VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ŘÍDICÍ SYSTÉM MATICOVÉ ZOBRAZOVACÍ JEDNOTKY S RASPBERRY PI
CONTROL SYSTÉM OF MATRIX DISPLAY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Jan Benda**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Zdeněk Bradáč, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: Jan Benda ***ID:*** 195270
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Řídicí systém maticové zobrazovací jednotky s Raspberry Pi

POKYNY PRO VYPRACOVÁNÍ:

1. Proveďte literární rešerši informačních panelů.
2. Nastudujte funkci stávajícího displeje BUSE, nastudujte a oživte stávající elektroniku autonomního informačního portálu, který umožní zobrazovat data a animace na displeji BUSE.
3. Využijte zabudovaný systém na základě Raspberry PI. Vytvořte programové vybavení, které umožní návrhy obrazovek pro zobrazovací jednotku a zobrazení dat.
4. Ověřte funkčnost a demonstrujte ji.

DOPORUČENÁ LITERATURA:

Pavel Herout: Učebnice jazyka C, KOPP, 2004, IV. přepracované vydání, ISBN 80-7232-220-6

Dle pokynů vedoucího práce.

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 20.5.2019

Vedoucí práce: doc. Ing. Zdeněk Bradáč, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

Abstrakt

Obsahem této bakalářské práce je návrh a realizace programového vybavení pro maticovou zobrazovací jednotku BUSE na základě Raspberry Pi. Maticový displej byl nejprve oživen, neboť při pokusu o zprovoznění vykazoval známky nefunkčnosti. Došlo k výměně několika elektronických součástek. Řídicí systém umožňuje návrhy obrazovek a animací pro maticovou jednotku a jejich následné vykreslení na displej. Návrh obrazovek je umožněn přes grafický prvek bitmapy. V této práci byly vytvořeny celkem tři programy. Řídicí program na PC, program na mikrokontrolér ATmega128A, který z přijatých dat vykresluje obrazovky na displej a program do mikropočítače Raspberry Pi, který se stará o přeposílání dat mezi mikrokontrolérem a PC.

Klíčová slova

Maticová zobrazovací jednotka, řídicí systém, Raspberry Pi, UART, bitmapa

Abstract

The content of this thesis is to design and implement control system for a matrix display BUSE using Raspberry Pi. At first the matrix display was revived because after several attemps of commissioning it seemed to be broken and there were some electrical components that needed to be replaced. The control system allows to design screens and animations for matrix display through graphical element called bitmap. The thesis contains creation of three programs. The control program running on PC, program for microcontroler ATmega128A which displays screens on BUSE panel and program for Raspberry PI which resends data between microcontroller and computer.

Keywords

Matrix display, Control system, Raspberry Pi, UART, bitmap

Bibliografická citace:

BENDA, Jan. *Řídicí systém maticové zobrazovací jednotky s Raspberry Pi* [online]. Brno, 2019 [cit. 2019-05-11]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/119248. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Zdeněk Bradáč.

Prohlášení autora o původnosti díla

„Prohlašuji, že svou bakalářskou práci na téma Řídicí systém maticové zobrazovací jednotky s Raspberry Pi jsem vypracoval samostatně pod vedením vedoucí/ho bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské 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 části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.“

.

V Brně dne: **20. května 2019** ……

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce doc. Ing. Zdeňku Bradáčovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne: **20. května 2019** ……

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 ÚVOD

Cílem závěrečné práce je návrh a realizace programového vybavení, které umožní zobrazování dat a animací na panelu BUSE s využitím zabudovaného systému na základě mikropočítače Raspberry Pi.

Nejprve je provedena literární rešerše obdobných panelů. Panely jsou rozděleny podle technologie zobrazování informace. V rešerši jsou jednotlivé panely vyjmenovány a je popsán princip jejich funkce. V závěrečné práci je využit terčíkový displeje, proto je na tento panel v rešerši zaměřena největší pozornost.

Dalším bodem zadání je průzkum stávající elektroniky panelu. Nejprve jsou elektronické prvky vyjmenovány, poté je každý z prvků detailněji popsán. Mezi dva hlavní elektronické prvky se řadí mikropočítač Raspberry Pi a mikrokontrolér ATmega128A. Tyto prvky komunikují přes rozhraní UART, proto je také popsán princip této komunikace.

Dalším úkolem je oživení panelu a ověření jeho funkčnosti. Tento bod zadání je rozdělen do více bloků. Hlavními prvky jsou Raspberry Pi a ATmega128A, proto je nejprve ověřeno, zda je Raspberry Pi, které bude řídit komunikaci přes rozhraní UART, schopno přijímat a odesílat data. Dále jsou data odeslána z Raspberry Pi na mikrokontrolér k ověření, zda dokáže přes UART komunikovat i ATmega128A. V poslední části je pak ověřena schopnost mikrokontroléru řídit vykreslování na displej. Rozdělením úkolu do více menších částí je zaručeno, že při výskytu chyby bude jednodušší tuto chybu lokalizovat.

Hlavním úkolem této práce je návrh a realizace programového vybavení, přes které bude možno navrhovat a zobrazovat data na displej. Systém bude také schopen zobrazovat animace. Návrh obrazovek bude probíhat přes aplikaci na operačním systému Windows. Součástí této aplikace bude bitmapa, jejíž rozlišení bude odpovídat rozlišení panelu. Na bitmapě dochází k návrhu obrazovek na displej. Návrh obrazovek probíhá pomocí počítačové myši, která slouží pro změnu jednotlivých pixelů, nebo klávesnice pro zadávání znaků. Dalším způsobem návrhu obrazovek bude načtení celé obrazovky ze souboru, ve kterém bude v definovaném formátu uložena.

2 LITERÁRNÍ REŠERŠE

Obecně se zobrazovací jednotky mohou lišit vlastním rozvržením plochy zobrazující informace. Existují displeje segmentové a displeje maticové.

Segmentové displeje mají zobrazovanou plochu rozdělenou na jednotlivé bloky segmentů, popř. je celý displej složen pouze z jednoho bloku. Nejčastějším typem segmentových displejů jsou displeje sedmi-segmentové, kdy se jeden blok skládá ze sedmi segmentů. Sedmi-segmentový displej zobrazuje číslice 0-9. Je také schopen zobrazovat čísla v hexadecimálním tvaru, kdy se na displeji zobrazí znaky A, B, C, D, E nebo F. Tyto displeje se používají především k zobrazení informace ve formě číslice.

Oproti tomu maticové zobrazovací jednotky mohou zobrazit prakticky libovolný znak, mají-li požadované rozlišení, tj. počet bodů v řádku ku počtu bodů ve sloupci. Displej maticové jednotky je tedy dělen na jednotlivé body a počet těchto bodů udává rozlišení. Maticové displeje se mohou dělit na alfanumerické a grafické displeje.

Grafické displeje neobsahují žádné menší bloky. Celá plocha obrazovky je rovnoměrně rozdělena na matici bodů. Grafické displeje se dělí podle technologie zobrazování informace. Tématem této práce je návrh řídícího systému pro terčíkový displej, nejprve budou stručně popsány ostatní typy displejů, a poté samotný terčíkový displej. [1]

Alfanumerické displeje dokáží zobrazovat číslice a písmena, popřípadě jednoduché znaky. Displej alfanumerické jednotky se skládá z více menších bloků, podobně jako u segmentových displejů. Zde se však bloky nedělí na segmenty, ale na matice bodů. Tyto bloky se obvykle skládají z 5x7 bodů (popř. 5x8). Informace zobrazena na tomto displeji se skládá ze znaků (číslic a písmen) zobrazených na jednotlivých blocích. Alfanumerické displeje jsou používány v aplikacích, kdy není zobrazovaná informace rovnoměrně rozložena po celém displeji a zobrazuje se pouze na některých částech obrazovky. Při použití grafického displeje by vznikly oblasti na displeji, které by zůstaly nevyužity. [2]

2.1 LED displeje

LED displej je složen z matice LED diod. LED dioda (Light Emitting Diode) je složena z PN přechodu, který po přiložení napětí mezi anodu (kladný náboj) a katodu (záporný náboj) emituje světlo. Barva vyzařovaného světla závisí na chemickém složení samotného polovodiče. Dioda má na povrchu obvykle optický prvek, který zlepšuje její optické vlastnosti. [3]

Výhoda LED displejů spočívá v dobré viditelnosti zobrazované informace i za sníženého osvětlení. Nutnost stálého napájení a odběr energie je hlavní nevýhodou

LED displejů. Tento problém se řeší impulsním napájením LED diod, čímž je snížena spotřeba energie. Člověk nevnímá svícení LED diody napájené impulsně jako blikání, je to způsobeno nedokonalostí lidského oka. Toto platí pouze za splnění určité minimální hodnoty frekvence. Při nižší hodnotě frekvence bychom již blikání vnímali. [4]

2.2 LCD displeje

LCD technologie (Liquid Crystal Display) je založena na principu tekutých krystalů. Tekutý krystal je látka, která dokáže setrvat jak v kapalném, tak v pevném skupenství a je schopna vytvořit krystalickou strukturu. Tekuté krystaly nejsou opticky aktivní, ke své činnosti potřebují zdroj světla.

Využívá se dvou principů. Prvním z principů je využití dopadajícího světla, které se odráží od zrcadla nacházejícího se pod krystaly. Záleží na natočení krystalů, zda odražené světlo propustí či nikoli. Druhým principem je umístění zdroje světla pod krystaly. Stejně jako v předchozím případě zapnutí/vypnutí pixelu záleží na natočení daného krystalu. Tento princip nevyžaduje okolní osvětlení a funguje tedy i za snížené viditelnosti.

Průhlednost nebo zabarvení krystalů je dáno velikostí přiloženého střídavého napětí nebo působením tepla na krystaly. Velikost dodávaného náboje určuje míru zakalení krystalů. LCD displej je složen ze dvou navzájem otočených polarizačních filtrů, mezi kterými jsou tekuté krystaly. [5]

2.3 OLED displeje

Fungují na základě organické elektroluminiscenční látky. Organická látka je v několika vrstvách umístěna mezi dvě elektrody (katodu a anodu), jedna z nich musí být průhledná. Nejčastější volbou průhledné elektrody je anoda. Při přiložení napětí excitují fotony z polovodičové vrstvy, tento jev je způsoben rekombinací záporných elektronů a děr v emisní vrstvě. Organická látka je schopna při excitaci vyzářit světlo o dané vlnové délce.

Existuje více typů OLED displejů, jedním z nich jsou pasivní PMOLED displeje (Passive Matrix Organic Light Emitting Diode). Elektrody vytvářejí mříž, jednotlivé pixely se nacházejí v místech křížení anody a katody. Aktivní AMOLED displeje (Active Matrix Organic Light Emitting Diode) mají anodu překrytou tenkou vrstvou TFT (Thin Film Transistor) maticí. TFT matice rozhoduje o zapnutí/vypnutí jednotlivých pixelů. Aktivní OLED displeje nepotřebují externí ovladače, a proto mají menší spotřebu než pasivní. [6]

Výhodnou OLED displeje oproti například LCD displejům je menší elektrická náročnost, nepotřebují totiž podsvícení. Avšak tyto displeje mají krátkou

životnost, což je jejich hlavní nevýhodou. Dokonce není ani stejná životnost pro různé barvy.

2.4 Elektroforetické displeje

Princip elektroforetického displeje spočívá v použití mikro kapslí obsahujících opačně nabité bílé a černé částice. Displej je pokryt tenkou vrstvou malých mikro kapslí, počet mikro kapslí se pohybuje v řádech milionů, záleží na velikosti displeje. Natočení kapslí je způsobeno impulsem elektrického proudu. Na základě polarity se zobrazí černá nebo bílá. Informace zůstane na displeji čitelná i po odpojení napájení. Je to způsobeno přítomností vysoce viskózního roztoku, který uchová kapsle v poloze beze změny. K zobrazení barevné informace se používá displej složený z více vrstev mikro kapslí. [7]

Displeje nefungují na principu externího podsvícení, k čitelnosti tedy potřebují dobrou viditelnost, podobně jako dále zmiňované terčíkové displeje. Displeje se vyznačují i širokými pozorovacími úhly. Používají se převážně v elektronických čtečkách knih.

Mezi výhody patří velký pozorovací úhel, malá spotřeba energie, čehož se využívá právě v elektronických čtečkách, displeje tedy nejsou náročné na baterii. Dále se displeje využívají v digitálních hodinkách. Další výhodou je tenkost displeje. Nevýhodou displejů je pomalá změna informace.

2.5 Terčíkové displeje

Terčíkové zobrazovací jednotky jsou složeny z matice otočných disků. Jedna strana disku je černá druhá strana barevná. Nejčastěji volenou barvou je žlutá. Pro lepší viditelnost bývá barevná strana vyrobena z reflexního materiálu. Zobrazení informace tedy probíhá otáčením jednotlivých terčíků ze stavu „vypnuto“, kdy je viditelná černá plocha a stavu „zapnuto“, kdy je viditelná barevná plocha. Pod terčíkem se nachází cívka, která je zdrojem magnetického pole. Disk je připevněn na ose, na které se nachází permanentní magnet. Různé panely mají různá místa přichycení magnetů, někdy se také využívá přichycení magnetu na samotný disk. V tomto případě se magnety přichytí na dva konce disku. Disk je otáčen pomocí magnetického pole, které vzniká díky cívce, která je protékána proudem. Do cívky je přiveden proudový impuls, který způsobí otočení disku. Směr magnetického pole (směr otočení terčíku) závisí na směru proudu cívkou. Příklad terčíkového displeje je na Obr. 2.1. [8]

Terčíkové displeje se používají ve vozidlech městské hromadné dopravy, popř. na informačních tabulích podél cest. Nutností je použití na dobře osvětlených místech, jelikož neobsahují žádný zdroj světla. Někdy se využívají tzv. DOT-LED

panely, které jsou doplněny o LED diodu pro dosažení čitelnosti i za snížené viditelnosti. Využívají se především na velkých plochách s malou rychlostí změny zobrazované informace. Panely jsou využívány i na sportovištích, např. pro zobrazení skóre. [9]

Obr. 2.1 Terčíkový displej [10]

2.5.1 Použitý panel

K minimalizování počtu přívodních kabelů se používá k ovládání jednotlivých terčíků pole cívek. Cívky mají jeden kontakt společný ve sloupci a druhý kontakt společný pro řádek.

