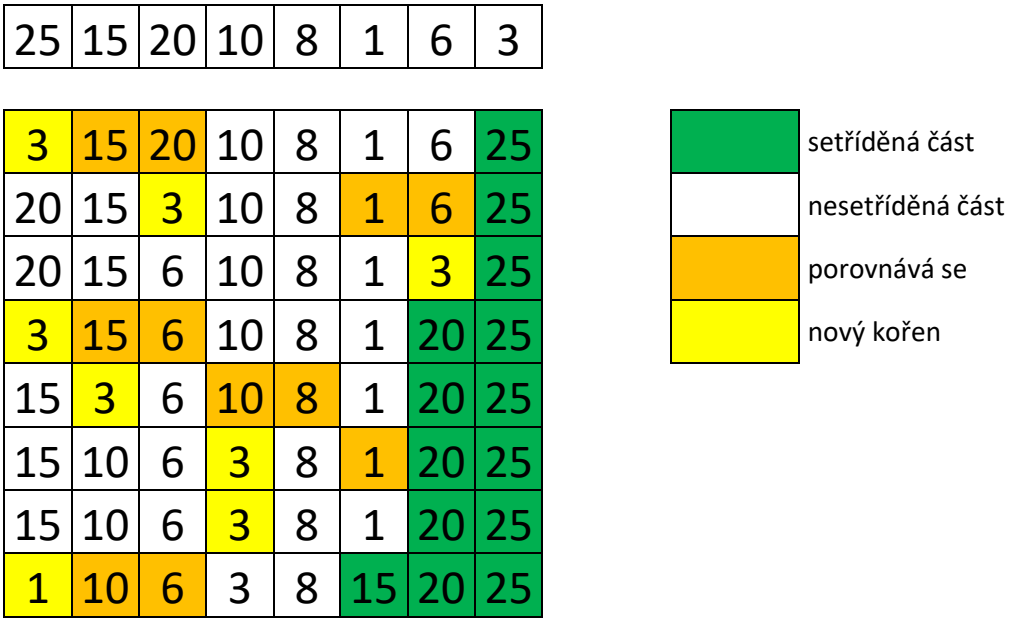
25	15	20	10	8	1	6	3

Obrázek 9. Pole hodnot sestrojené haldy

Jak již bylo psáno dříve, algoritmus je založený na stejném principu jako SelectionSort, a budeme tedy prohazovat maximum s posledním prvkem. S tímto maximem nadále nepočítáme a je zařazen do setříděné části. Maximum u tohoto pole bude vždy na jeho začátku, takže se nemusí hledat a ihned se prohodí s posledním prvkem nesetříděné části.

Zde ale nastává problém. Prohozením těchto prvků jsme dost pravděpodobně haldu rozbili a musí se znovu přetřídit. Porovná se tedy kořen s jeho syny, a pokud je některý z nich větší, tak se prohodí (v případě, že jsou větší oba, se prohodí s větším synem). Pokud se prohodil kořen se synem, musí se také porovnat další patra, dokud porovnávaný prvek nebude větší než synové, nebo nedojde až k poslednímu patru. To vše se opakuje, dokud se nedojde na poslední prvek neseřazené části, který již bude seřazen správně.

Algoritmus je nestabilní a rychlý.



Obrázek 10. Ukázka seřazení prvních 3 prvků pomocí algoritmu HeapSort

2.4.2 QuickSort

QuickSort je jeden z nejrychlejších algoritmů, který se velmi často používá v praxi. Algoritmus využívá principu rozděl a panuj, což znamená, že složitý problém rozdělíme na více menších, které se dají snadněji řešit.

Algoritmus si vybere nějaké číslo z pole. Toto číslo se dále bude jmenovat pivot. Dále se rozdělí pole hodnot na dvě části, a to tak, že v jedné části budou všechny menší hodnoty než pivot a ve druhé všechny větší než pivot. Na tyto vzniklé části se opět volá algoritmus, který je posléze rozdělí. Algoritmus se tedy opakuje, dokud jednotlivé části nebudou pole o jedné hodnotě a tím je pole setříděné. [5]

Hlavní problém, který se u algoritmu QuickSort řeší je volba pivota. Nejlepší pivot je takový, který nám rozdělí pole na dvě stejně velké části. Takový pivot se dá nalézt například pomocí mediánu. Tato metoda se často nepoužívá, protože nalezení mediánu algoritmus natolik zpomalí, že se to nevyplatí.

Dále se může pivot volit například jako první (nebo poslední, z 1/3 atp.) hodnota v poli. Tato metoda je ale velmi neúčinná na již předtříděných polích, protože zvolený pivot tak může být blízko maximu (případně minimu). Pokud by zvolený pivot byl přímo extrémem pole, nerozdělil by pole na dvě části, ale pouze by ho zmenšil o jeden prvek.

Výhodné může být tedy pivot volit náhodně. Tento výběr časově algoritmus výrazně nezatíží a pro většinu případů bude algoritmus velice spolehlivý a rychlý.

Další možnost je kontrolovat rekurzi, a pokud je průběh algoritmu pomalý, přepne se na jiný algoritmus jako například HeapSort, který má zaručenou spolehlivost. Tato metoda nemusí ošetřovat volbu pivota a má zaručenou složitost na jakémkoli poli a je proto nejpoužívanější.[5]

Průběh algoritmu:

8	3	1	10	25	20	6	15

8	3	1	10	25	20	6	15
15	3	1	10	25	20	6	8
15	3	1	10	25	20	6	8
3	15	1	10	25	20	6	8
3	1	15	10	25	20	6	8
3	1	15	10	25	20	6	8
3	1	15	10	25	20	6	8
3	1	15	10	25	20	6	8
3	1	6	10	25	20	15	8
3	1	6	8	25	20	15	10

	setříděná část
	nesetříděná část
X	pivot
	rozdělená část
X	První větší prvek

Obrázek 11. Ukázka prvního cyklu algoritmu QuickSort

Na obrázku (Obrázek 11. Ukázka prvního cyklu algoritmu QuickSort) je znázorněný první cyklus algoritmu QuickSort. Nejprve zvolíme pivota (zde první hodnota v poli neboli číslo 8) a prohodíme ji s poslední hodnotou, aby se s polem lépe pracovalo. Začneme porovnávat od začátku pole a zachováváme si pozici prvního většího čísla, než je pivot. Číslo 15 je větší než 8, a proto s ním nic neděláme a pokračujeme na další hodnotu. Číslo 3 je menší než pivot, a tak se prohodí s prvním větším číslem a přičteme jedničku k pozici prvku. Číslo 1 je znovu menší než pivot a znovu prohodíme a přičteme jedničku k pozici. Číslo 10 už není menší než pivot, proto ho necháme být a pokračujeme. Tímto principem pokračujeme až dojdeme k hodnotě před pivotem. Pivot poté prohodíme s číslem, které je na pozici, kterou jsme si počítali (první větší prvek) a máme tak zařazenou první hodnotu a ukončený první cyklus algoritmu.