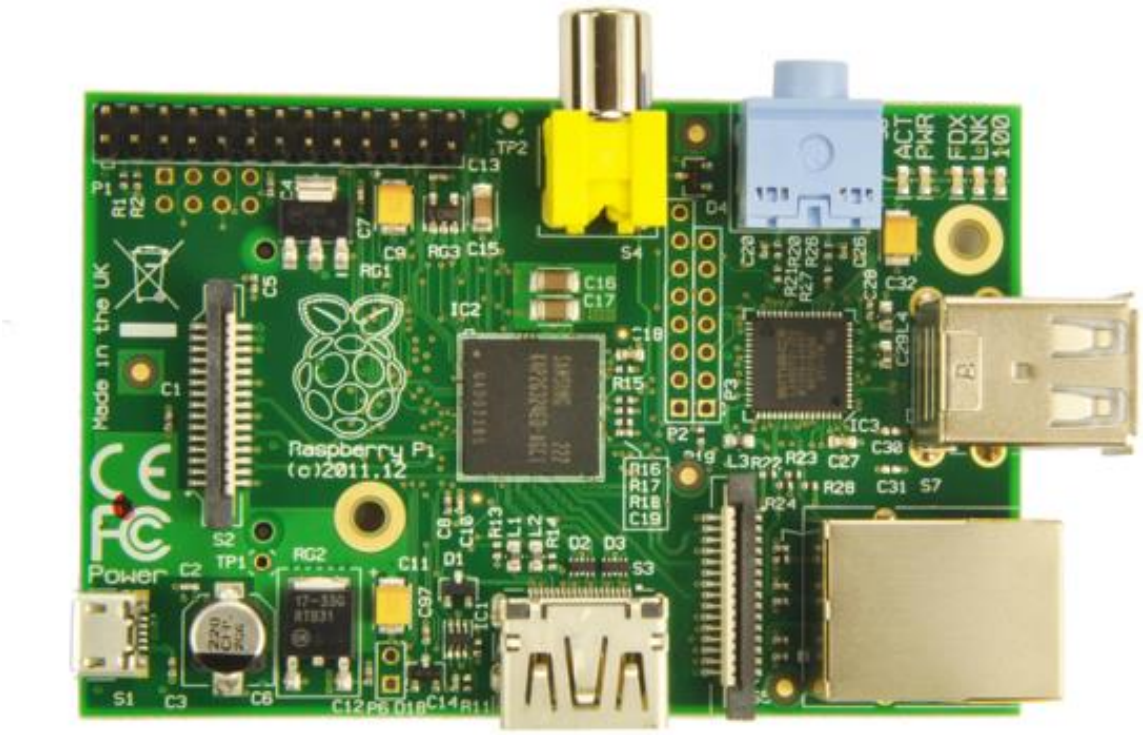
Na vstupu displeje se nachází pinový konektor o 50 pinech. Zde použitý displej má rozlišení 19 řádků na 140 sloupců, sloupce se dále dělí 5 panelů, které se dále dělí na 4 skupiny po 7 sloupcích. Celkem vznikne 20 skupin po 7 sloupcích. K výběru řádku slouží piny 1-19. K výběru sloupce slouží piny 30-37, přičemž piny 35, 36 a 37 slouží k výběru jednoho z 5 menších panelů, piny 30 a 31 k výběru jedné ze čtyř skupin sloupců. Piny 32, 33 a 34 slouží k výběru konkrétního sloupce. Výhoda těchto panelů spočívá v menší spotřebě energie. Po přivedení impulsu na cívku a otočení disku není dále třeba další dodání energie. Informace je zobrazena i po odpojení napájení od displeje. Zobrazení informace se zakládá na mechanickém principu, terčíkové displeje jsou tedy náchylné na různé poruchy opotřebením. Snížení četnosti poruch se dá dosáhnout pomocí pravidelné údržby. Nutnost údržby je jedna z nevýhod, jelikož u předem zmíněných panelů údržba prakticky není potřebná. Další nevýhodnou je pomalá změna zobrazované informace stejně jako u elektroforetických displejů a nutnost externího osvětlení.

3 ELEKTRONIKA PANELU

Základním hardwarovým prvkem zobrazovací jednotky je mikropočítač Raspberry Pi. Mikropočítač řídí komunikaci s mikrokontrolérem ATmega128A, který řídí vykreslování na displej. ATmega128A využívá k řízení displeje obvody s dalšími elektronickými prvky. Mezi tyto prvky patří sada několika dekodérů typu 74HC238 a tranzistorová pole typu ULN2803 a UDN2980LW. Napájecí zdroj je složen ze dvou transformátorů napětí, usměrňovacích diod a stabilizátorů napětí.

3.1 Mikropočítač Raspberry Pi

Při práci byl využit mikropočítač Raspberry Pi model B, deska mikropočítače je na Obr. 3.1. Jedná se o jednočipový mikropočítač s ARM procesorem patřícím do skupiny RISC (Reduced Instruction Set Computers). Tyto architektury pracují s redukovanou instrukční sadou.



Obr. 3.1 Mikropočítač Raspberry Pi [11]

Velikost operační paměti je 256 MB. Mikropočítač umožňuje propojení pomocí dvou USB portů a Ethernet portu, který je připojen skrz rozhraní USB. Mikropočítač je napájen přes mikro-USB konektor zdrojem 5 V s maximálním odběrem 1 A. Na spodní straně se nachází slot pro SD kartu, na kterou je nahráván operační systém a představuje většinu paměti mikropočítače. [12]

Verze mikropočítače Raspberry Pi B má 26 pinů. Na pinech jsou vyvedeny dvě sběrnice SPI, jedna UART a I2C. V této práci bude využita pouze UART. Dále má na pinech vyvedené napěťové úrovně 3,3; 5 a 0 *V* (tedy GND – zem). Raspberry Pi má v této aplikaci hlavní řídicí funkci. Pomocí sběrnice UART posílá na mikrokontrolér ATmega128A data, která mají být zobrazena na displeji. Sběrnice UART je vyvedena na pinech 14 (TXD) a 15 (RXD). [13]

Již dříve jsem se setkal s verzí obrazu Raspbian Jessie Lite, proto i nyní využiji tuto verzi. Obraz operačního systému je k dispozici ke stažení na oficiálních stránkách ve formátu zip. Operační systém je nejprve třeba zapsat na SD kartu, na které se nachází hlavní část paměti mikropočítače. SD karta se vkládá do slotu na spodní straně desky mikropočítače.

Komunikace s mikropočítačem bude probíhat přes vzdálený přístup. K tomu bude využit terminál SSH (Secure Shell). Funkce SSH je defaultně vypnuta, pro použití SSH hned při prvním bootování je třeba na SD kartě ve svazku Boot vytvořit prázdný soubor ssh (bez přípony), který SSH pro prvotní bootování povolí. Na straně počítače bude použit klient PuTTY.

Program pro mikropočítač bude vytvářen v programovacím jazyce C. Nabízí se možnost volby mezi jazykem Python a C, avšak vzhledem k žádné předešlé zkušenosti s jazykem Python jsem se rozhodl pro jazyk C. Pro první nastavení a oživení Raspberry bude použit ethernet port. Při další práci bude využit Wi-Fi adaptér, jehož použití je v této aplikaci výhodnější. Wi-Fi adaptér se nastaví upravením konfiguračních souborů *interfaces* a *wpa.conf*, nacházejících se v adresáři *etc*. A v navazující práci bude využit ke komunikaci s informačním systémem. [14]

3.2 Mikrokontrolér ATmega128A

Jedná se o 8-bitový mikrokontrolér se 128 kB programovatelnou Flash pamětí. Pracuje s rozšířenou RISC architekturou. Má 53 programovatelných vstupně/výstupních pinů. Je schopen komunikovat přes dvě sběrnice UART. V této práci je využita sběrnice s číslem 0. Ke zvolení jedné ze dvou sběrnic dochází právě připsáním čísla k názvu registru. Mikrokontrolér dále obsahuje rozhraní JTAG a ISP, která jsou obě vyvedena na desce plošných spojů.

Na mikrokontroléru ATmega128A se nachází hned několik typů pamětí. Výše zmíněná paměť Flash o velikost 128 kB sloužící k uložení instrukcí programu. Data jsou v paměti uchována i po vypnutí napájení. Dalším typem paměti je EEPROM o velikosti 4 kB. Používá se k uložení nastavení při běhu programu. Podobně jako u paměti Flash se data z paměti EEPROM po odpojení napájení neztratí. Dále obsahuje 4 kB interní SRAM paměti na kterou se ukládá paměť přechodných dat. [15]

Mikrokontrolér řídí vykreslovaní obrazu na displej. Displej se skládá z pole cívek, které vytvořením magnetického pole po průchodu proudového impulsu otočí terčík. Směr natočení závisí na směru proudu. Cívka má vždy jeden kontakt společný s celým řádkem cívek a druhý kontakt společný s celým sloupcem cívek. K otočení konkrétního terčíku je třeba vybrat konkrétní sloupec a konkrétní řádek a poté zvolit směr proudu. Ke spojení s displejem je z displeje vyveden 50-ti pinový konektor. Význam jednotlivých pinů je následující.

Piny 1-19 slouží pro výběr řádku. Piny 30-37 slouží k výběru sloupce. Jak již bylo zmíněno výše, displej je rozdělen na 5 panelů, na každém z nich se nachází 28 sloupců, které se dále dělí na 4 skupiny sloupců. Piny 35-37 slouží k výběru jednoho z pěti panelů. Piny 30 a 31 slouží k výběru jedné ze čtyř skupin sloupců a piny 32-34 slouží k výběru konkrétního sloupce. Výběr probíhá pomocí přivedení logických úrovní 1 nebo 0 na příslušné piny. Dále je potřeba zvolit natočení terčíku (žlutá nebo černá strana), to je umožněno pinem 39. K přivedení proudového impulsu na vybranou cívku slouží pin 38. Piny 30-39 jsou přímo přivedeny na mikrokontrolér, jejich řízení bude vysvětleno v kapitole 4.2.

3.2.1 Logické obvody pro výběr řádku

Obvody jsou složeny ze tří hlavních součástek, mezi které patří dekodér typu 74HC238, tranzistorové pole UDN2982LW pro záporný směr proudu a tranzistorové pole ULN2803 pro kladný směr proudu. Dekodér 74HC238 se zde nachází celkem 7krát. V první vrstvě je jeden dekodér, pomocí kterého se vybere jeden ze šesti dalších dekodéru. Dekodér 74HC238 obsahuje tři adresové vstupy, tři enable piny a osm výstupních pinů. [16]

Úkolem dekodéru v první vrstvě je vybírat jeden z šesti dalších dekodéru. Výběr je prováděn na základě nastavení tří vstupních adresových pinů, které nastavuje řídicí mikrokontrolér. Dekodéry v druhé vrstvě jsou také typu 74HC238, jejich dva enable piny jsou přivedeny na GND, na třetí enable každého dekodéru je přiveden vždy jeden výstupní signál z dekodéru v první vrstvě. Na tři adresové vstupní piny šesti dekodérů v druhé vrstvě jsou připojeny další tři výstupní piny mikrokontroléru. Výstupy těchto dekodérů jsou připojeny na vstupy tranzistorových polí, která se starají o výkonové přizpůsobení. Podrobný popis výběru řádku bude uveden v kapitole 4.2.

3.3 Napájecí zdroj

Napájecí zdroj je rozdělen na dvě části. V první částí je na vstup ze sítě přes pojistku připojen transformátor. Tento transformátor transformuje sítově napětí na napětí 40 *V*. Za transformátorem se nachází stále střídavé napětí. Pro účel usměrnění napětí je použit usměrňovací diodový můstek. Výstupní napětí

z můstku je v jedné větvi přivedeno na snižující stabilizátor 78S24. Jedná se o stabilizátor pevného napětí 24 *V*. Za stabilizátorem se nachází pojistka a poté je napětí 24 *V* přivedeno na konektor, který bude připojen na desku s mikrokontrolérem.

Druhá větev z usměrňovače v první části napájecího zdroje je přivedena na snižující spínaný regulátor LM2575-12. Na vstupu tohoto stabilizátoru je také 40 *V*. Výstupních 12 *V* regulátoru je také přes pojistku přivedeno na konektor.

V druhé části zdroje se nachází transformátor transformující napětí sítě na 7,5 *V*. Opět má na výstupu střídavé napětí, které je usměrněno přes usměrňovací diodový můstek. Za diodovým můstkem je zařazen opět snižující spínaný regulátor. Druhý regulátor je však typu LM2596T-5. Na výstupu regulátoru je napětí 5 *V*. Toto napětí je přivedeno jak na konektor, tak na USB výstup zdroje, opět jsou osazeny pojistky.

4 OŽIVENÍ PANELU

Před samotnou tvorbou řídicího systému bude nejprve oživena komunikace mezi jednotlivými elektronickými prvky panelu. Oživení panelu bude rozděleno do dvou částí. Nejprve bude zprovozněna komunikace mezi mikropočítačem Raspeberry Pi a mikrokontrolérem ATmega128A, poté bude pomocí mikrokontroléru zprovozněno vykreslení dat na displej.

Součástí této práce nebyl návrh, tvorba a osazení desek plošných spojů. Proto došlo nejprve ke kontrole obvodů nacházejících se na těchto deskách, jelikož jsem si všiml, že některé spoje byly propojeny pomocí drátů, nikoliv rozvodem po desce plošných spojů. K propojení byl místy použit i cínový drát bez vnější izolace, mohlo tedy dojít ke zkratům. Při kontrole citlivých míst, na kterých by se mohly vyskytovat zkraty díky absenci izolace, jsem použil voltmetr a daná místa zkontroloval. Žádný zkrat jsem na deskách neobjevil. Zjistil jsem však, že USB konektor, který je na napájecí desce vyveden za účelem napájení Raspberry Pi, nemá být využíván. Raspberry Pi bude tedy třeba napájet z vnějšího zdroje.

Po připojení napájecího zdroje k síti jsem zjistil, že ze zdroje jde místo napětí o velikosti 5 *V*, napětí velikosti 7,5 *V*. Tato větev je tvořena pouze malým počtem prvků, mezi které patří kondenzátory, usměrňovač, transformátor a snižující napěťový regulátor LM2596T-5. Jako první možnost zdroje problému přišel v úvahu posledně zmíněný napěťový regulátor. Na základě dokumentace tohoto prvku jsem se dozvěděl, že by měl při vstupním napětí velikosti 7 až 40 *V*, držet na výstupu hodnotu 4,8 – 5,2 *V*. Na výstupu jsem však naměřil 7,5 *V*. Rozhodl jsem se tedy pro výměnu tohoto prvku. [17]

Po osazení nové součástky bylo na výstupu stále napětí o velikosti 7,5 *V*. Problém tedy nebyl v regulátoru. Postupně jsem zkusil proletovat kontakty všech ostatních prvků této větve zdroje a ukázalo se, že při opětovném proletování kontaktu u jednoho z kondenzátorů se na výstupu objevila správná hodnota napětí.