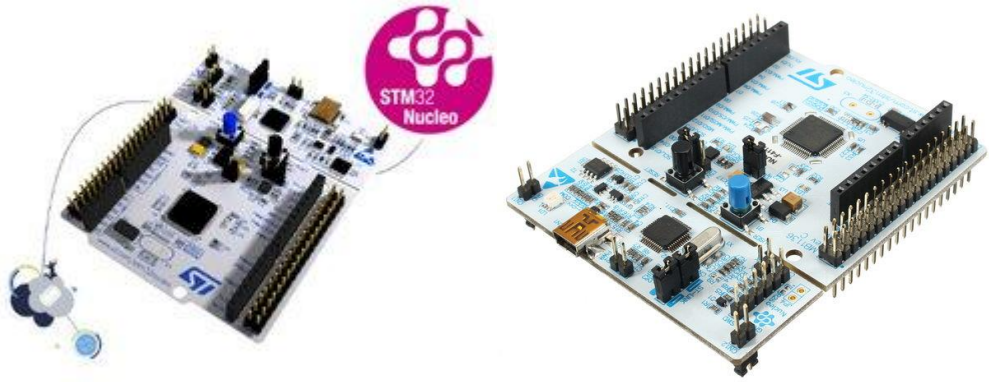
Praktická část

3 HARDWARE

V této části si rozebereme hardware použitý pro vizualizaci třídících programů pomocí mikroprocesoru.

3.1 Vývojová deska NUCLEO STM32 411RE

Tato vývojová deska, vyrobena firmou STMicroelectronic, byla použita pro vizualizaci třídících algoritmů na LED diodách.



Obrázek 12. Vývojová deska Nucleo STM32f411RE [9]

Vývojová deska je osazena mikrokontrolérem STM32F411RET6. Jeho hlavní vlastnosti jsou:

- 32bitový procesor ARM® Cortex®-M4 s FPU
- 100 MHz maximální frekvence procesoru
- VDD od 1,7 V do 3,6 V
- 512 kB Flash
- 128 kB SRAM
- GPIO (50) s možností externího přerušení
- 12bitový ADC se 16 kanály
- RTC
- Časovače (8)
- I2C (3)

- USART (3)
- SPI (5)
- USB OTG Full Speed
- SDIO

Vlastnosti vývojové desky Nucleo:

- Dva typy doplňkových zdrojů
    - Připojení Arduino Uno Revision 3
    - STMicroelectronics Morpho prodlužovací kolíkové konektory pro plný přístup ke všem STM32 I / O
- Integrovaný debugger / programátor ST-LINK / V2-1 s konektorem SWD
    - Přepínač režimu výběru pro použití soupravy jako samostatného ST-LINK / V2-1
- Flexibilní napájení desky
    - USB VBUS nebo externí zdroj (3,3 V, 5 V, 7 - 12 V)
    - Přístupový bod pro správu napájení
- USER LED
- Dvě tlačítka: USER a RESET
- Možnost opětovného výčtu USB: tři různá rozhraní podporovaná na USB
    - Virtuální Com port
    - Velkokapacitní paměť (USB disk) pro programování drag'n'drop
    - Debug port [9]

Dále se budeme bavit pouze o periferiích a registrech, které byly v práci použity. [11]

3.1.1 Reset and clock control (RCC)

Periférie obstarává povolování/resetování periferií, ovládání hodinových signálů a rozvod hodin na procesoru. Tato periférie umožňuje pomocí několika registrů nastavit například frekvenci procesoru, nebo povolit různé periférie jako USART, GPIO a další.

Než se budeme bavit o registrech RCC je třeba si říct něco o zdrojích hodin.

Fázový závěs (PLL) – zpětnovazební obvod navržený tak, aby umožňoval jedné desce s plošnými spoji synchronizovat fázi jejích palubních hodin s externím časovacím signálem.

Interní krystal (HSI) – hodinový signál je generován z interního oscilátoru a lze jej použít přímo jako systémové hodiny nebo jako vstup PLL.

Externí krystal (HSE) - hodinový signál je generován z externího oscilátoru a lze jej použít stejně jako HSI. Oproti HSI dokáže dodávat velmi přesnou frekvenci.

3.1.1.1 Registr RCC_CR

32bitový registr, který slouží pro ovládání hodin. V práci byly použity tyto bity:

Bit 24 PLLON – slouží k zapnutí (1) nebo vypnutí (0) PLL

Bit 16 HSEON – slouží k zapnutí (1) nebo vypnutí (0) HSE

Bit 0 HSION – slouží k zapnutí (1) nebo vypnutí (0) HSI

3.1.1.2 Registr RCC_PLLCFGR

Registr RCC_PLLCFGR je 32bitový registr, který se používá pro nastavení PLL podle těchto rovnic:

$$f_{(\text{VCO clock})} = f_{(\text{PLL clock input})} \cdot \frac{PLLN}{PLLM}$$

$$f_{(\text{PLL general clock output})} = \frac{f_{(\text{VCO clock})}}{PLLP}$$

$$f_{(\text{USB OTG FS,SDIO,RNG clock output})} = \frac{f_{(\text{VCO clock})}}{PLLQ}$$

Použité bity:

Bit 22 PLLSRC – nastavuje zdroj hodin pro PLL na HSI (0) nebo HSE (1)

Bit 16-17 PLLP – nastaví hodnotu PLLP na 2 (00), 4 (01), 6 (10), 8 (11)

Bit 6-14 PLLN – binární číslo určuje hodnotu PLLN. PLLN může nabývat hodnot od 50 do 432 ostatní hodnoty jsou chybné.

Bit 0-5 PLLM – binární číslo určuje hodnotu PLLM. Hodnota může nabývat pouze hodnot od 2 do 63, ostatní hodnoty jsou chybné.

3.1.1.3 Registr RCC_CFGR

Registr RCC_CFGR je 32bitový registr, který slouží pro nastavení cesty pro signál hodin. Použity byly tyto bity:

Bit 13-15 PPRE2 – nastavuje hodnotu vysokorychlostní před děličky APB2. Dělička může nabývat těchto hodnot: 1 (0xx), 2 (100), 4 (101), 8 (110), 16 (111)

Bit 10-12 PPRE1 – nastavuje hodnotu nízkorychlostní před děličky APB1. Dělička může nabývat těchto hodnot: 1 (0xx), 2 (100), 4 (101), 8 (110), 16 (111)

Bit 4-7 HPRE – nastavuje hodnotu před děličky AHB. Dělička může nabývat těchto hodnot: 1 (0xxx), 2 (1000), 4 (1001), 8 (1010), 16 (1011), 64 (1100), 128 (1101), 256(1110), 512 (1111).

Bit 2-3 SWS – udává stav spínače systémových hodin. Možné stavy: HSI (00), HSE (01), PLL (10), nepoužitý (11).

Bit 0-1 SW – nastavuje stav spínače systémových hodin. Možné stavy: HSI (00), HSE (01), PLL (10), nepovolený (11).

3.1.1.4 Registry RCC_xxxxRSTR a RCC_xxxxENR

RCC_xxxxRSTR je 32bitový registr, který se používá pro resetování periferií (místo "xxxx" je daná periférie, jako např AHBx, APBx,…). RCC_xxxxENR je 32bitový registr, který se používá pro povolování periferií. V práci byly použity periférie AHB1 a APB1.

Pod AHB1 se nachází periférie DMA, CRC a GPIO a Pod APB 1 jsou periférie PWR, I2C, USART, SPI, WWDG a TIM2-TIM5. O použitých periferiích si něco řekneme později.

3.1.2 General-purpose I/Os (GPIO)

Název general-purpose (česky univerzální) vystihuje význam této periférie. Umožnuje nastavení vstupů/výstupů dle různých požadavků. Je možné piny nastavit jako vstupy a získávat z nich data (snímání senzorů, tlačítek, …), nebo naopak můžeme nastavit piny jako výstupy, do kterých data posíláme (ovládání displeje, LED diody, …).

Vývojová deska umožňuje použít šest GPIOx, a to A, B, C, D, E, H, kdy každý port lze nastavit pomocí čtyř 32bitových konfiguračních registrů (MODER, OTYPER, OSPEEDER, PUPDR), dvou 32bitových datových registrů (IDR, ODR), jednoho 32bitového registru set/reset (BSRR),

jednoho 32bitového uzamykacího registru (LCKR) a dvou 32bitových registrů pro výběr alternativních funkcí (AFRL, AFRH).

3.1.2.1 Konfigurační registry

Registr GPIOx_MODER – registr nastavuje režim I/O. Má dva konfigurační bity a lze tak vybrat z režimů vstup (00), výstup (01), alternativní režim (10), analogový režim (11).

Registr GPIOx_OTYPER – registr nastavuje typ výstupu. Má pouze jeden konfigurační bit, proto lze nastavit pouze dva typy push-pull (0) a open-drain (1).

Registr GPIOx_OSPEEDR – registr nastavuje rychlost výstupu. Má dva konfigurační bity, se kterými lze nastavit rychlost low speed (00), medium speed (01), fast speed (10), high speed (11).

Registr GPIOx_PUPDR – registr nastavuje použití pull-up a pull-down rezistorů na I/O. Dvěma konfiguračními bity lze nastavit I/O bez pull-up, pull-down (00), s pull-up (01), s pull-down (10), reserved (11).

3.1.2.2 Datové registry

Registr GPIOx_IDR – registr lze použít pouze pro čtení hodnot vstupů.

Registr GPIOx_ODR – registr, který nastavuje hodnotu na výstup.

3.1.2.3 Set/Reset registr

Registr GPIOx_BSRR – registr umožnuje nastavit hodnotu více vstupů najednou.

3.1.2.4 Uzamykací registr

Registr GPIOx_LCKR – registr umožňuje uzamčení konfigurace GPIO portu.

3.1.2.5 Registry pro výběr alternativní funkce

Registr GPIOx_AFRL/Registr GPIOx_AFRH – registry slouží k nastavení alternativní funkce I/O. V našem případě jsme použili alternativní funkci generování PWM.

3.1.3 Universal synchronous asynchronous receiver transmitter (USART)

Univerzální synchronní/asynchronní vysílač/přijímač nabízí flexibilní řešení plně duplexní výměny. USART nabízí velmi širokou škálu přenosových rychlostí pomocí generátoru zlomkové přenosové rychlosti (anglicky Baud rate). Podporuje synchronní jednosměrnou komunikaci a poloduplexní jednovodičovou komunikaci.

Periférie USART se dá tedy použít pro posílání dat do terminálu, kde můžeme číst různé informace o aktuálním stavu programu, nebo program přímo ovládat.

3.1.3.1 Registr USART_SR

Anglický název status register dostatečně vystihuje význam 32bitového registru a to, že se stará o signalizaci stavů periférie.

Použité bity:

Bit 7 TXE – Tento bit je nastaven hardwarem, v případě že byl přenesen obsah registru do TDR posuvného registru. Jednoduše řečeno se na bitu objeví hodnota 1 pokud byl proveden přenos dat.

Bit 5 RXNE – Tento bit je nastaven hardwarem, pokud byl přenesen obsah posuvného registru RDR do registru USART_DR. Jednoduše řečeno se na bitu objeví hodnota 1 v případě že byl proveden příjem dat.

3.1.3.2 Registr USART_DR

32bitový datový registr, který obsahuje na prvních osmi bitech přijatá nebo přenesená data, v závislosti na tom, zda je čteno z registru nebo psáno do registru. Tyto data byly použity pro komunikaci s terminálem.

3.1.3.3 Registr USART_BRR

32bitový registr slouží pro nastavení zlomkové přenosové rychlosti. V našem případě byla nastavena hodnota 38400.

3.1.3.4 Registry USART_CRx

Tři 32bitové registry (CR1, CR2, CR3), které slouží pro ovládání a povolování funkcí periférie USART. Registr CR1 byl použit pro povolení samotné periférie USART, povolení vysílače a povolení přijímače.

3.1.4 General-purpose timers (TIM2 to TIM5)

Periférie dvou 16bitových (TIM3 a TIM4) a dvou 32bitových (TIM2 a TIM5) univerzálních časovačů. Časovače mohou plnit funkci čítače, generátoru PWM nebo generování DMA.
V práci byl použit 16bitový časovač TIM3, který měl za úkol správně načasovat tok dat do RGB ledek. Pro tento účel se musely použít následující registry.

3.1.4.1 Registry TIM3_CRx

Tyto registry (CR1 a CR2) ovládají a povolují různé funkce a režimy časovače. Registr CR1 slouží pro nastavení a povolení režimu časovače jako čítače. Čítač může pracovat v režimu čítání nahoru, dolů nebo obousměrně. Registr CR2 slouží pro nastavení a povolení funkce DMA. //použití v práci//

3.1.4.2 Registr TIM3_DIER

Registr TIM3_DIER má za úkol povolování přerušení nebo DMA. V práci byl registr použit pro povolení přerušení.

3.1.4.3 Registr TIM3_SR

Stejně jako v předchozích případech se jedná o registr, který zaznamenává stavy periférie. V případě časovače TIM3 se zde zaznamenávají přerušení nebo přetečení čítače. V práci byl registr použit pro inicializaci přerušení.

3.1.4.4 Registry TIM3_CCMRx

Registry TIM3_CCMRx slouží pro nastavení režimu určitého kanálu časovače. Kanály lze použít na vstupu (režim snímání) nebo na výstupu (režim porovnání).

V práci byl použit registr CCMR1, který určuje chování referenčního signálu, ze kterého se například odvozují stavy signálů po přetečení časovače.

3.1.4.5 Registr TIM3_CCER

Registr povoluje režim snímání/porovnávání.

3.1.4.6 Registr TIM3_CNT

16bitový datový registr, který v sobě zaznamenává hodnotu časovače/čítače.

3.1.4.7 Registr TIM3_PSC

Registr nastavující předděličku časovače.

3.1.4.8 Registr TIM3_ARR

Registr nastavující hodnotu, ve které časovač přeteče (nastane akce) a čítá znova od nuly. Společně s TIM3_PSC slouží pro přesné nastavení frekvence časovače.

3.1.4.9 Registry TIM3_CCRx

Registry (CCR1, CCR2, CCR3, CCR4) slouží pro zaznamenávání hodnoty čítače, která byla přenesena při poslední události (při nastavení kanálu jako vstup), nebo pro vkládání hodnot, se kterými se následně pracuje (při nastavení kanálu jako výstup).

3.1.5 Serial peripheral interface (SPI)

SPI neboli periférie sériového rozhraní umožňuje poloduplexní/plně duplexní, synchronní, sériovou komunikaci s externími zařízeními.

Periférie má čtyři hlavní piny, a to MISO, MOSI, SCK, NSS

MISO (Master In / Slave Out) – pro vysílání dat v režimu “slave“ a pro příjmání dat v režimu “master“.

MOSI (Master Out / Slave In) – pro vysílání dat v režimu “master“ a pro přijímání v režimu “slave“.