4.1 Komunikace Raspberry Pi s ATmega128A

Jelikož jsem se nikdy dříve nesetkal s programováním mikrokontroléru, rozhodl jsem se nejprve ověřit funkčnost sériové komunikace na Raspberry Pi. Poté budu schopen oživit komunikaci s mikrokontrolérem Atmega, neboť budu mít ověřeno, že na mikrokontrolér posílám validní data. Jak již bylo zmíněno v kapitole 3.1, na Raspberry Pi jsou vyvedeny *RX* a *TX* piny sloužící k sériové komunikaci, přes kterou jsou data odesílána na mikrokontrolér ATmega. Nejdříve ověřím, zda dokáže Raspberry Pi přes vyvedené *RX* a *TX* piny odesílat a přijímat data.

4.1.1 Sériová komunikace na Raspberry Pi

Jednou z možností zprovoznění komunikace by bylo připojení externího přijímače a vysílače. Zvolil bych konkrétně Arduino, se kterým jsem se již setkal. Informace by se poslala na Arduino a z Arduina zpět na Raspberry Pi. Problém by představovaly různé napěťové úrovně, na kterých tyto prvky pracují. Arduino pracuje s napěťovou úrovni 5 *V*, Rasberry Pi na napěťové úrovni 3,3 *V*. Problém by se dal vyřešit například napěťovým děličem. Jednodušším způsobem je zkratování *TX* a *RX* pinů na Raspberry Pi. Tím bude zaručeno, že se odesílaná data budou přijímána zpět na mikropočítači. Tvorba programu bude probíhat v programovacím jazyce C. Při výběru knihovny pro komunikaci přes UART jsem se rozhodoval mezi knihovnou PIGPIO a knihovnou Wiring Pi. Pro prvotní oživení spojení mezi mikropočítačem a mikrokontrolérem jsem zvolil Wiring Pi, jejíž použití se mi zdá jednodušší.

Při prozkoumávání knihovny jsem zjistil, že je nejprve třeba zvolit rychlost komunikace a zařízení, se kterým bude Raspberry Pi komunikovat. Zvolené zařízení je v prvním argumentu funkce *serialOpen*. V mém případě jsem zvolil zařízení, které je připojeno na *TX* a *RX* pinech. Pomocí druhého argumentu funkce se definuje přenosová rychlost. Rychlost nastavím na nejčastěji používanou hodnotu 9600 *Bd*. Funkce *serialOpen* vrací identifikaci zařízení, se kterým byla zahájena komunikace. V mém případě je identifikace uložena do proměnné *fd* typu *int*. Při chybě vrací *serialOpen* hodnotu -1. Pomocí podmínky *if* ověřuji, zda bylo správně inicializováno zařízení, při neúspěchu je program ukončen. [18]

```
if ((fd = serialOpen ("/dev/ttyAMA0", 9600)) < 0)
        return 1;
```

Po inicializování knihovny WiringPi pomocí funkce *wiringPiSetup*, jsem již schopen posílat a přijímat data přes *TX* a *RX* piny. K odesílání dat použiji funkci *serialPutchar* a k přijímání dat funkci *serialGetchar*. Nejprve je odeslán znak na zařízení s identifikací uloženou v proměnné *fd*. Odeslaný znak je pro následnou kontrolu vytisknut na konzoli. Po odeslání znaku se ve smyčce *while* čeká na přijatá data na *RX* pinu pomocí funkce *serialDataAvail*. Jsou-li nějaká data dostupná, funkce *serialDataAvail* vrací hodnotu 1, v opačném případě vrací funkce hodnotu 0. Tím je detekováno, že jsou nějaká data dostupná v přijímacím zásobníku. Uvnitř smyčky *while* dochází po splnění podmínky ke čtení těchto dat pomocí výše zmíněné funkce *serialGetchar*. Přečtená data jsou opět pro kontrolu vytisknuta na konzoli. Jediným parametrem funkce *serialGetchar* je identifikace zařízení, ze kterého se mají data číst.

```
serialPutchar (fd, 0xff);
        printf ("Odeslano: %x\n", 0xff);

while (serialDataAvail (fd))
      printf ("Prijato: %x\n", serialGetchar(fd));
```

Po spuštění programu se na konzoli vypsal očekáváný výstup, odeslaná data se shodovala s přijatými. Sériová komunikace na Raspberry Pi je tedy funkční. Dále bude místo zkratování *RX* a *TX* pinů připojen přes UART mikrokontrolér ATmega. A stejný program v Raspberry Pi bude použit ke komunikaci s mikrokontrolérem.

4.1.2 Komunikace s ATmega128A

Při komunikaci s mikrokontrolérem ATmega přes rozhraní UART jsou využity piny *TXD0* a *RXD0*. Před zahájením komunikace je nejprve třeba definovat kmitočet oscilátoru, aby bylo možné zadat přenosovou rychlost. Kmitočet oscilátoru je nastaven na 8 *MHz*. Přenosová rychlost je nastavena opět na 9 600 *Bd*. Ke správné funkci je třeba na obou zařízeních nastavit shodnou hodnotu rychlosti. Na mikrokontroléru je třeba přenosovou rychlost přepočítat přes nastavenou frekvenci oscilátoru.

```
#define F_CPU 8000000UL
#define BAUDRATE 9600
#define _BAUDRATE F_CPU/16/BAUDRATE - 1
```

V programu jsou definovány dvě globální proměnné. První proměnná *Data* typu *unsigned char* bude sloužit jako zásobník, budou se do ní tedy ukládat přijatá data. Druhá proměnná *State* typu *unsigned int* bude sloužit ke zjištění, zda došlo k přijetí dat či nikoliv.

```
unsigned char Data;
unsigned int State;
```

Po odeslání znaku z Raspberry Pi bude znak přijat na mikrokontroléru. Při přijetí znaku bude proměnná *State* nastavena na hodnotu 1. V nekonečné smyčce ve funkci *main* se bude kontrolovat, zda došlo ke změně hodnoty proměnné *State*. Při detekci změny hodnoty této proměnné bude zavolána funkce *UARTWrite*, která přes UART pošle načtená data v proměnné *Data* zpět.

Pro inicializování UART komunikace jsou využity registry *UBRR0L* a *UBRR0H*, které slouží pro nastavení přenosové rychlosti. Dále jsou využity registry *UCSR0B* a *UCSR0C* pro povolení komunikace a nastavení formátu přenášených dat.

```
void UARTInit(uint16_t baud)
        {
        UBRR0L = baud;
        UBRR0H = (baud >> 8);

        UCSR0B = ((1<<TXEN0) | (1<<RXEN0) | (1<<RXCIE0));
        UCSR0C = (1<<UCSZ01) | (1<<UCSZ00));
        }
```

K inicializování UART komunikace slouží funkce *UARTInit*. Jediným parametrem této funkce je rychlost komunikace. Ve funkci se nejprve nastaví registry *UBRR0L* a *UBRR0H*. Do spodního registru (*UBRR0L*) se uloží spodních 8 bitů informace o rychlosti přenosu, zbylé vyšší bity jsou uloženy do horního registru (*UBRR0H*). Uložení vyšších bitů do horního registru je umožněno pomocí operace bitového posunu, kdy je informace o přenosové rychlosti binárně posunuta o 8 bitů směrem doprava. Dále se v registru *UBRR0B* nastaví bity *RXEN0*, *TXEN0* a *RXCIE0* na hodnotu 1. Bit *RXEN0* povolí příjem dat na pinu RX0, bit *TXEN0* povolí odesílání dat přes pin TX0. Pomocí bitu *RXCIE0* se povolí přerušení z pinu *RX0*. Pomocí registru *UBRR0C* nastavím formát přenášených dat. Bitem *UCSZ00* a *UCSZ01* je nastavena délka přenosové informace na 8 bitů.

Ke čtení dat slouží funkce přerušení *ISR*. Do parametru funkce se vybere přerušení z vektoru přerušení, v tomto případě tedy *USART0_RX_vect*. Ve funkci se data z registru *UDR0* načtou do proměnné *Data* a proměnná indikující přečtení dat *State* se nastaví na hodnotu 1.

```
ISR (USART0_RX_vect)
        {
        Data = UDR0;
        State = 1;
        }
```

Data se odesílají zpět pomocí funkce *UARTWrite*. Funkce má v parametru data, která se mají odeslat. Je využito registru *UCSR0A*, pomocí bitu *RXC0* se kontroluje, zda se v přijímacím zásobníku nacházejí nějaká nepřečtená data. Do registru *UDR0* se uloží data na odeslání, která byla předána pomocí parametru

funkce. Po splnění podmínky ve smyčce *while* se data uloží do registru *UDR0* a jsou odeslána.

```
void UARTWrite(char Data)
        {
        while(!(UCSR0A &(1<<UDRE0)));
        UDR0 = Data;
        }
```

Hlavní funkcí programu je funkce *main*. Ve funkci *main* je postupně inicializováno rozhraní UART pomocí funkce *UARTInit*, které nastaví rychlost přenosu a formát dat. Poté se proměnná *State*, indikující přijetí dat, nastaví na výchozí hodnotu 0. Nyní tedy není v nekonečné smyčce ve funkci *while* splněna podmínka. Ve chvíli, kdy se data načtou a proměnná *State* se nastaví na hodnotu 1, splní se podmínka ve smyčce a data z proměnné *Data* se odešlou. Po odeslání dat je proměnná *State* resetována na výchozí hodnotu 0.

```
int main()
        {
        UARTInit();
        sei();
        State = 0;
        while(1)
                {
                if (State == 1)
                        {
                        UARTWrite(Data);
                        State = 0;
                        }
                _delay_ms(250);
                }
        }
```

Při testování komunikace mezi mikropočítačem a mikrokontrolérem jsem odeslal data z Raspberry Pi. Data se přečetla na mikrokontroléru a úspěšně vypsala na konzoli zpět v Raspberry Pi. Komunikace mezi ATmega128A a Raspberry Pi je tedy funkční.

4.1.3 UART

Výše zmiňované rozhraní pro sériovou komunikaci UART (Universal Asynchronous Receiver-Transmitter) se řadí mezi tzv. full duplex. U těchto zařízení lze současně využívat funkci přijímání i odesílání. Na jedné straně je master, který řídí komunikaci, na druhé straně slave. V tomto případě slouží Raspberry Pi jako master a ATmega128A jako slave. [19]

Odesílaný paket začíná start bitem, ten má logickou hodnotu 0. Po start bitu následuje pět až devět bitů obsahujících data. Za datovými bity je paritní bit, který je volitelný, kontroluje se přes něj stav odeslaných/přijatých dat. Nakonec za paritním bitem se nacházejí jeden až dva stop bity s logickou úrovní 1. Přenosová rychlost se udává v baudech (počet přenesených bitů za vteřinu). Nejčastěji používanými přenosovými rychlostmi jsou 9600 a 19200 baudů. Avšak používají se i další hodnoty od 300 do 115000 *Bd*, záleží na konstrukci daného zařízení.

4.2 Řízení displeje mikrokontrolérem

Řízení displeje probíhá zápisem do tří datových registrů mikrokontroléru ATmega128A. Registr *PORTA* slouží k výběru řádku, *PORTB* slouží k výběru sloupce a část *PORTC* sloužící k výběru směru natočení terčíku a generování enable impulsu. Displej je složen z matice cívek, z nichž každá cívka má jeden společný kontakt pro sloupec a druhý společný kontakt pro řádek. Po vybrání konkrétního sloupce a konkrétního řádku je možno přes řídicí elektroniku panelu přivést na danou cívkou proudový impuls v kladném nebo záporném směru a natočit konkrétní bod displeje. Význam jednotlivých bitů *PORTC* je v Tabulka 4.1.

Tabulka 4.1 Význam bitů PORTC

PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
SMĚR	ENABLE	-	-	-	-	LED	LED

4.2.1 Výběr řádku

Jak bylo zmíněno výše, displej obsahuje celkem 19 řádků. Výběr řádku je řízen 7 dekodéry ve dvou vrstvách. Tři dekodéry řídí výběr řádku pro kladný směr proudu a tři dekodéry pro záporný směr proudu. Poslední dekodér volí jeden ze šesti těchto dekodérů. Výběr jednoho z šesti dekodérů probíhá přes *PORTA7-6*. Po vybrání konkrétního dekodéru v druhé vrstvě zbývá vybrat konkrétní řádek. Řádek je vybrán pomocí *PORTA5-3*. Tyto piny jsou přivedeny na vstupní adresové bity dekodérů v druhé vrstvě. Význam jednotlivých bitů pro *PORTA* je přehledně uveden v Tabulka 4.2.

Tabulka 4.2 Význam bitů PORTA

PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
VÝBĚR DEKODÉRU		VÝBĚR ŘÁDKU			-	-	-

4.2.2 Oprava zapojení dekodérů

Zde se objevil největší problém s oživením displeje, jelikož se dekodéry nacházejí na různých deskách, jsou propojeny přes konektor. Na jedné desce jsou ve schématu značeny Y na druhé desce E. Logicky jsem se domníval, že Y6 bude připojeno na E6. Ukázalo se však, že konektory nejsou takto jednoduše kompatibilní. Po vícero pokusech o vykreslení dat na displej jsem zkontroloval, zda opravdu sedí tato čísla. Pomocí osciloskopu jsem tedy detekoval impulsy za hlavním dekodérem. Hlavní dekodér vybírá jeden z šesti dalších dekodérů. Ukázalo se, že skutečné propojení pinů obou konektorů je poměrně chaotické. Správné propojení je pak znázorněno v Tabulka 4.3. Pro správné zapojení bylo vytvořeno schéma, které zobrazuje propojení sedmi dekodérů. Schéma se nachází v přílohách.