SCK – Sériový výstup hodin pro SPI master a vstup pro SPI slave.

NSS – Výběr podřízeného zařízení. Tento pin slouží jako "chip select", aby SPI master mohl komunikovat s podřízenými zařízeními samostatně a aby se zabránilo soupeření na datových linkách.

3.2 RGB NeoPixelStick 8

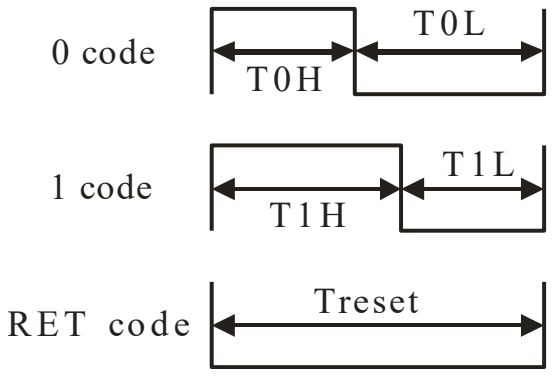
Jedná se o sadu osmi inteligentních RGB LED diod WS2812 od firmy Adafruit. Tyto diody jsou zvláštní hlavně díky vlastním ovládacím obvodům, díky kterým se dá celá sada diod řídit pouze jedním datovým vstupem. Tato vlastnost je při porovnání se staršími variantami RGB LED diod, kdy se musela každá dioda řídit zvlášť, velká výhoda. Na celé sadě se tedy nachází pouze čtyři vstupní a čtyři výstupní piny. Tři z těchto pinů jsou na obou stranách použité pro napájení (5VDC a dvakrát GND) a zbývající dva piny (Din, Dout) slouží k datovému přenosu.

Díky výstupním pinům dokonce můžeme zapojit do série více sad těchto diod, které lze stále ovládat pouze jedním datovým vstupem. Pokud bychom nebrali v potaz napájení, můžeme tak teoreticky ovládat neomezený počet těchto diod. [10]

3.2.1 Důležité parametry

Parametr	Značka	Hodnota	Jednotka
Napájecí napětí	V_{DD}	+3.5~+5.3	V
Vstupní napětí	V_I	-0.5～VDD+0.5	V
0 code ,high voltage time	T0H	0.4	μs
1 code ,high voltage time	T1H	0.8	μs
0 code , low voltage time	T0L	0.85	μs
1 code ,low voltage time	T1L	0.45	μs
1 code ,low voltage time	RES	>=50	μs

Tabulka 1. Důležité parametry WS2812 (převzato a upraveno z [10])



Obrázek 13. Časování logických hodnot a resetovacího kódu (převzato z [10])

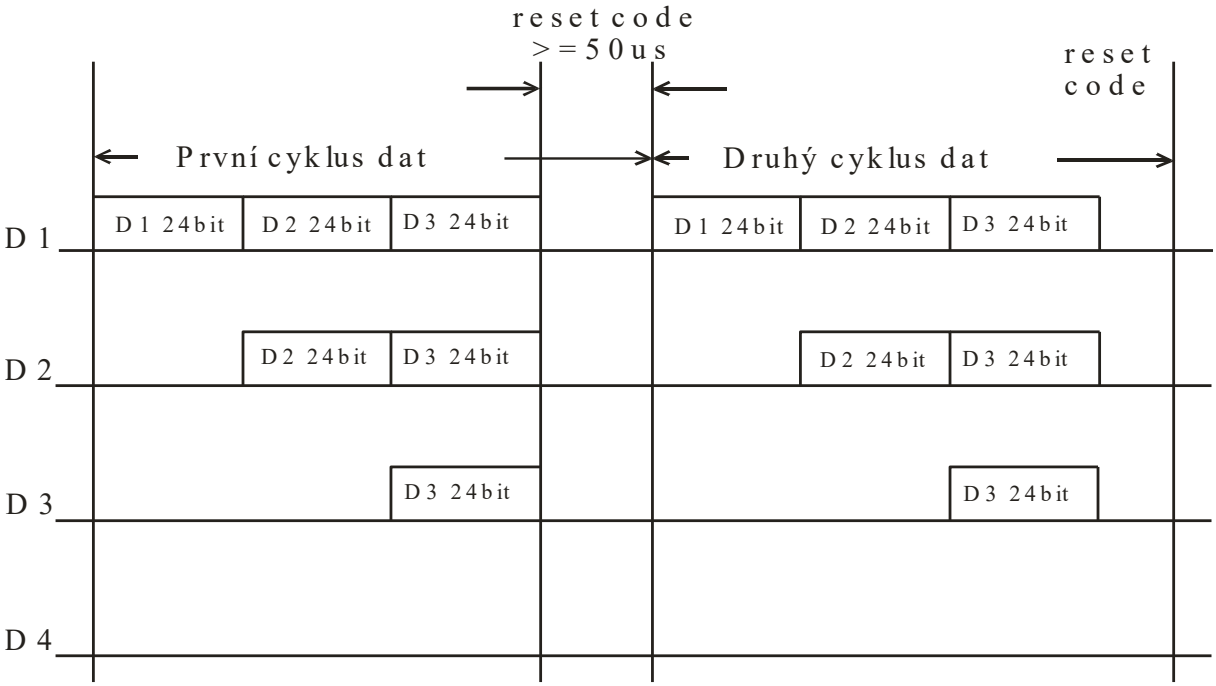
3.2.2 Ovládání

Jak již bylo psáno, veškeré ovládání diod jde pouze přes jednu cestu. Tato skvělá vlastnost je zároveň hlavním problémem. K ovládání těchto diod je totiž zapotřebí mít přesný mikroprocesor, který je schopný přesně načasovat tok dat.

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

Obrázek 14. Balík dat řídící RGB LED diody

Na obrázku (Obrázek 14) můžeme vidět balík dat jednoho cyklu. Data se posílají v balíku o velikosti 24 bitů, kdy každých 8 bitů je jedna složka RGB. Po odeslání těchto dat se musí odeslat "RESET" code a poté lze poslat další data.



Obrázek 15. Způsob posílání dat při propojení více sad RGB diod (převzato a upraveno z [10])

Při propojení více sad do série posíláme data také po sobě (Obrázek 15). Každá sada si přečte svůj balík dat a dále se již nedostane.

3.2.3 Realizace komunikace s WS2812

Pro komunikaci s WS2812 můžeme použít čtyři základní principy: manuální časování, přerušení časovače, SPI a časovač s PWM.

3.2.3.1 Manuální časování

Při této komunikaci využíváme manuálně nastavené časování (například pomocí systick, nebo zpožďovací smyčky), které nám zajistí dobu, po kterou vysíláme 1 (800ns pro stav "1" a 450ns pro stav "0") a po kterou 0 (400ns pro stav "1" a 850ns pro stav "0"). Pro posílání dat můžeme vytvořit funkce, které pomocí zpoždění pošlou balík dat pro chtěné barvy. Toto řešení je sice jednoduché, ale není vhodnou variantou pro zadanou práci. Hlavní nevýhodou je, že při vysílání dat není možné jinak pracovat s procesorem.

Příklad funkce pro červenou barvu:

```
void draw_red(void)
{
  char i;
    for(i = 0; i < 8; i++)                          //zelená = 0
    {
      GPIOWrite(GPIOB, 5, 1);
      Delay(450);
      GPIOWrite(GPIOB, 5, 0);
      Delay(850);
    }
    for(i = 0; i < 8; i++)                          //červená = 1
    {
      GPIOWrite(GPIOB, 5, 1);
      Delay(800);
      GPIOWrite(GPIOB, 5, 0);
      Delay(400);
    }
    for(i = 0; i < 8; i++)                          //modrá = 0
    {
      GPIOWrite(GPIOB, 5, 1);
      Delay(450);
      GPIOWrite(GPIOB, 5, 0);
      Delay(850);
    }
  Delay(50000);           //Reset code
}
```

3.2.3.2 Přerušení časovače

V této metodě se využívá k časování pouze vhodně nastavený časovač pomocí PSC a ARR. Hlavní problém je zde s rychlostí procesoru. V mém případě, kdy byla použita vývojová deska Nucleo s procesorem STM32F411RE, je minimální perioda časovače několik mikro sekund. Pro možnost použití této metody musí časovač být schopný zvládnout periodu v řádech nano sekund. Tato možnost tedy také není vhodným řešením pro naše účely.

3.2.3.3 SPI

Nyní už se dostáváme k vhodnějším řešením. Využijeme zde SPI pin MOSI. Je zde potřeba nastavit několik hodnot pro SPI. Potřebujeme nastavit SPI jako jednosměrný vysílač. Dále nastavíme formát datového rámce na 8 bit. Jako další nastavíme možnost softwarově ovládat pin NSS, protože naše sada LED nemá vstup pro řízení této hodnoty. Potřeba je také nastavit hodnoty, které bude pin MOSI generovat. 1 bit ledek budeme generovat pomocí čtyř bitů. Stav "0" představuje hodnota 1000 a stav "1" hodnota 1100. Tyto hodnoty zaberou tedy dva bity v SPI bajtu. Jako poslední je třeba nastavit hodnotu baud rate, která udává kolik bitů za sekundu je vysíláno. Hodnota baud rate je tedy třeba vypočítat:

$$Baud\ rate = f \cdot bity\ v\ SPI\ bajtu = 800000 \cdot 2 = 1{,}6Mbit/s$$

Frekvence f je potřebná frekvence naší sady. Tato frekvence se vypočítá z periody jednoho signálu neboli TH + T0 = 1.25µs ± 600ns (800kHz). Při nastavení hlavních hodin na 51,2 MHz nám na vstup clk SPI (SCK) jde frekvence 25,6 MHz a je tedy třeba hodnotu vydělit šestnácti. Toho docílíme pomocí předděličky, na které nastavíme číslo 16.

3.2.3.4 Časovač s PWM

V této metodě používáme k vysílání dat PWM řízený časovačem. Musíme nastavit stejnou periodu jako u předchozí metody, a to 800kHz neboli 1.25µs. Dále musíme nastavit hodnoty PWM pro stavy vycházející z datasheetu. Stav "1" je tedy 67 % a stav "0" 33 %. Po každém přetečení časovače se vyvolá přerušení, kde se nastaví nová hodnota PWM registru (do registru CCR2). Před každou LED (24 bit) musí být poslán stav "0" po dobu alespoň čtyřiceti period (Reset code = 50 µs). V práci byla zvolena tato metoda.

4 Simulace algoritmu na PC

V následující kapitole ukážu použité vývojové prostředí a uvedu základní popis programu napsaného v jazyce C# a návod k jeho použití. Dále zde uvedu příklady třídících algoritmů.

4.1 Vývojové prostředí Microsoft Visual Studio

Microsoft Visual Studio je integrované vývojové prostředí (IDE) od společnosti Microsoft. Používá se k vývoji počítačových programů pro Microsoft Windows a také pro webové stránky, webové aplikace, webové služby a mobilní aplikace. Visual Studio používá vývojové platformy softwaru Microsoft, jako jsou Windows API, Windows Forms, Windows Presentation Foundation, Windows Store a Microsoft Silverlight.[12]

Visual Studio nabízí výkonné editory HTML, CSS, JavaScript a JSON, LESS, SASS, PHP, Visual Basic, C++, F#, Python nebo jazyk C#.[13]

Dále se bude psát pouze o jazyku C# s Windows Forms, ve kterém je program napsán.

4.1.1 Windows Forms

Jedná se o framework, který usnadňuje tvorbu formulářových aplikací s pomocí grafického designeru. Nachází se zde plná sada ovládacích prvků pro usnadnění tvorby aplikací. Nachází se zde i možnost vytvořit vlastní nebo upravit nějakou existující.[14]

Windows form byly použity pro vizualizaci a ovládání simulace třídících algoritmů na počítači.

4.1.2 Programovací jazyk C#

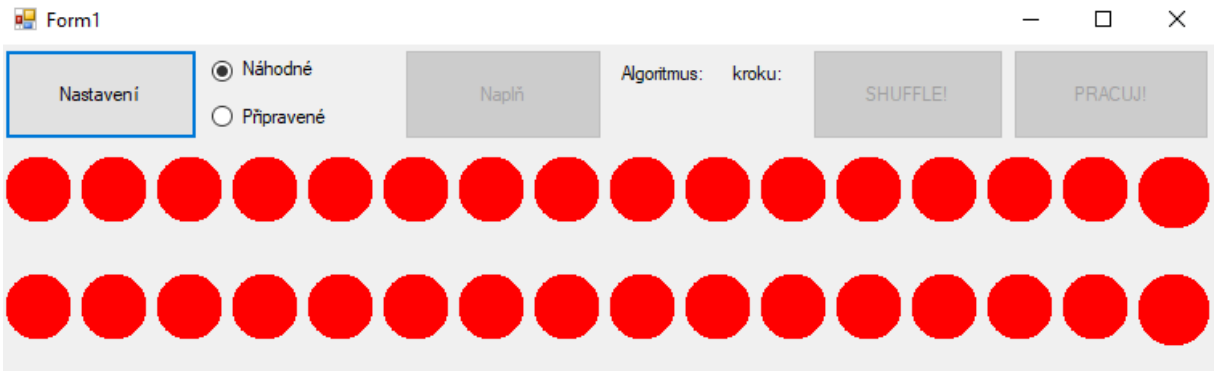
Jazyk C# je vyvinut firmou Microsoft. Tento jazyk vychází v mnohém z programovacího jazyka C/C++, ale v mnoha ohledech se více podobá programovacímu jazyku Java.

Základní charakteristika:

- čistě objektově orientovaný jazyk
- v programu není žádný hlavní cyklus, ale pracuje na základě událostí
- podobně jako Java obsahuje pouze jednoduchou dědičnost s možností násobné implementace rozhraní
- zajišťuje typovou bezpečnost
- podporuje zpracování chyb pomocí výjimky. [15]

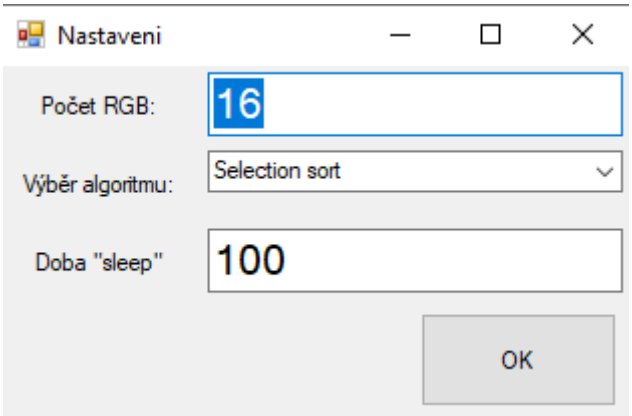
4.2 Popis programu

Při zpuštění programu se objeví hlavní okno (Obrázek 16), kde máte možnost kliknout pouze na “Nastavení” a zvolit si pole hodnot (“Náhodné”, “Připravené”). Na spodní části se nachází dvě řady kruhů, které simulují RGB diody (dále jen RGB). Vrchní řádka představuje aktuální tvar pole a spodní řádka představuje chtěný výsledek.