Tabulka 4.3 Logický výběr dekodérů druhé vrstvy

PORTC7	PORTA7	PORTA6	Y/E
0	0	0	Y6/E1
0	0	1	Y5/E2
0	1	0	Y4/E3
1	0	0	Y2/E4
1	0	1	Y1/E5
1	1	0	Y0/E6

K otočení daného pixelu je třeba zároveň zvolit řádek i sloupec, na kterém se daný pixel nachází. Jelikož byly špatně zapojeny vstupy dekodérů, docházelo k nekorektní volbě řádku. Po nahrání programu displej nereagoval, přestože jsem zkontroloval výstupní hodnotu přímo na pinech mikrokontroléru, zda není chyba v programu. Na pinech jsem naměřil vždy správné napěťové úrovně, a proto jsem se zprvu domníval, že chyba bude spíše v napájecím zdroji nebo v části s výběrem sloupce, která se nachází přímo na panelech, než ve špatném pojmenování spojů ve schématu.

4.2.3 Výběr sloupce

Pro výběr sloupce je na konektoru z displeje vyvedeno 8 pinů. Na displeji se nachází logický výběr sloupce, principem podobný jako pro výběru řádku. Já tedy pouze řídím těchto 8 vyvedených pinů pomocí zápisu informace do *PORTB*. Displej obsahuje celkem 140 sloupců, které jsou rozděleny mezi 5 panelů.

Je z nich třeba zvolit jeden aktivní. Výběr aktivního panelu se volí pomocí tříbitové logické hodnoty zapsané v registru *PORTB* na spodních třech bitech *PORTB2-0*. Zapsáním například 001, pro jednotlivé bity registru konkrétně *PB2* = 0, *PB1* = 0 a *PB0* = 1 bude vybrán 2. panel. Na panelu se nachází 28 sloupců po 4 skupinách. K výběru skupiny stačí dvoubitová informace. Výběr skupiny řídí bity *PORTB7* a *PORTB6*. Zápisem například 01, teda *PB7* = 0 a *PB6* = 1 bude vybrána 2. skupina. Zbývá zvolit konkrétní sloupec pomocí třech zbylých bitů. Zapsáním binárního čísla 001 na bity *PORTB5-3*, konkrétně *PB5* = 0, *PB4* = 0 a *PB3* = 1 bude vybrán 2. sloupec. Přehledný význam bitů *PORTB* je v Tabulka 4.4.

Tabulka 4.4 Význam bitů PORTB

PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
VÝBĚR SKUPINY		VÝBĚR SLOUPCE			VÝBĚR PANELU		

4.2.4 Program

Před použitím data registrů mikrokontroléru je nejprve třeba nastavit konkrétní piny těchto registrů jako výstupní. Nastavení používaných pinů na výstupní je umožněno zapsáním logické hodnoty 1 na příslušný pin do příslušného Data Direction Registru. K nastavení pinů na výstupní vytvořím funkci *PortsInit*. Funkce neobsahuje žádné argumenty. Jelikož bude funkce zavolána pouze jednou s pevně definovanými hodnotami, nastavení Data Direction Registrů bude následující. V *PORTB* používám všechny piny, proto musím nastavit všechny bity Data Direction Registru do logické hodnoty 1, nastavit je tedy jako výstupní. Ve dvou zbývajících je použita vždy pouze část registru. [20]

```
void PortsInit(void)
        {
        DDRC = 0b11000000;
        DDRA = 0b11111000;
        DDRB = 0b11111111;
        }
```

Po nastavení jednotlivých pinů na výstupní, je možné začít ovládat displej. Pro jednoduchou zkoušku ověření funkčnosti displeje se pokusím otočit pouze jeden konkrétní bod na displeji. K otočení daného terčíku dojde po přivedení proudového impulsu na cívku pod terčíkem. Před přivedením impulsu je nejprve třeba vybrat řádek a sloupec.

Pro účel přivedení proudového impulsu slouží funkce *Write*, která nastaví *PORTC6* na hodnotu 1 po dobu 0,5 *ms*. Po uplynutí doby bude nastaven zpět do

logické hodnoty 0. Význam funkce *_delay_us* spočívá v zajištění délky impulsu. Nejprve se tedy nastaví *PORTC6* na hodnotu 1, poté následuje *_delay_us*, která zařídí mezeru 0,5 *ms* a poté je *PORTC6* přiveden opět na hodnotu 0. Nastavení PORTC6 probíhá přes proměnnou *ENABLE*, která je definována jako (1<<*PC6*).

```
void Write(void)
        {
        PORTC |= ENABLE;
        _delay_us(500);
        PORTC &=~ ENABLE;
        }
```

Nyní ve funkci *main* použiji výše zmíněné funkce. Následujícím kódem je konkrétně vybrán pixel v prvním sloupci a v prvním řádku. Terčík bude otočen žlutou stranou ven.

```
int main(void)
        {
        PortsInit();
        PORTC = 0b00000000;
        PORTB = 0b00000000;
        PORTA = 0b00000000;
        Write();
        }
```

Očekávaným výsledkem nahrání tohoto programu bylo otočení zvoleného pixelu žlutou stranou ven. Očekávaný výsledek však nenastal. Na desce s mikrokontrolérem se nachází LED dioda připojená na napájení 5 *V*. Jelikož LED svítí, napájení by mělo pravděpodobně fungovat. Proměřil jsem výstupy mikrokontroléru, zda správně nastavuji jednotlivé piny. Po proměření všech použitých pinů jsem zjistil, že fungují správně. Napájecí zdroj je rozdělen do dvou částí. Část s výstupním napětím 5 *V* je v pořádku, ale chyba může být v druhé části zdroje. Jelikož jsou body displeje otáčeny pomocí 24 *V*, může být chyba pouze v této části napájení. Na desce jsem místo 24 V naměřil nulové napětí. Zkontroloval jsem pojistky a všechny byly v pořádku. Postupně jsem pokračoval po dalších prvcích. Za konektorem na desku s mikrokontrolérem následuje pojistka, která je v pořádku. Po pojistce následuje stabilizátor napětí 74S24. Na vstupu stabilizátoru jsem naměřil 40 *V*, na výstupu však 0 *V*. Je použit stabilizátor, který má podle dokumentace maximální dovolené vstupní napětí 40 V. [21]

Chyba může být v překročení tohoto napětí, jelikož jsem na vstupu stabilizátoru naměřil 40 *V*, pracuje se na hranici jeho rozsahu. Osadil jsem nový stabilizátor, na odpojeném konektoru napájení (z desky mikrokontroléru) jsem naměřil požadovaná napětí 5, 12 a 24 *V*. Po připojení konektoru a opětovném proměření jsem však zjistil, že je znovu zničený stabilizátor. Chyba tedy není ve stabilizátoru, protože při odpojeném konektoru napájení fungovalo. Odpojil jsem opět konektor, ostatní desky byly propojené, konkrétně desky s dekodéry byly propojeny s deskou mikrokontroléru. Při měření jsem zjistil, že se v obvodech objevuje zkrat mezi 24 *V* a GND. Odpojil jsem desky s dekodéry a zkrat zmizel. Desky s dekodéry jsou celkem dvě, jedna pro záporný směr proudu a jedna pro kladný směr proudu. Po proměření jednotlivých prvků jsem zjistil, že se na obou deskách nachází proražená tranzistorová pole. Tranzistorová pole slouží k volbě řádku. Na jeden řádek jsou vždy připojena dvě tranzistorová pole, jedno pro kladný směr proudu, druhé pro záporný směr proudu. Na jednom řádku jsem objevil dvě proražená pole. Jedno pole bylo proraženo na 24 *V* a druhé na GND. Tím pádem zde vznikal zkrat a docházelo k ničení stabilizátoru 74S24. Při proměření dalších tranzistorových polí jsem objevil celkem tři zničená tranzistorová pole. Dvě typu ULN2803 a jedno typu UDN2982LW. Po osazení nových polí a nahrání programu pro natočení pixelu byl pixel úspěšně otočen.

Nyní se pokusím natočit všechny body. Nejjednodušším způsobem, jak měnit zvolený řádek a sloupec je vytvořit pole hodnot, které se budou postupně zapisovat do *PORTA* a *PORTB*. Tyto hodnoty budou uloženy v polích *Row* a *Column*. Hodnoty jsou zapsány v hexadecimálním tvaru. Každá hodnota představuje nastavení jednoho z registru. Tato pole budou postupně procházena a jejich hodnoty budou zapisovány do příslušných registrů. Z proměnné *Row* do PORTA pro volbu řádku a hodnoty z proměnné *Column* budou zapisovány do *PORTB* pro volbu sloupce. Následující proměnné jsou definovány jako globální, je tedy možno vyčítat z nich hodnoty kdekoliv v programu, což výrazně zjednodušuje řízení displeje.

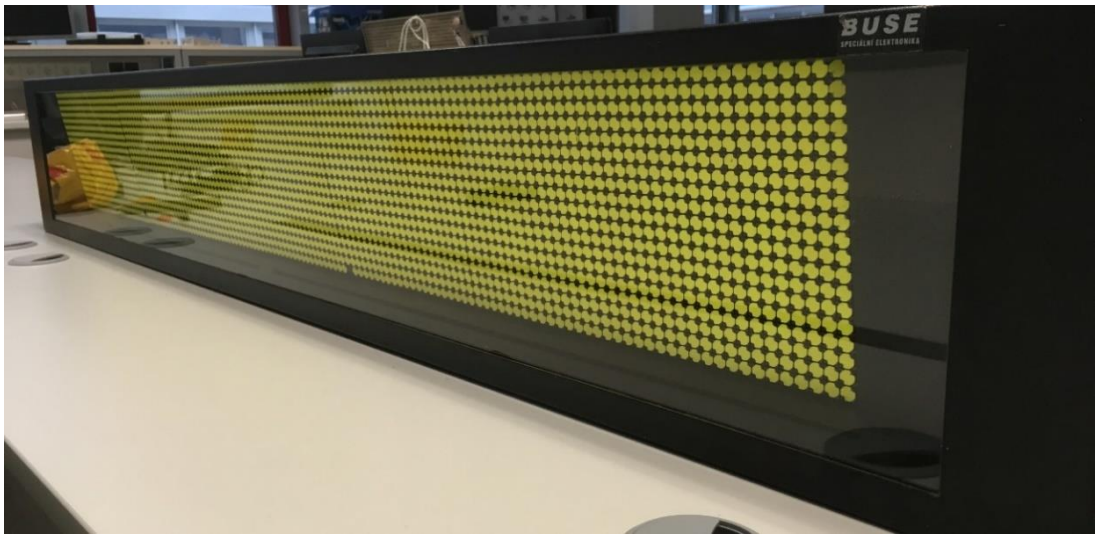
```
static const unsigned char Row[19] =
{ 0x90, 0x88, . . . 0x10, 0x08, 0x00 };
static const unsigned char Column[140] =
{ 0x00, 0x80, . . . 0x99, 0x59, 0xD9 };
```

Ve funkci *main* bude postupně vybíráno z těchto polí a hodnoty budou zapisovány do příslušných registrů. Zápisem registrů se zvolí, jaký pixel má být

natočen a voláním funkce Write se daný pixel natočí. Tím docílím, že se budou postupně otáčet body displeje.

```
int main(void)
	{
	PortsInit();
	PORTC = 0b00000000;
	for (int i = 0; i < 140; i++)
		{
		for (int j = 0; j < 19; j++)
			{
			PORTB = Column(i);
			PORTA = Row(j);
			Write();
			}
		}
	}
```

Po nahrání toho programu dojde k postupném natáčení pixelů na žlutou stranu. Otáčení probíhá po sloupcích. Výsledkem tohoto programu je nastavení všech bodů na žlutou stranu. Z Obr. 4.1 je patrné, že vykreslení proběhlo úspěšně a vykreslily se všechny body. Při vykreslení se vyskytlo pár bodů, které jsou pravděpodobně nefunkční, neboť ani při opakovaném nahrání stejného programu nedošlo k jejich natočení. Jedná se o pixely, které jsou natočeny na černou stranu. Tyto panely jsou náchylné na mechanické opotřebení, jak již bylo zmíněno v kapitole 2.5.



Obr. 4.1 Vykreslený displej

5 NÁVRH PROGRAMOVÉHO VYBAVENÍ

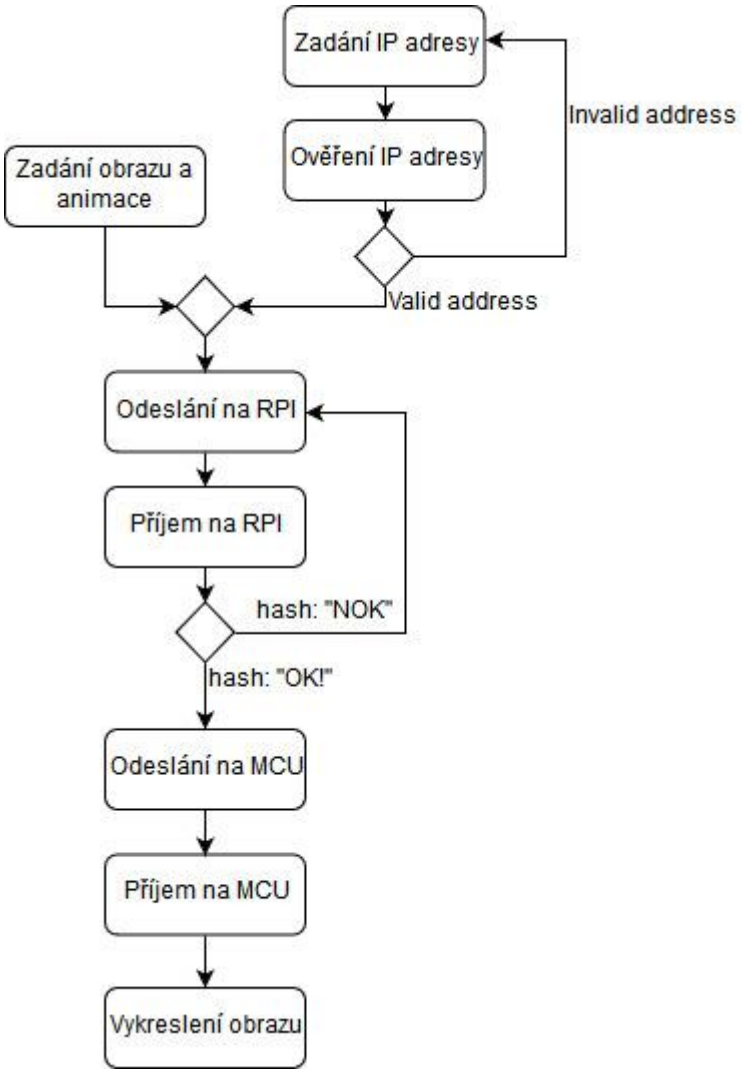
Nejprve je třeba zvolit způsob komunikace s Raspberry Pi. Jak bylo zmíněno výše, při oživování komunikace mezi Raspberry a ATmega128A byl použit ethernet kabel. Pro komunikaci s řídicím systémem by bylo výhodnější použít bezdrátové připojení. Bohužel Raspbbery Pi v používané verzi (Raspberry Pi B) neobsahuje zabudovaný Wi-Fi modul. Ke zprovoznění bezdrátové komunikace bude použit Wi-Fi adaptér zasunutý do USB slotu.

První možností struktury řídicího systému je vytvoření programu, který bude po přístupu na Raspberry Pi spuštěn a bude zahájena obsluha displeje. Přes vzdálený přístup by se přihlásilo na Raspberry a probíhala by obsluha displeje. Tento způsob se mi zdá neefektivní a zdlouhavý. Jiným způsobem je vytvoření ještě dalšího programu, který by byl spuštěn na PC a komunikoval by s Raspberry přes rozhraní Wi-Fi a síťové sockety. Program na Raspberry by byl spuštěný při startu mikropočítače a na základě dat získaných z druhého programu by vytvořil paket a tato data by poslal na mikrokontrolér ATmega128A, který by vykreslil bitmapu na displej. Program v Raspberry Pi by nejprve inicializoval vlákno IP serveru a poté by odchytil příjem dat z PC aplikace a na základě přijatých informací by posílal data na mikrokontrolér. Využiji druhý způsob, který se mi zdá elegantnější, bude stačit připojit Raspberry Pi k napájení a dále se věnovat pouze aplikaci v PC.

Úkolem řídicího sytému je návrh a následné vykreslení obrazovek a animací na maticové zobrazovací jednotce BUSE. Princip funkce toho systému je znázorněn na Obr. 5.1**.** K vykreslení obrazu na displej je nejprve třeba zadat obraz v řídicí aplikaci na PC. V řídicí aplikaci na PC je vykreslena bitmapa pro zadávání obrazu. Pomocí této bitmapy se zadává vstupní obraz, který má být vykreslen. Po vytvoření obrazu a zadání případných informací o animaci je třeba obraz odeslat na zobrazovací jednotku. Odesílání obrazu probíhá ve dvou fázích. Nejprve je obraz odeslán na Raspberry Pi. K odeslání obrazu na Raspberry Pi je nejprve potřeba zadat IP adresu tohoto zařízení, jelikož komunikace mezi PC a mikropočítačem probíhá přes síť. Dále by bylo třeba zadat port, ale ten bude zadán napevno v obou zařízeních. Po přijetí obrazu na Raspberry Pi je v druhé fázi odesílání obraz poslán na ATmega128A, pomocí kterého je následně vykreslen.

K realizaci vykreslení obrazu je třeba vytvořit tři aplikace. Hlavní aplikaci na PC, ze kterého se bude vykreslení řídit. Aplikace na Raspberry Pi, která bude sloužit k přeposílání dat na ATmega a aplikaci na ATmega, přes které bude na základě přijatých dat vykreslen obraz nebo animace na displeji BUSE. Hlavním předpokladem k vykreslení obrazu je vytvoření komunikace mezi třemi prvky systému. Komunikace mezi PC a Raspberry bude probíhat pomocí síťového socketu. Jedná se o koncový bod pro připojení v rámci internetové sítě.

Komunikace mezi mikrokontrolérem a Raspberry bude probíhat přes rozhraní UART. Tato komunikace byla oživena již v rámci kapitoly 4.1, bude tedy pouze rozšířena o přeposílání většího objemu dat.



Obr. 5.1 Stavový diagram řídicího systému

6 APLIKACE NA ŘÍDICÍM PC

Aplikace Win32 je řízena událostmi. V hlavní smyčce programu je při každé interakci (např. stisk klávesnice, kliknutí na tlačítko) přijata zpráva, která je odeslána do funkce *WndProc*. Tato funkce má za úkol události zpracovat. Zpracování událostí probíhá pomocí struktury *switch-case*. Hlavní zpráva obsahuje dva další parametry nazývané *lParam* a *wParam*, ve kterých může být obsažena například pozice myši, informace o klávesnici, identifikace právě stisknutého tlačítka a další. Úkolem aplikace na PC je příjem vstupního obrazu a informací o animaci. Tato data následně převést do paketů a ty odeslat na mikropočítač Raspberry Pi.

Obr. 6.1 Vzhled řídicí aplikace

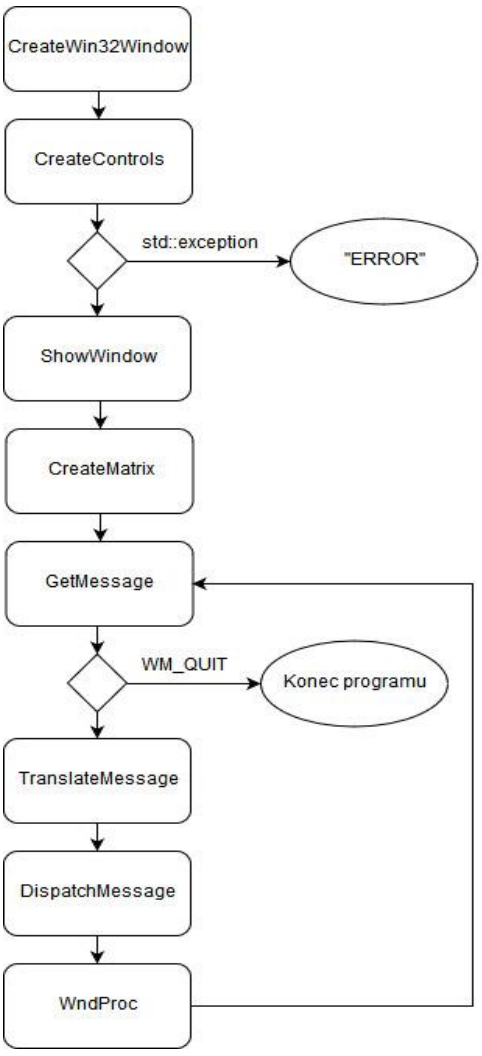
Před odesláním paketů je však nejprve třeba obraz zadat. Zadávání obrazu je umožněno pomocí více vstupů. Mezi tyto vstupy patří myš, klávesnice nebo soubor. K odeslání dat je také třeba zadat IP adresu Raspberry Pi. Číslo portu nebude možné měnit, bude pevně nastaveno v aplikaci. V levém horním rohu aplikace se zadává IP adresa, jejíž formát je následně kontrolován. Výsledek kontroly je indikován pomocí indikátoru, který se nachází na pravé straně od bloku pro zadávání IP adresy. K zadání animace slouží blok prvků v pravém horním rohu. Tento blok se skládá ze tří posuvných prvků a jednoho prvku výběru z hodnot. Posuvnými prvky se nastavuje postupně krok animace, interval mezi posuny, počet opakování posunů a výběrem ze seznamu se volí směr posunu.

Mezi další tlačítka, která jsou umístěna nad vykreslenou bitmapou patří tlačítko *Send Data* sloužící k odeslání dat na Raspberry. Dále *Cursor*, které slouží k posunu kurzoru na vstupní bitmapě. Tlačítko *Cursor* slouží k lepší orientaci při zadávání znaků vstupem z klávesnice. Dalšími dvěma tlačítky jsou *Save* a *Load*.

Jedná se o tlačítka, díky kterým je možno vytvořený obraz uložit do souboru nebo obraz ze souboru načíst. Posledním tlačítkem je tlačítko *Clear*. Toto tlačítko slouží k vymazání celé bitmapy a přesunu kurzoru do základní polohy. Vzhled aplikace je na Obr. 6.1.

6.1 Funkce WinMain

Funkce *WinMain* je hlavní funkcí aplikace. Je volána při startu programu. Z hlavní funkce jsou volány další funkce, které se starají o vykreslení hlavního okna programu, vytváření ovládacích prvků a vykreslení bitmapy pro zadávání obrazu. Dále se v hlavní funkci vytvoří smyčka, které slouží k přeposílání událostí do funkce *WndProc*, kde jsou události dále zpracovávány pomocí *switch–case* struktury. [22]



Obr. 6.2 Stavový diagram funkce WinMain

Hlavní okno aplikace je vytvořeno pomocí funkce *CreateWin32Window*. V této funkci se definuje styl okna, obrázek ikony a kurzoru, pozadí, název okna a velikost okna. Hlavním parametrem této funkce je *HINSTANCE*. Jedná se o 32-bitové číslo, které bylo od operačního systému přiděleno aplikaci po startu. Tento parametr odlišuje danou aplikaci od ostatních aplikací v rámci operačního systému. [23]

Na stavovém diagramu funkce *WinMain*, viz Obr. 6.2, je vytvořena smyčka odesílání událostí pomocí tří funkcí, které se v programu nacházejí uvnitř smyčky *while*. Nejprve je událost zachycena pomocí funkce *GetMessage*, současně je kontrolována návratová hodnota této funkce. Funkce vrací vždy nenulovou hodnotu, pouze při požadavku na zavření aplikace, vrátí funkce nulu. Poté je odeslána událost *WM_QUIT* a aplikace je přes funkci *WndProc* zavřena. Při chybě vrací funkce -1. [24]

Pokud nebyl obdržen požadavek k zavření aplikace, pokračuje se k funkci *TranslateMessage*. Přítomnost této funkce je vyžadována vzhledem k využívání vstupu z klávesnice. Vstup z klávesnice je překládán pomocí tabulky *virtual-key*. [25] Po překladu je zavolána funkce *DispatchMessage*, která odešle přijatou událost spolu s parametry do funkce *WndProc*.

6.2 Funkce CreateControls

Ve funkci *CreateControls* se definují řídicí prvky aplikace, mezi které patří tlačítka, pole pro zadání IP adresy nebo prvky pro zadání animace. Řídící prvky jsou vytvořeny pomocí funkce *CreateWindow*. Mezi parametry této funkce patří typ prvku, název prvku, styl prvku, pozice a rozměry prvku. Dalším parametrem je handle na okno, na kterém se prvek nachází, handle na instanci, jejíž je součástí a handle na menu. [26]

Handle na menu je typu *enum* a je definován pro každý prvek v hlavičkovém souboru *WinMain.h*. Pomocí *enum* parametru je možno dále v programu, při použití jednotlivých prvků, tyto prvky identifikovat, jelikož je předáván v parametrech spolu s událostí. Například při kliknutí na tlačítko je spolu s událostí o stisku tlačítka odeslána identifikace, o jaké tlačítko se jedná. Neskončí-li volání funkcí *CreateControls* a *CreateWin32Window* výjimkou, je okno zobrazeno pomocí funkce *ShowWindow*. Příklad kódu naznačuje vytváření řídicích prvků, tento kód slouží k vytvoření tlačítka *Send*.

```
if (CreateWindow("Button", "Send", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE,
50, 210, 100, 25, hWnd, (HMENU)IDC_BUTTON_SEND, hInstance, NULL) == NULL)
    throw std::exception("Failed to create controls");
```

Tlačítko je vytvořeno na pozici *[50,210]* má rozměry *100x25*. Tlačítko je součástí hlavního okna *hWnd*, při stisku tlačítka bude v parametru *wParam* hodnota *IDC_BUTTON_SEND*. Obdobně se vytvářejí i další tlačítka a další prvky. Vytvoření prvku je kontrolováno podmínkou, pokud by funkce *CreateWindow* selhala, je vyhozena výjimka, která vypíše chybovou hlášku a ukončí program. Takto je kontrolováno vytvoření všech řídicích prvků.

6.3 Funkce WndProc

Jak již bylo zmíněno, aplikace je řízena událostmi. Při ovládání aplikace dochází ke generování událostí. Díky jejich zpracování ve funkci *WndProc*, je aplikace řízena. Nejprve je nutno rozhodnout, co je v události uloženo. K tomu odchází pomocí několika *switch-case* struktur. Hlavní struktura rozhoduje o obsahu hlavní zprávy, tedy proměnné s názvem *uMsg*. Ve funkci je odchytáváno hned několik těchto zpráv. Mezi nejdůležitější patří *WM_QUIT*, která je generována při požadavku na zavření aplikace. Po odchycení této zprávy je tedy celá aplikace zavřena.

Další zprávou, která je odchytávána především kvůli grafickému prvku bitmapy, je *WM_PAINT.* Tato událost je generována při požadavku na překreslení okna aplikace. Pokud by byla aplikace posunuta a nedocházelo by k odchytávání této události, došlo by k deformaci bitmapy. Při odchycení zprávy je tedy bitmapa znovu vykreslena na základě proměnné *Data*, ve které se nachází navržená obrazovka.

Při stisku levého tlačítka myši je zachycena událost *WM_LBUTTONDOWN*. Počítačová myš slouží k ovládání aplikace, ale také k návrhu obrazovek. Při zachycení této zprávy, jsou v pomocném parametru s názvem *lParam* uloženy souřadnice pozice kurzoru myši. Souřadnice se vztahují k celému oknu aplikace. V levém horním rohu aplikace je tedy počátek tohoto souřadnicového systému. Při kliknutí na bitmapu dochází po přepočtu souřadnic ke změně pixelu bitmapy.

K návrhu obrazovek slouží také vstup z klávesnice. Při detekci stisku některé klávesy je zachycena událost *WM_KEYDOWN*. V parametru s názvem *wParam* je pak uložen hexadecimální kód klávesnice, který byl získán překladem pomocí funkce *TranslateMessage*. Při odchycení události je tedy dále nutno rozhodnout, o jakou klávesu se jednalo. Jedná-li se o klávesnici se znakem, je na pozici, kterou určuje kurzor nacházející se na bitmapě, znak vykreslen. Pomocí kláves, mezi které patří například šipky nebo mezerník, dochází k posunu kurzoru.