Obrázek 16. Hlavní okno simulace na PC

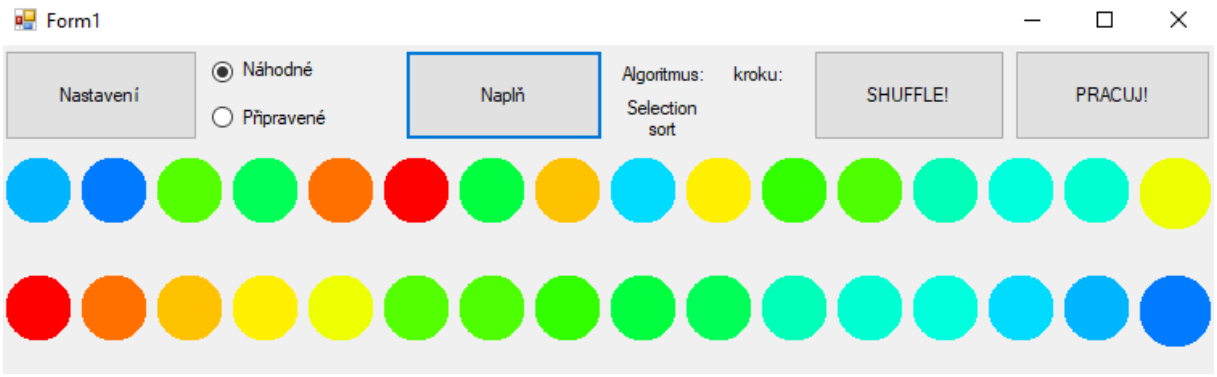
Při kliknutí na tlačítko “Nastavení” se nám objeví další okno (Obrázek 17) ve kterým lze nastavit libovolný počet RGB, neboli počet hodnot v poli pro třídění. Dále si zde můžeme vybrat z pěti algoritmů a nastavit dobu “sleep”, což je vlastně doba, po kterou je zobrazený jeden krok.



Obrázek 17. Vyskakovací okno "Nastavení" simulace na PC

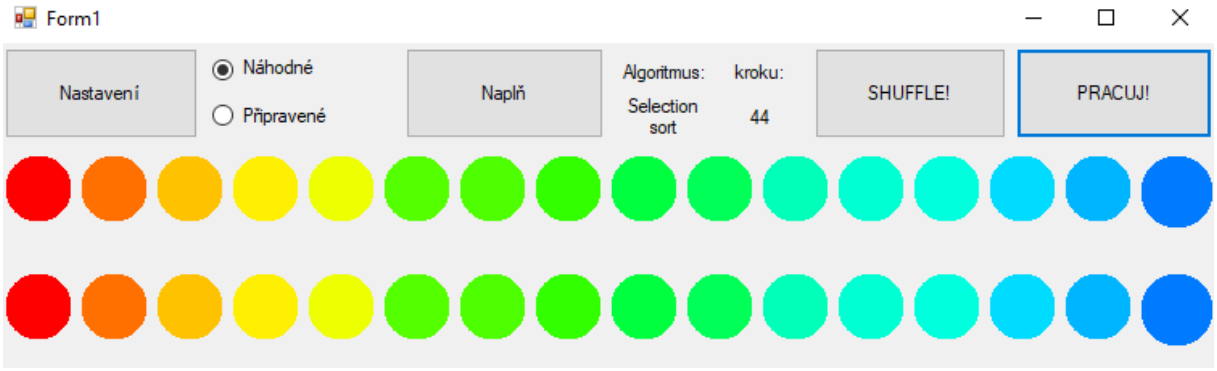
Po vyplnění našich požadavků stiskneme tlačítko OK a poté se okno s nastavením zavře. Na hlavním okně se nám otevřou možnosti kliknout na tlačítka “Naplň”, “SHUFFLE”, “PRACUJ!” a vypíše se zvolený algoritmus. Tlačítka “SHUFFLE” a “PRACUJ!” jsou v tuhle chvíli prakticky nevýznamná, a proto si jejich funkci rozepíšeme déle. Po stisknutí tlačítka “Naplň” se nám naplní pole hodnot dle našich požadavku z nastavení a zvoleného režimu vedle tlačítka.

Pole hodnot se také zobrazí na vrchní řádce RGB a na spodní řádce RGB se zobrazí chtěný výsledek (Obrázek 18).



Obrázek 18. Hlavní okno simulace po naplnění pole

Dále nás již budou zajímat už zmiňovaná tlačítka “SHUFFLE” a “PRACUJ!”. Tlačítko “SHUFFLE”, jak název napovídá, slouží pro zamíchaní pole hodnot. Po stisku tlačítka se tedy zamíchají barvy na vrchní řádce RGB. Při stisknutí tlačítka “PRACUJ!” se spustí třídící algoritmus na pole a roztřídí postupně vrchní řadu RGB do požadované podoby (Obrázek 19).



Obrázek 19. Hlavní okno simulace po roztřídění pole

V celém průběhu je možné nastavovat algoritmus nebo naplnit pole hodnot dle nových požadavků.

4.3 Příklady základních koster použitých algoritmů

SelectionSort

```
int Temp = 0;
for (int i = 0; i < pocetRGB - 1; i++)
{
  pozice = pocetRGB - 1;
  for (int j = i; j < pocetRGB - 1; j++)
     if (pole[pozice] > pole[j])          // hledani minima
     {
        pozice = j;
        cntKroky++;
     }
  // prohozeni prvku
  temp = pole[pozice];
  pole[pozice] = pole[i];
  pole[i] = temp;
}
```

BubbleSort

```
int pozice = pocetRGB - 2;
// kontrola prohozeni
bool swapped = true;
while (swapped)
{
  swapped = false;
  for (int i = 0; i <= pozice; i++)
  {
    cntKroky++;
    // prohozeni
    if (pole[i] > pole[i + 1])
    {
      temp = pole[i];
      pole[i] = pole[i + 1];
      pole[i + 1] = temp;
      swapped = true;
    }
  }
  pozice--;
}
```

InesrtionSort

```
int item;
int cntKroky = 0;
int pozice = 0;
for (int i = 1; i <= (pocetRGB - 1); i++)
{
   // ulozeni prvku
   item = pole[i];
   pozice = i - 1;
   while ((pozice >= 0) && (pole[pozice] > item))
   {
      pole[pozice + 1] = pole[pozice];
      cntKroky++;
      pozice--;
   }
   pole[pozice + 1] = item;
   cntKroky++;
}
```

HeapSort

```
heapify(list,kroky,vysledek,sleep,tbl);  //tvorba haldy z pole
int index = list.Length - 1;             //posledni prvek
int temp;
while (index > 0)
{
  temp = list[0];                        // prohození posledního prvku s maximem
  list[0] = list[index];
  list[index] = temp;
  index -= 1;                            //nastaveni noveho posledniho prvku
  down(list, index, kroky, vysledek, sleep, tbl);
}
```

QuickSort

```
void limited_quicksort(int[] list, int left, int right, int sleep, TableLayoutPanel
tbl, bool vysledek, int kroky)
{
  if (right >= left)
  { // podminka rekurze
    int pivot = left; // vyber pivotu
    int[] div = new int[2];
    div = divide(list, left, right, pivot, sleep, tbl, vysledek, kroky);
    int new_pivot = div[0];
    kroky = div[1];
    // rekurzivni zavolani na obe casti pole
    limited_quicksort(list, left, new_pivot - 1, sleep, tbl, vysledek, kroky);
    limited_quicksort(list, new_pivot + 1, right, sleep, tbl, vysledek, kroky);
  }
}
```

5 SOFTWARE

Stejně jako v části o simulaci na PC si v následující kapitole napíšeme o vývojovém prostředí Atollic TrueSTUDIO for STM32, popíšeme ovládání programu a uvedeme si příklady stejných algoritmů upravených pro menší zátěž procesoru.