Další událostí, která je odchytávána hlavní *switch-case* strukturou, je *WM_COMMAND*. Ke generování této zprávy dochází při změně pole pro zadávání textu nebo při stisknutí některého z implementovaných tlačítek. Tímto způsobem je tedy například získávána IP adresa, která se v aplikaci zadává. Dalšími prvky, které generují tuto zprávu, jsou prvky sloužící k nastavení animace. Po změně

hodnot v textových polích v bloku pro nastavení animace je tedy generována událost *WM_COMMAND* a v parametru *wParam* je uložena identifikace prvku, díky kterému došlo ke generování této zprávy. V následujícím kódu je znázorněno zpracování události pro detekci změny hodnoty v textovém poli s velikostí kroku animace.

```
if(HIWORD (wParam) == EN_UPDATE && LOWORD (wParam) == IDC_EDIT_STEP)
   {
   GetWindowText (GetDlgItem (hWnd, IDC_EDIT_STEP), (LPSTR)Step, 3);
   SetScrollPos(GetDlgItem(hWnd,IDC_BAR_STEP),SB_CTL,TextToInt(Step),TRUE);
   }
```

Textové pole je identifikováno parametrem *IDC_EDIT_STEP*. Pomocí parametru *EN_UPDATE* je indikováno, že došlo ke změně hodnoty v tomto poli. Zadaná hodnota je funkcí *GetWindowText* uložena do příslušné proměnné (*Step*). Hodnota je uložená v proměnné typu *string*. Následující funkcí by mělo dojít ke změně pozice posuvného prvku. Pozice je však nutno zadat v proměnné typu *int*. Proto byla vytvořena funkce, která zajistí tento převod. Vstupní parametr funkce *TextToInt* je ukazatel na proměnnou typu *string*, návratová hodnota je již typu *int*. Jak bude zmíněno dále v textu, k nastavení animace slouží tři typy zadání hodnot. Mezi ně patří zadání hodnoty číselně, k čemuž slouží výše popsaný kód. Dalším způsobem je změna pozice prvku. Ke změně pozice dojde buď posunem nebo pomocí šipek, které se nacházejí na obou stranách tohoto prvku, viz Obr. 7.2. Při posunu prvku je odchycena funkcí *WndProc* zpráva *WM_HSCROLL*. Prvek je posunut a do pole je zapsána příslušná hodnota.

7 NÁVRH OBRAZOVEK

V řídicí aplikaci na PC je vykreslena bitmapa, viz Obr. 6.1. Rozlišení bitmapy odpovídá rozlišení panelu. Binární hodnota každého bodu bitmapy je uložena v globální proměnné pole s názvem *Data*. Změnou pixelu v bitmapě se zároveň změní hodnota proměnné *Data* na příslušné pozici. Význam této proměnné je uložení obrazu k následnému odesílání na Raspberry Pi. Při spuštění aplikace je bitmapa prázdná. K vymazání celé bitmapy slouží tlačítko *Clear*, jehož stisknutím dojde zároveň k vymazání proměnné *Data*, tedy nastavení všech prvků na nulové hodnoty.

Návrh obrazovek je umožněn zadáváním znaků přes klávesnici, změnou jednotlivých pixelů pomocí myši nebo nahráním celého obrazu ze souboru. Obraz musí být v souboru uložen v přesně definovaném formátu, aby došlo ke korektnímu nahrání do bitmapy implementované v aplikaci

Na bitmapě je zobrazen i kurzor, který slouží k lepší orientaci při zadávání znaků pomocí klávesnice. Po zapsání znaku je kurzor posunut na novou pozici, na kterou bude zobrazen další znak. K posunu kurzoru dochází buď při zapsání nebo smazání znaku. Další možností posunu jsou šipky. Velikost posunutí kurzoru je rovna počtu pixelů, které zabere znak v daném směru. K posunu kurzoru na libovolné místo slouží tlačítko *Cursor*. Při kliku na tlačítko je rozhodnuto, že má při následujícím kliku na vykreslenou bitmapu dojít k posunu kurzoru, nikoliv ke změně barvy pixelu. K umožnění návrhu obrazovek musí být však bitmapa nejprve v aplikaci vykreslena.

7.1 Vykreslení bitmapy

Při startu aplikace je po zobrazení okna a všech řídicích prvků vykreslena bitmapa obrazovky pomocí funkce *CreateBM*. Bitmapa patří mezi grafické objekty. Před použitím grafického objektu je nutno získat kontext zařízení, tzv. *Device Context* pomocí funkce *GetDC*. K vykreslení grafického objektu je třeba vytvoření kompatibilního kontextu zařízení typu *HDC* pomocí funkce *CreateCompatibleDC* na základě dat z funkce *GetDC*, která získává informace o daném zařízení. Samotná bitmapa bude vytvořena v proměnné typu *HBITMAP*. [27]

```
HDC hDC = GetDC (hBitmap);
HDC hCompDC = CreateCompatibleDC (hDC);
HBITMAP hBMP;
```

Vytvoření vstupní bitmapy probíhá pomocí funkce *CreateBitmap*. Bitmapa je tvořena na základě vstupního pole. Vstupním polem jsou hodnoty z proměnné *Data*, převedené do datového typu *UINT32.* [28]

```
hBMP = CreateBitmap (140, 19, 1, 32, &Data_U32);
hBMP = (HBITMAP)SelectObject (hCompDC, hBMP);
```

Rozměry obrazovky jsou 140:19. Při vykreslení v tomto rozlišení by byla bitmapa nečitelná. Proto jsou rozměry bitmapy zvětšeny pomocí měřítka, tedy jeden prvek bitmapy je složen z několika pixelů obrazovky počítače. K vykreslení zvětšené bitmapy slouží funkce *StretchBlt*. [29] Prvních pět vstupních parametrů se týká výsledné bitmapy, druhých pět se týká vstupní bitmapy. Postupně se jedná o handle *DC*, pozici bitmapy a rozměry bitmapy. U parametrů z druhé části neznačí první dva indexy, kam se má bitmapa vykreslit, ale naopak od jaké pozice původní bitmapy se mají brát data.

```
StretchBlt (hDC, 1, 1, 140*Scale, 19*Scale, hMemDC, 0, 0, 140, 19, SRCCOPY);
```

7.2 Návrh obrazu pomocí myši

Prvním možným způsobem návrhu obrazovky je pomocí počítačové myši. Nacházel-li se kurzor myši nad zobrazenou bitmapou, je po stisku levého tlačítka myši změněn bod bitmapy, který se pod kurzorem nacházel. Při zmáčknutí levého tlačítka myši je do funkce *WndProc* odeslána událost *WM_LBUTTONDOWN* a spolu s ní je v parametru *lParam* uložena informace o pozici kurzoru myši. Při zpracování této události ve *switch-case* struktuře je nejprve ověřena pozice kurzoru myši. Ke změně pixelu bitmapy se musí myš nacházet nad vykreslenou bitmapou. Pomocí maker *GET_X_LPARAM* a *GET_Y_LPARAM* jsou z parametru *lParam* zjištěny souřadnice, které je nejprve nutno přepočítat. Souřadnice jsou totiž z maker vráceny vzhledem k hlavnímu oknu aplikace. Výsledná souřadnice je následně převedena na indexy pole pomocí měřítka, které bylo použito ke zvětšení rozměrů bitmapy. [30]

```
int x = (GET_X_LPARAM (lParam) - 20) / Scale;
int y = (GET_Y_LPARAM (lParam) - 250) / Scale;
```

Hlavní funkcí, která slouží ke změně pixelu, je funkce *ChangePixel*. Funkce *ChangePixel* funguje obdobně jako funkce *CreateBM*. V předchozí podkapitole bylo zmíněno, že ve funkci *StretchBlt* je možno zvolit pozici a velikost bitmapy. Ve funkci ChangePixel bude tedy změněn pouze ten bod bitmapy, který se nachází na

souřadnicích získaných z parametru *lParam*. Při kliku myši je funkci *ChangePixel* spolu se souřadnicemi bodu předána i výsledná barva daného pixelu. Barva, na kterou se má daný bod změnit, je zjištěna z proměnné *Data*. Při návrhu obrazu pomocí myši se tak vždy jedná o inverzní hodnotu původní hodnoty na těchto souřadnicích. Po změně pixelu na novou barvu je zároveň aktualizována hodnota proměnné *Data* na příslušné pozici.

```
StretchBlt (hDC, x*Scale + 1, y*Scale + 1, Scale, Scale, hMemDC, 0, 0, 1, 1, SRCCOPY);
```

7.3 Návrh obrazu pomocí klávesnice

Další možností vstupu je zadávání znaků pomocí klávesnice. Po zmáčknutí znaku na klávesnici je daný znak přeložen přes tabulku virtuální znaků, kde je každý znak reprezentován hexadecimální hodnotou. Po překladu, ke kterému slouží funkce *TranslateMessage*, je ve funkci *WndProc* rozhodnuto, o jaké tlačítko se jedná. [31] Poté je znak zobrazen na vstupní bitmapě a kurzor je posunut automaticky na novou pozici. V aplikaci je pro každý znak vytvořeno předdefinované pole hodnot pevných rozměrů. Znaky jsou definovány v polích o rozměrech 9x5 pixelů. Vzhledem k rozměrům displeje jsem přizpůsobil velikost znaků zadávání do dvou řádků. V aplikace jsou implementovány jednak znaky, ale také další tlačítka, mezi které patří enter, mezera, delete a šipky pro posun kurzoru.

Kurzor je na bitmapě odlišen od ostatních pixelů. Má bílou barvu a jeho poloha je při spuštění aplikace nebo vymazání bitmapy na začátku prvního řádku. Jak bylo zmíněno v úvodu této kapitoly, ke změně polohy kurzoru dochází pomocí šipky a tlačítka *Cursor*. Kliknutí na tlačítko *Cursor* nastaví globální proměnnou *TextPressed* do logické 1. Poté jsou vybrány souřadnice pixelu, na kterých se má nově nacházet kurzor. Pozice kurzoru je uchovávána v globálních proměnných *Pos_x* a *Pos_y*. Tyto hodnoty se mění vždy s posunem kurzoru.

Po přesunu kurzoru na novou pozici je třeba nastavit hodnotu předešlé pozice kurzoru z bílé barvy na správnou hodnotu. K tomuto účelu slouží proměnná *LastValue*, která danou hodnotu uchovává. Pokud byla barva pixelu na předešlé pozici žlutá, je v proměnné *LastValue* logická hodnota 1. Pokud byla barva černá, je v ní uložena logická hodnota 0. Po přesunu kurzoru je proměnná *TextPressed* opět vynulována na logickou hodnotu 0 a na předešlou pozici je zapsána hodnota z proměnné *LastValue*. V následující části kódu je tedy nejprve vyhodnocena poslední hodnota pixelu na současné pozici kurzoru. Daná hodnota je na pozici vykreslena. Po vykreslení pixelu na správnou barvu je do proměnné *LastValue* uložena hodnota nové pozice kurzoru. Na tuto pozici je vykreslen kurzor a poté je aktualizována pozice kurzoru v proměnných *Pos_x* a *Pos_y*. Nakonec je vynulována

proměnná *TextPressed*. Vynulování hodnoty této proměnné dochází zároveň při kliknutí mimo vykreslenou bitmapu. Tímto je negována funkce tlačítka *Cursor*.

```
if(TextPressed)
        {
        if(LastValue)
                ChangePixel(Pos_x, Pos_y, 0xffff00);
        else
                ChangePixel(Pos_x, Pos_y, 0x00000000);
        LastValue = Data[x][y];
        ChangePixel(x, y, 0xffffffff);
        Pos_x = x;
        Pos_y = y;
        TextPressed = 0;
        }
```

Druhým způsobem posunu kurzoru jsou tlačítka šipek. Je-li pomocí šipek kurzor posunut na novou pozici, je opět ukládána hodnota pixelu nové pozice a barva pixelu na předešlé pozici je změněna zpět na správnou hodnotu.

K mazání znaků slouží tlačítka *Backspace* a *Delete*. Funkce tlačítka *Backspace* je smazání znaku a posunutí kurzoru o jednu pozici (5 pixelů) vlevo. Pomocí tlačítka *Delete* je smazáno písmeno a v této aplikaci dojde k posunutí kurzoru o jednu pozici doprava. Posledními tlačítky jsou enter a mezerník. Mezerník slouží k posunu kurzoru o jednu pozici doprava. Nacházel-li se na předešlé pozici znak, je po stisku klávesy mezerník smazán. Enter slouží k posunu kurzoru na nový řádek. Jelikož má displej malý počet řádků, je kvůli čitelnosti umožněno zobrazit pouze dva řádky textu.

Vstup z klávesnice využívá více prvků. Je využit jednak při návrhu obrazovky, ale také při zadávání informací do tzv. *Edit Controls*. To jsou prvky, které slouží k nastavení animace nebo IP adresy. Proto je nutné rozhodnout, který prvek v daný moment vstup z klávesnice využívá. Je-li zvolen některý z *Edit Controlů*, je vstup z klávesnice přiveden na něj, a tudíž není možný návrh obrazovky. Aby bylo možno pomocí klávesnice opět navrhovat obrazovku, je zavolána funkce *SetFocus*. Tato funkce má jako vstupní parametr proměnou *HWND*, která definuje okno, kterému má být přidělen vstup z klávesnice. Při stisku levého tlačítka myši s kurzorem myši mimo *Edit Control*, je nastaven *focus* na hlavní okno aplikace, a tím je možno využít klávesnici k návrhu obrazovek a zadávání znaků.

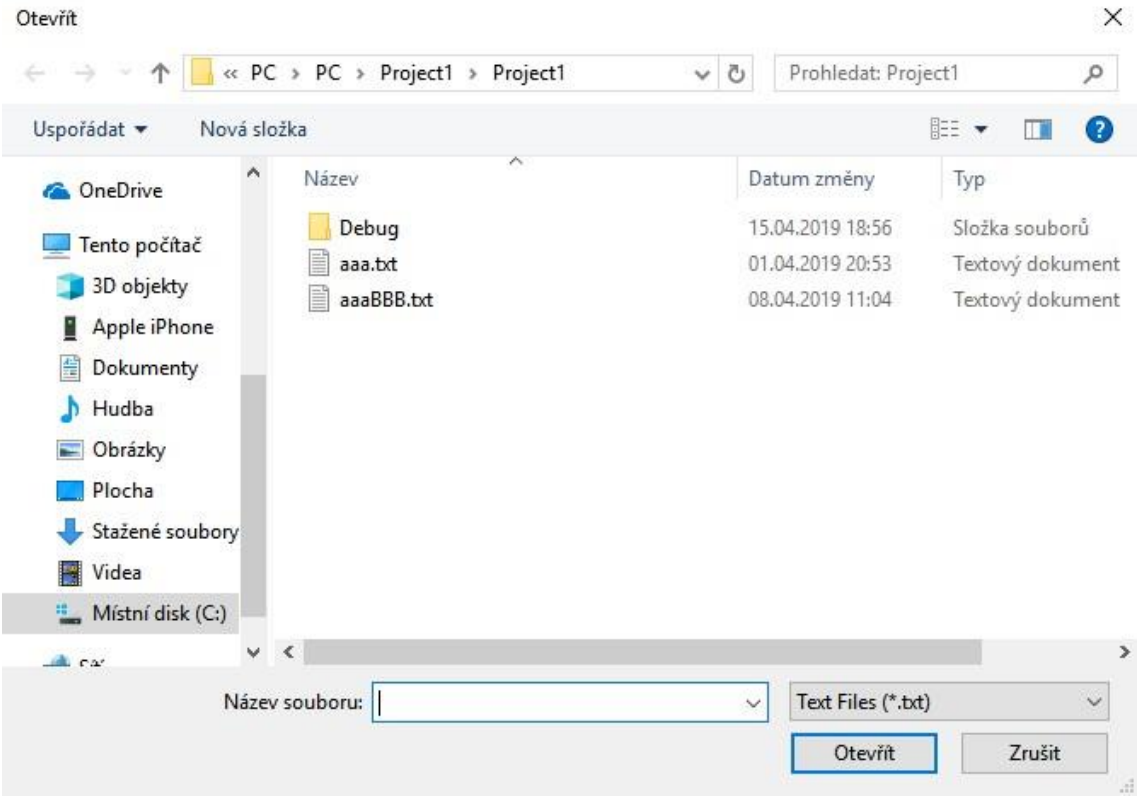
7.4 Vstup ze souboru

Poslední možností návrhu obrazovky je načtení dat ze souboru. Data v souboru však musí být validní. Musí splňovat předem definovaný formát, kterým jsou hodnoty pixelů oddělené pomocí čárek. Rozložení znaků v souboru odpovídá rozložení displeje, tedy na samostatném řádku v souboru musí být uložen vždy jeden řádek obrazu. Poslední znak na řádku není již oddělen čárkou. Za posledním znakem v řádku následuje znak pro odřádkování.

Pro vstup ze souboru slouží tlačítko *Load*. Při práci se souborem je nejprve třeba daný soubor otevřít nebo vytvořit. Při načítání ze souboru se bude vždy otevírat již existující soubor. Existuje více typů otevření souboru. Soubor lze otevřít s právy pro čtení, zápis nebo pro čtení i zápis. Při načítání obrazu ze souboru je vhodné, aby byla zvolena pouze možnost čtení. Nedojde tak k přepsání informací ve vstupním souboru. Otevírání souboru jsem rozdělil do více částí. Nejprve je třeba zvolit, který soubor má být otevřen. Vhodným řešením volby souboru je otevření dialogového okna s průzkumníkem souborů, pomocí kterého se soubor zvolí, viz Obr. 7.1. Informace zjištěné z dialogového okna, především tedy cesta k souboru, se ukládají do proměnné typu *OPENFILENAME*. Pro inicializování této proměnné pomocí dialogového okna slouží funkce *GetOpenFileName*. Pokud byl úspěšně zvolen soubor a volba potvrzena, je možné daný soubor otevřít. K otevření souboru slouží funkce *CreateFile*. Zde zmíním dva důležité parametry funkce. Jsou jimi oprávnění přístupu a akce, která má být provedena se souborem. Pro načtení obrazu volím *GENERIC_READ*, tedy práva pouze ke čtení, a *OPEN_EXISTING*. Pomocí druhého parametru je definováno, že se bude otevírat pouze již existující soubor. [32]

```
HANDLE hFile = CreateFile (szFileName, GENERIC_READ, 0, NULL,
                    OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

Soubor je nyní otevřen a lze z něj číst. Čtení souboru bude probíhat pomocí funkce *ReadFile*. Tato funkce potřebuje znát velikost souboru, proto je velikost zjištěna a zároveň použita k alokování paměti, která bude potřeba k načtení dat ze souboru. Formát zapsaných dat jsou hodnoty jednotlivých pixelů (hodnota 1 nebo 0) oddělené čárkami. Vzhledem ke známému formátu je soubor postupně procházen a hodnoty na daných pozicích jsou zapisovány do globální proměnné *Data*. Po načtení hodnot do proměnné je obraz vykreslen do bitmapy.



Obr. 7.1 Dialogové okno pro načtení ze souboru

7.5 Uložení obrazu do souboru

Podobně jako načítání dat ze souboru je aplikace schopna obraz do souboru uložit. Ukládání dat do souboru je umožněno pomocí tlačítka *Save*. Princip ukládání souboru je podobný principu čtení. Opět je otevřeno dialogové okno, avšak nyní k účelu vytvoření nového nebo ke zvolení již existujícího souboru, do kterého se mají data uložit. Pokud bude zvolen již existující soubor, bude tento soubor přepsán novými daty. Po zvolení souboru je soubor opět otevřen pomocí funkce *CreateFile*. Výše zmíněné vytvoření nebo přepsání souboru je dosaženo pomocí parametru *CREATE_ALWAYS*. [33] Přístupová práva budou pouze pro zápis, jelikož není potřeba ze souboru nic číst.

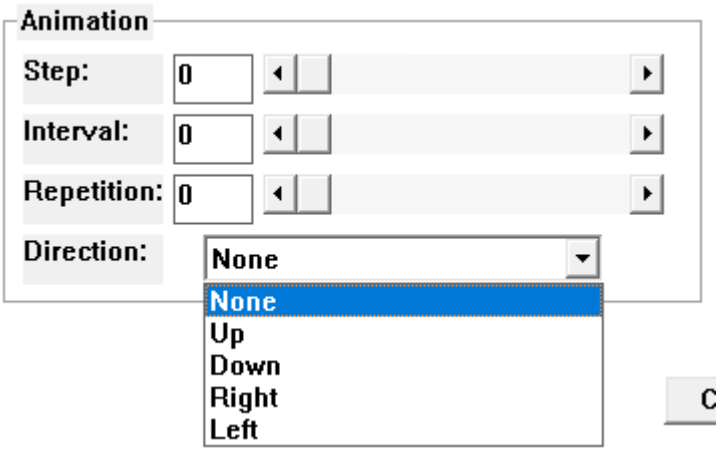
```
HANDLE hFile = CreateFile (szFileName, GENERIC_WRITE, 0, NULL,
                    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
```

Samotný zápis dat probíhá postupným procházením proměnné *Data* a rozhodování o hodnotách jednotlivých pixelů. Pokud je pixel žlutý, zapíše se hodnota 1, pokud je černý, zapíše se na danou pozici hodnota 0. Jednotlivé hodnoty jsou opět oddělené čárkami. Po zapsání celého řádku je za posledním znakem zapsán místo oddělovače čárky znak k odřádkování „\n“.

```
for (int j = 0; j < 19; j++)
        {
        for (int i = 0; i < 140; i++)
         {
         if (Data[i][j])
                  WriteFile(hFile, "1", (DWORD)strlen ("1"), &dwBytesWritten, NULL);
         else
                  WriteFile(hFile, "0", (DWORD)strlen ("0"), &dwBytesWritten, NULL);
         if (i < 139)
                  WriteFile(hFile, ",", (DWORD)strlen (","), &dwBytesWritten, NULL);
         }
        WriteFile (hFile, "\n", (DWORD)strlen ("\n"), &dwBytesWritten, NULL);
        }
```

7.6 Návrh animací

Systém je schopen vykreslit jednoduchou animaci. Animací se rozumí posun vykresleného obrazu v horizontálním nebo vertikálním směru. Vzhledem ke zjednodušení odesílaní a přijímání dat, obsahuje i statický obraz informace o animaci. Zda má nebo nemá dojít k animaci je rozhodnuto při čtení informace o směru animace. Pokud je v tomto parametru hodnota *None*, k animaci nedochází.



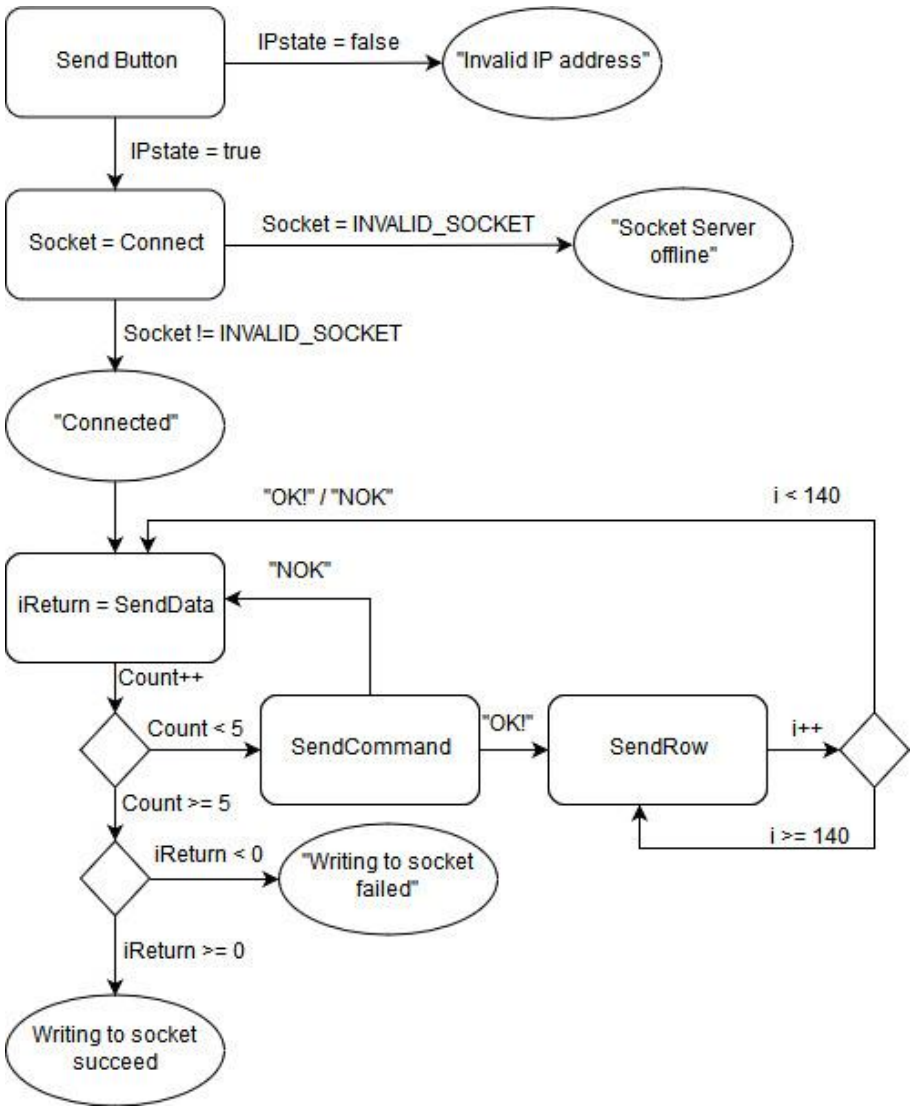
Obr. 7.2 Prvky pro návrh animace

Mezi informace o animaci patří kromě směru animace také velikost posunu, interval mezi jednotlivými posuny a počet opakování posunu. Tyto hodnoty je v aplikaci možno nastavit v pravém horním rohu celkem třemi způsoby. Pomocí posunu prvku, pomocí šipek u řídicího prvku nebo pomocí zapsání hodnoty do příslušného pole. Výběr směru animace probíhá výběrem ze seznamu, viz Obr. 7.2.

Posuvný prvek je tvořen jako ostatní řídicí prvky funkcí *CreateWindow*. Navíc je třeba tomuto prvku nastavit rozsah. Při volbě rozsahu jednotlivých informací jsem musel zvážit rozměry displeje a rychlost vykreslování. Krok animace je nastaven v rozsahu 0-20 pixelů. Aplikace by sice byla schopna poslat i krok většího rozsahu, avšak při šířce 140 pixelů a především výšce 19 pixelů nemá větší rozsah smysl. Interval animace je možno nastavit po kroku v rozsahu 0-255 *ms*. Počet opakování má stejný rozsah jako interval, tedy 0-255 opakování. Tři typy zadávání těchto hodnot jsou zpětně kompatibilní, tedy posun pomocí šipek posune prvek o jednu pozici a zároveň změní číslo o hodnotu 1. Zápisem čísla se zpětně posune prvek na příslušnou pozici a posunem prvku dojde i ke změně číselné hodnoty.

8 KOMUNIKACE MEZI RASPBERRY PI A PC

K odeslání dat je implementováno tlačítko *Send*. Před stiskem tlačítka *Send* je nejprve třeba korektně zadat IP adresu Raspberry Pi. Po kliknutí na tlačítko *Send* se aplikace na Raspberry Pi připojí, odešle data a odpojí. V této aplikaci bude Raspberry Pi sloužit jako server a PC bude sloužit jako klient, který se bude na server připojovat. Ke korektnímu zadání IP adresy slouží kontrola, která kontroluje formát zadané IP adresy. Přesněji kontroluje, zda došlo k zadání 4 čísel oddělených tečkami v rozsahu 0-255. Proběhne-li zadání IP adresy a připojení na Raspberry bez komplikací, přejde se k odesílání dat. Princip odesílání dat z řídicí aplikace na Raspberry Pi je zobrazen na stavovém diagramu na Obr. 8.1.



Obr. 8.1 Stavový diagram odesílání dat

Odesílání dat začíná nultým paketem, tedy odesláním informací o animaci. Poté se čeká na zpětnou vazbu z Raspberry Pi, ze kterého je zpětně odeslána kontrola, zda se shoduje odeslaný a nově vypočtený *hash* kód. Pokud přijde informace *NOK*, opakuje se odesílání nultého paketu. Pokud přijde *OK!*, přejde se k odesílání obrazu. Je provedeno celkem pět pokusů o odeslání dat, po pěti neúspěšných pokusech je vypsána chybová hláška a odesílání dat je ukončeno. Při odesílání obrazu probíhá opět zpětná kontrola. Kontrola však probíhá až po odeslání všech 140 paketů, jelikož kontrola po odeslání jednotlivých paketů by příliš zpomalovala proces odesílání. Přijde-li z Raspberry Pi pětkrát negativní kontrola *hash* kódu, je odesílání ukončeno a je zobrazena zpráva o neúspěšném poslání obrazu.

8.1 Síťový socket

Komunikace mezi PC a Raspberry bude probíhat pomocí tzv. síťového *socketu*. Raspberry Pi bude v této aplikaci sloužit jako server. Bude na něm tedy vytvořen hlavní *socket*, kterému bude přiřazen port. Na tomto portu bude očekávat připojení ze strany PC. Na straně PC bude zadán port a IP adresa serveru a bude proveden pokus o připojení na server. Pokud server zaznamená pokus o připojení, vytvoří se nový *socket*, na kterém bude mezi PC a Raspberry docházet k výměně dat. Po dokončení výměny dat bude nově vytvořený *socket* uzavřen a PC bude od Raspberry odpojeno. Při odeslání dalších dat budě opět vytvořen nový *socket*, přes který bude docházet k odesílání nových dat.

8.1.1 Socket server na Raspberry Pi

K vytvoření serveru na platformě Linux slouží sada funkcí z knihovny *socket.h*. Před vytvořením hlavního *socket*, na kterém se bude přijímat připojení od PC, je nejprve třeba zadat jeho typ. Pro komunikaci s PC volím typ adresy IPv4, protokol TCP a formát přenosu dat je stream. Při volení parametrů jsem se rozhodoval mezi dvěma formáty přenosu dat. První možností byl datagram, který je rychlejší, je však větší šance, že odeslaná data nedorazí. Druhou volbou byl stream, který je obousměrný, je pomalejší, ale zajištěna větší spolehlivost odeslání dat, proto byl zvolen formát stream.

Na následujícím kódu je demonstrováno vytvoření *socket* s výše zmíněnými parametry. Po vytvoření je třeba k *socket* přiřadit port, na kterém se bude čekat na příchozí spojení. Slouží k tomu funkce *bind*. V proměnné *address* je jednak nastaven typ na IPv4, ale také je v ní napevno nastaven port, na kterém bude očekáváno příchozí spojení ze strany PC. Po přidělení portu se zavolá funkce *listen*, která čeká na příchozí připojení ze strany PC.