5.1 Vývojové prostředí Atollic TrueSTUDIO for STM32

Atollic TrueSTUDIO je komerčně vylepšené C / C ++ IDE postavené na Eclipse®, CDT ™, GCC a GDB. Tento nástroj poskytuje vývojářům výkonná profesionální rozšíření, funkce a nástroje pro snadný a efektivní vývojový proces.

Vývoj Atollic TrueSTUDIO je od roku 2017 ukončen, ale všechny jeho hlavní funkce jsou zdarma zahrnuty ve "vše v jednom" vývojovém STM32CubeIDE.

Hlavní funkce:

- analyzátor poruch CPU
- živé sledování globálních proměnných
- pokročilé možnosti sledování v reálném čase
- multi-jádrové a multi-deskové ladění. [16]

5.1.1 Eclipse

V kontextu výpočetní techniky je Eclipse integrovaným vývojovým prostředím (IDE) pro vývoj aplikací využívajících programovací jazyk Java a další programovací jazyky jako jsou C/C ++, C#, Python, PERL, Ruby atd.

Integrovaná vývojová prostředí se vyznačují tím, že napomáhají svými funkcemi ke snadnější práci při vývoji. Tato vývojová prostředí mohou mít například našeptávač, který vám pomáhá při psaní programu, předpřipravené funkce a jiné výhody. [17]

5.1.2 GCC

GNU Compiler Collection (GCC) je jedním z nejsložitějších softwarových systémů dostupných v plné zdrojové formě. Byl vyvinut společností Free Software Foundation a dnes je řízen nezávislým řídícím výborem. Zatímco zpočátku to začalo jako efektivní kompilátor jazyka C

pro 32bitové stroje, vyvinulo se to v rozumnou obecnou architekturu, která oficiálně pojme asi 7 zdrojových jazyků.

Úlohou GCC je kontrolovat správnou syntaxi algoritmu, kontrolovat různé chyby jako nepřiřazená proměnná, nepoužití středníku a další. [18]

5.2 Popis programu

Pro úplný využití programu je potřeba mít na PC, přes který se program nahrává, nějakou formu terminálu s podporou sériové komunikace. Na terminálu je třeba nastavit Baud rate na 38400 a zvolit COM port, na kterém máme připojenou vývojovou desku.

Při spuštění programu se do terminálu vypíše návod k ovládání pomocí klávesnice a také pomocí tlačítka na desce (Obrázek 20).

```
.............nabídka možností.............
' ' - zahájit/pozastavit třídění
'p' - nastavit do pole hodnot hodnoty 105,20,230,48,75,141,25,15
'r' - nastavit do pole hodnot náhodné hodnoty
'c' - nastavit režim cyklus
'j' - změnit jas RGB
'a' - změna algoritmu
'+' - zvětšit čekací dobu mezi kroky o 10
'-' - snížit čekací dobu mezi kroky o 10
'x' - přepnout režim palety barev HSL/RGB
'R' - zapnout R složku RGB palety
'G' - zapnout G složku RGB palety
'B' - zapnout B složku RGB palety
.........................................

.........ovládání pomocí tlačítka.........
stisknutí - změna algoritmu
stisknutí na 200ms - změna čekací doby mezi kroky na 100, 500, 1000, 2000
stisknutí na 2s - zahájit/pozastavit třídění
.........................................
```

Obrázek 20. Výpis z terminálu po spuštění programu

Některé z těchto ovládacích prvků můžeme použít kdykoli, a některé jdou použít pouze při zastavení třídění. Zastavit a spustit třídění lze pomocí podržení tlačítka na desce na 2 sekundy, nebo stiskem mezerníku.

Ovládací prvky, které lze nastavit jen když je program zastaven:

Vkládání hodnot pro třídění – hodnoty můžeme vkládat dvěma způsoby. Můžeme třídit přednastavené pole hodnot, které se nastaví při stisku klávesy ‘p’, nebo můžeme nechat program vygenerovat náhodné pole hodnot stiskem klávesy ‘r’.

Režim cyklus – stisknutím tlačítka ‘c‘ povolíme nebo zakážeme režim cyklus. Tento režim od spuštění třídí náhodné pole hodnot vždy jiným algoritmem do té doby, než cyklus manuálně zastavíme. Pokud je režim cyklus vypnutý a spustíme třídění, tak program provede třídění dle požadavků, které se zvolily před spuštěním. Po dokončení třídění vypíše výsledek a sám se zastaví.

Změna algoritmu – pro volbu algoritmu stiskneme klávesu ‘a‘, nebo stiskneme krátce tlačítko na desce. Po stisku klávesy nebo tlačítka se přepne na následující algoritmus v pořadí SelectionSort, BubbleSort, InesrtionSort, HeapSort, QuickSort.

Ovládací prvky, které lze nastavit kdykoli:

Změna jasu – při stisku tlačítka ‘j‘ se zvětší nebo zmenší jas RGB diod.

Změna čekací doby mezi kroky – změnu čekací doby můžeme provést dvěma způsoby. Jedna z možností je stisknutí tlačítka na 200ms (při podržení déle jak 2s má tlačítko spínací/zastavovací funkci, proto je potřeba tlačítko nedržet zbytečně dlouho), kdy se čekací doba vybere z přednastavených hodnot 100, 500, 1000, 2000 ms na krok. Druhá možnost je pomocí tlačítek ‘+‘ a ‘-‘, kdy se čekací doba zvýší (+) nebo sníží (-) o 50ms. Maximální povolená rychlost je 50ms na krok.

Změna barev – můžeme vybrat ze dvou základních palet barev, a to RGB a HSL. Volbu palety provádíme tlačítkem ‘x‘. Dále můžeme tlačítky ‘R‘, ‘G‘, ‘B‘ zapnout nebo vypnout složku RGB.

5.3 Příklady základních koster použitých algoritmů

Oproti simulaci na PC zde kroky nejsou prováděny ve for cyklu, ale pomocí časovače, který přičte krok po nastaveném intervalu. Umožnuje nám to tak ovládat program v průběhu třídění.

```
if (CUR_TICKS > tmSleep)
{
  tmSleep = CUR_TICKS + intervalSleep;

  x++;
}
```

SelectionSort

```
if (firstX)
{
  x = 0;
  firstX = false;
}
temp = 0;
pozice = numPixels - 1;
for (int j = x; j < numPixels - 1; j++)
  if (pole[pozice] > pole[j])          // hledani minima
  {
    pozice = j;
  }                                    // prohozeni prvku
temp = pole[pozice];
pole[pozice] = pole[x];
pole[x] = temp;
```

BubbleSort

```
if (firstX)
{
  x = 0;
  xBubble = 0;
  firstX = false;
  swapped = false;
}
pozice = numPixels - 2;
temp = 0;
bool konec = false; // kontrola prohozeni
if (pole[x] > pole[x + 1]) // prohozeni
{
  temp = pole[x];
  pole[x] = pole[x + 1];
  pole[x + 1] = temp;
  swapped = true;
  lastx = x;
}
if (x > pozice && !swapped)
  konec = true;
if (x > pozice)
{
  x = 0;
  swapped = false;
}
pozice--;
```

Při porovnání s algoritmy ze simulace jde vidět, že krom hlavních cyklů se vlastně nic nezměnilo. Jediná velká změna byla potřeba u algoritmu QuickSort, kde bylo třeba napsat algoritmus bez rekurze kvůli časování kroků.

QuickSort

```
if (z >= 0)
{
  if(!pivotNastaven)
  {
    pivot = pole[L];
    pivotNastaven = true;
  }
  while (pole[R] >= pivot && L < R)
    R--;
  if (L < R)
  {
    pole[L++] = pole[R];
  }
  while (pole[L] <= pivot && L < R)
    L++;
  if (L < R)
  {
    pole[R--] = pole[L];
  }
  if (L >= R)
  {
    pole[L] = pivot;
    beg[z + 1] = L + 1;
    end[z + 1] = end[z];
    end[z++] = L;
    pivotNastaven = false;
  }
}
else
  z--
```

Závěr

Cílem této práce bylo vybrat si určité algoritmy, nastudovat jejich principy a následně je předvést pomocí simulace na PC a vizualizací pomocí RGB LED diod, které byly ovládány procesorem.

Po rešerši různých základních třídících algoritmů a jejich zdrojových kódů jsem zvolil algoritmy SelectionSort, BubbleSort, InsertionSort, HeapSort a QuickSort. U těchto algoritmů byly v teoretické části vysvětleny základní principy a vlastnosti. V praktické části poté byly předvedeny jejich základní kostry, které byly použity v obou programech.

Všech cílů této práce bylo dosaženo. Na obou programech můžeme vidět rozdíly v rychlosti vybraných algoritmů a lze říct, že BubbleSort je znatelně nejpomalejší, naopak pokročilejší algoritmy HeapSort a QuickSort jsou nejrychlejší.

Simulace na PC nebyla brána jako hlavní priorita, a tak výsledek není v porovnání s programem ovládající RGB příliš domyšlený. Mohlo by zde být například použito stejného principu krokování jako u programu pro procesor a použít místo zpoždění časovač. Z důvodu použití zpoždění není možné program ovládat v průběhu třídění.

Přestože na simulaci chybí některé funkce, které byly použity v programu, který řídil procesor, byla simulace výborná příprava na program pro procesor a mohli v ní být vyzkoušeny funkce jako zobrazování pomocí palety HSL, implementaci samotných řídicích algoritmů a ovládání celého programu.

Program pro ovládání RGB LED diod je vyzkoušen na sadě šestnácti LED a teoreticky by měl být schopen fungovat na libovolném počtu.

Oba programy by se daly dále zlepšovat. Mohly by být například přidány další třídicí algoritmy, možnost zadávat vlastní pole hodnot nebo možnost krokovat si průběh například stiskem tlačítka.

Seznam literatury a informačních zdrojů

[1] TÖPFER, Pavel. *Algoritmy a programovací techniky*. Druhé vydání. Praha: Spoločnosť Prometheus, 2007. ISBN 8071963509.

[2] ČÁPKA, David. *Úvod do teorie algoritmů* [online]. 2013 [cit. 2021-03-11]. Dostupné z: https://www.itnetwork.cz/navrh/algoritmy/teorie/uvod-do-teorie-algoritmu-definice-casova-slozitost-stabilita

[3] ŽOLTÁ, Lucie. *Složitost algoritmů* [online]. [cit. 2021-03-12]. Dostupné z: http://lucie.zolta.cz/index.php/zaklady-teoreticke-informatiky/15-turingovy-a-ram-stroje/15-slozitost-algoritmu

[4] TESAŘ, Karel a Martin MAREŠ. *Složitost* [online]. 2012 [cit. 2021-03-12]. Dostupné z: https://ksp.mff.cuni.cz/kucharky/slozitost/

[5] ČÁPKA, David. *Třídicí/řadicí algoritmy* [online]. 2013 [cit. 2021-03-11]. Dostupné z: https://www.itnetwork.cz/navrh/algoritmy/algoritmy-razeni

[6] BAYER, Tomáš. *Třídící algoritmy* [online]. In: . 2007 [cit. 2021-03-12]. Dostupné z: https://web.natur.cuni.cz/~bayertom/images/courses/Prog2/programovani_trideni.pdf

[7] VALLA, Tomáš, Martin MAREŠ a Dan KRÁL. *Třídění* [online]. 2011 [cit. 2021-03-11]. Dostupné z: http://ksp.mff.cuni.cz/kucharky/trideni/

[8] *Halda* [online]. In: 2017, s.1-2 [cit. 2021-5-7]. Dostupné z: https://web.archive.org/web/20170610080655/http://kam.mff.cuni.cz/~kuba/ka/halda.pdf

[9] *NUCLEO-F411RE* [online]. Cambridge (Massachuttes): Global Headquarters, 2021 [cit. 2021-4-15]. Dostupné z: https://os.mbed.com/platforms/ST-Nucleo-F411RE/

[10] *WS2812B: Intelligent control LED integrated light source* [online]. In: . [cit. 2021-5-7]. Dostupné z: https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf

[11] *STM32F411xC/E Reference manual* [online]. [cit. 2021-5-13]. Dostupné z: https://www.st.com/resource/en/reference_manual/dm00119316-stm32f411xc-e-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

[12] Visual Studio. *Baylor University* [online]. Waco, Texas [cit. 2021-5-16]. Dostupné z: https://www.baylor.edu/its/index.php?id=941583

[13] Web languages. *Microsoft: Visual Studio* [online]. [cit. 2021-5-1]. Dostupné z: https://visualstudio.microsoft.com/cs/vs/features/web/languages/

[14] ČÁPKA, David. Úvod do Windows Forms aplikací. *ITnetwork* [online]. 2013 [cit. 2021-5-1]. Dostupné z: https://www.itnetwork.cz/csharp/winforms/c-sharp-tutorial-windows-forms-okenni-aplikace-uvod

[15] BĚHÁLEK, Marek. *Základní charakteristika jazyka C#* [online]. 2007 [cit. 2021-5-1]. Dostupné z: http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text/ch02.html

[16] *TrueSTUDIO* [online]. STMicroelectronics [cit. 2021-5-1]. Dostupné z: https://www.st.com/en/development-tools/truestudio.html

[17] *Eclipse: Overview* [online]. tutorialspoint [cit. 2021-5-5]. Dostupné z: https://www.tutorialspoint.com/eclipse/eclipse_overview.htm

[18] *Basic Information about GCC* [online]. [cit. 2021-5-5]. Dostupné z: https://www.cse.iitb.ac.in/grc/intdocs/gcc-basic-info.html