```
fdSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
bind(fdSocket, (struct sockaddr *) &address, sizeof(address)
listen(fdSocket,5);
```

V hlavní smyčce programu se čeká na pokus o připojení. Jakmile je detekováno připojení, je pomocí funkce *accept* vytvořen nový *socket*, na kterém bude probíhat přenos dat. Poté dojde k přenosu dat. Nejprve jsou data přijata z aplikace v PC. Došlo-li ke korektnímu přijetí dat, jsou data odeslána na mikrokontrolér ATmega. Po dokončení přenosu dat je nově vytvořený *socket* uzavřen.

8.1.2 Socket klient na PC

Před použitím síťového *socket* je nejprve třeba inicializovat *Winsock*. Po inicializování pomocí funkce *WSAStartup* je možno funkce z *Winsock* používat. Pomocí funkce *WSAStarup* se volí verze *Winsock*. Pokud funkce vrátí nulovou návratovou hodnotu, je detekováno, že daný počítače nepodporuje zvolenou verzi *Winsock*. [34]

Postup práce se *socket* ze strany klienta je podobný jako při vytváření serveru na platformě Linux. Nejprve je vytvořen *socket* se stejnými parametry, jako má server. Po vytvoření *socket* je proveden pokus o připojení k Raspberry na předem definovaném portu a IP adrese, která byla zadaná do příslušného bloku v aplikaci. Port je definován na pevně zvolenou hodnotu a po spuštění aplikace nebude možné ho měnit. Port je do funkce *connect* předán proměnnou *AddrInfo.* Tato proměnná je obdobou proměnné s názvem *address* v přechozí kapitole u Raspberry Pi.

```
#define DEFAULT_PORT 27015
```

Pokud neskončí volání funkcí *socket* a *connect* chybou, je vytvořeno spojení mezi PC a Rasberry Pi a může se přejít k odesílání dat.

```
AddrInfo.sin_family = AF_INET;
AddrInfo.sin_port = htons(DEFAULT_PORT);
AddrInfo.sin_addr.s_addr = inet_addr(iAddress);

fdCliSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
connect(fdCliSock, (SOCKADDR*)&AddrInfo, sizeof(AddrInfo);
```

Pokus o připojení je proveden pouze za předpokladu, že byla zadaná validní IP adresa. Po úspěšné připojení k serveru je provedeno odeslání dat, tedy nejprve odeslání informací o animaci a poté odeslání obrazu. Tato posloupnost funkcí je provedena po kliknutí na tlačítko *Send*.

8.2 Formát paketu

Při definování formátu paketu pro odesílání obrazu bylo nejprve nutno zvážit velikost obrazu. K odesílání obrazu bude sloužit funkce, která je schopna odeslat data o velikosti 8 bitů, stejně tak rozhraní UART je schopno odesílat data o velikosti 8 bitů. Obraz bude odesílán po sloupcích. Odeslání informace o obrazu zabere tedy 19 bitů, které budou rozděleny do tří částí a to po 7, 7 a 5 bitech. Další informací bude číslo paketu neboli číslo sloupce. Informace o animaci jsou v nultém paketu. Poslední informací, kterou bude obsahovat paket bude hodnota vypočítaného *hash* kódu. Formát paketu s obrazem je zobrazen v Tabulka 8.1.

Tabulka 8.1 Formát paketů s obrazem

0	1	2	3	4	5
Číslo	Data 0-6	Data 7-13	Data 14-18	Hash 0-7	Hash 8-15

Paket s animací bude odesílán v uspořádání viz Tabulka 8.2. Formát paketu nesoucího informaci o animaci obsahuje číslo paketu, směr posunu, velikost posunutí v pixelech, interval mezi jednotlivými posuny v *milisekundách* a počet opakování posunu. Poslední informací je opět vypočtený *hash* kód.

Tabulka 8.2 Formát paketu s animací

0	1	2	3	4	5	6
Číslo	Směr	Krok	Interval	Opakování	Hash 0-7	Hash 8-15

Pro kontrolu správnosti odeslaných dat byl zvolen výpočet *hash* kódu. Kvůli přesnosti tohoto výpočtu je pro *hash* kód v paketu vyhrazeno 16 bitů. *Hash* kód se vypočítá jako postupné násobení jednotlivých osmic bitů, a vždy se předchozí součin přičte k novému. *Hash* kód se stejným způsobem vypočítá i na druhém zařízení a jejich následným porovnáním lze zjistit, zda byla data odeslána v pořádku.

8.3 Odesílání dat

Odesílání dat začíná nultým paketem, který obsahuje informace o animaci. Za nultým paketem následuje celkem 140 paketů s obrazem. Každý paket nese

informaci o jednom sloupci. K odesílání dat slouží funkce *send*. Po odeslání všech paketů se čeká na odpověď z Raspberry. Pokud se u všech paketů shodoval *hash* kód, vrátí se „OK!“. Pokud se vyskytla chyba v přepočtu *hash* kódu u jednoho a více paketů, přijde zpráva „NOK“ a odesílání dat probíhá znovu. Celkem je provedeno pět pokusů o odeslání dat, poté je zobrazeno dialogové okno o neúspěšném odeslání obrazu. Jinou možností by byla kontrola každého odeslaného paketu, avšak tento princip by velice zpomalil odesílání dat, proto jsem zvolil kontrolu vždy na konci odeslání všech paketů.

Na Raspberry Pi je po zaznamenání pokusu o připojení ze strany PC vytvořen nový *socket* a je zavolána funkce *receiveData*. Ve funkci jsou přijímána data z PC pomocí smyčky *while*, ve které se zavolá funkce na přijetí paketu s animací a poté funkce na přijetí paketů s obrazem. Pro příjem těchto informací slouží dvě funkce, protože obě informace jsou odesílány v různě dlouhých paketech.

Přijatá data se vloží do zásobníku. Při přijímání animace se kontroluje první prvek zásobníku, jelikož se jedná o číslo paketu a u animace by mělo být toto číslo rovno nule. Dále se při přijetí kontroluje *hash* kód. Pokud proběhly kontroly přijetí paketu s animací úspěšně, přejde se k přijetí obrazu. Zde se opět kontroluje obsah paketu pomocí *hash* kódu. Dále se kontroluje číslo paketu. Pokud proběhla kontrola čísla paketu i kontrola *hash* kódu úspěšně, jsou data o obraze a animaci uložena do globální proměnné *Data*, přes kterou jsou následně odeslána na mikrokontrolér ATmega.

9 KOMUNIKACE MEZI RASPBERRY A ATMEGA

Před odesíláním dat je nejprve třeba vytvořit komunikaci mezi mikrokontrolérem a Raspberry. K tomuto účelu je na Raspberry Pi vytvořena funkce *MCUConnect*. Mezi vstupní parametry této funkce patří identifikace zařízení a přenosová rychlost. Jedná se o asynchronní komunikaci, na obou zařízeních je tedy potřeba nastavit rychlost přenosu. Vzhledem ke zrychlení přenosu dat zvolím pro přenos mezi Raspberry Pi a ATmega větší rychlost, než byla použita v kapitole 4.1.2 při oživování této komunikace. Nejvyšší možnou rychlostí přenosu, vzhledem k rychlosti hodinového taktu mikrokontroléru ATmega, je 38 400 *Bd*. Ve funkci *MCUConnect* se nejprve naváže spojení s ATmega a poté je spuštěna knihovna *WiringPi*, která slouží k odesílání dat.

K oživení komunikace mezi mikrokontrolérem a Raspberry bylo v dřívější kapitole (4.1.2) ze strany mikrokontroléru pro příjem dat použito přerušení. Nyní jsem však zvolil jiný způsob. Místo příjmu v přerušení se bude na mikrokontroléru čekat na změnu hodnoty pinu, který indikuje příchozí data.

9.1 Odesílání dat z Raspberry PI

Pokud došlo k úspěšnému přijetí dat na Raspberry Pi, sedí tedy čísla paketů a kontrola *hash* kódu, přejde se k odeslání dat na ATmega. Před samotným odesíláním dat je nejprve třeba navázat komunikaci s mikrokontrolérem. K tomu slouží výše zmíněna funkce *MCUConnect*. Návratovou hodnotou funkce je identifikátor daného zařízení (mikrokontroléru ATmega128A), který je použit k odesílání dat. Funkce *MCUConnect* je volána při startu programu na Raspberry Pi.

```
int MCUConnect(const char *iDevice, const int iBaudRate)
        {
        int fd;
        if ((fd = serialOpen (iDevice, iBaudRate)) < 0)
                return -1;
        if (wiringPiSetup () == -1)
                return -1;
        return fd;
        }
```

K odesílání dat na mikrokontrolér byly vytvořeny funkce *SendAnimation* a *SendData*. Vytvoření dvou rozdílných funkcí bylo potřeba vzhledem k různé délce odesílaných paketů a přehlednosti kódu. V první funkci je odeslán paket s informací o animaci a v druhé funkci se opakovaně posílají pakety se sloupci

obrazu, dokud nedojde k odeslání všech 140 sloupců. Obě funkce postupně procházejí zásobníky dat a tato data odesílají.

```
void SendAnimation(int fd, char iBuffer[5])
        {
        for (int i = 0; i < 5; i++)
                serialPutchar(fd,iBuffer[i]);
        }

void SendData(int fd, char iData[140][3])
        {
        for (int i = 0; i < 140; i++)
                for (int j = 0; j < 3; j++)
                        serialPutchar(fd,iData[i][j]);
        }
```

9.2 Příjem dat na ATmega

K přijímání dat přes UART na mikrokontréleru ATmega slouží detekce změny bitu *RXC0* v registru *UCSR0A*. Tento bit je nastaven do logické 1, dokud se v zásobníku nacházejí nepřečtená data. Při přijetí dat přes UART se nejprve čeká, dokud je daný zásobník prázdný. Jakmile je detekováno naplnění zásobníku, jeho obsah se načte do proměnné. Jak bylo zmíněno výše (9.1), nejprve se odešle paket s informací o animaci. V hlavní smyčce programu je zavolána funkce *ReceiveAnimation*. V této funkci se čeká na zahájení komunikace, tedy na příchod prvních dat. Jakmile jsou data přijata, jsou postupně načtena do proměnné *iBuffer*. [35]

```
void ReceiveAnimation (char iBuffer[4])
      {
      for (int j = 0; j < 4; j++)
                {
                while (!(UCSR0A & (1<<RXC0)));
                        iBuffer[j] = UDR0;
                }
      }
```

Po načtení informací o animaci je z hlavní smyčky zavolána funkce *ReceiveData*. Tato funkce funguje na stejném principu jako výše zmíněna funkce *ReceiveAnimation*. Čeká se na příjem dat, tedy změnu *RXC0* bitu, a poté je obsah zásobníku načten do proměnné pole s názvem *iData* na příslušnou pozici.

```
void ReceiveData(char iData[140][3])
	{
	for (int i = 0; i < 140; i++)
		for (int j = 0; j < 3; j++)
			{
			while (!(UCSR0A & (1<<RXC0)));
			iData[i][j] = UDR0;
			}
	}
```

Přijatá data jsou typu *char*, jedná se o osmi bitové hodnoty. Hodnota pixelu na daném řádku a sloupci je velikosti jednoho bitu. Displej obsahuje 19 řádků. K uložení hodnoty o jednom sloupci je tedy třeba tří hodnot typu *char*. Celkem 19 hodnot jsem rozdělil do tří proměnných po 7, 7 a 5 bitech. Před odesláním obrazu byla informace o sloupci převedena z 19 hodnot do 3 hodnot, nyní je třeba informaci převést zpět. K převodu na 19 hodnot slouží bitové operace ve funkci *ConvertData*. Převod z jedné proměnné do sedmi proměnných je založen na principu bitového posunu osmibitové proměnné směrem doprava a následná kontrola, zda je hodnota lichá. Nachází-li se na nulté pozici logická hodnota 1, je osmibitová proměnná lichá. V opačném případě je proměnná sudá. Je-li hodnota lichá, tedy zbytek po dělení dvěma je 1, zapíše se na příslušnou pozici 1. V opačném případě se zapíše na danou pozici 0. Funkce *ConvertData* tedy postupně prochází sloupce a z pole o velikosti 140x3 vytváří nové pole o velikosti 140x19, které již odpovídá rozlišení displeje.

```
void ConvertData (char Input[140][3], char Output[140][19])
	{
	for (int i = 0; i < 140; i++)
		for (int j = 0; j < 19; j++)
			if ((Input[i][(int)(j/7)] >> (j - ((int)(j/7))*7)) % 2 == 1)
				Output[i][j] = '1';
			else
				Output[i][j] = '0';
	}
```

Po převedení proměnné pole do příslušného formátu je zavolána funkce *WriteData*, která vykresluje data na displej. Ve funkci se vytváří i příslušné animace, které přišly v prvních paketech.

10 VYKRESLENÍ NA DISPLEJ

K vykreslení přijatých dat na displej slouží funkce *WriteData*. Vstupními parametry této funkce jsou informace o animaci a obraz, který se má vykreslit. Informace o animaci se skládají ze směru animace, velikosti posunu, intervalu a počtu opakování posunu.

```
void WriteData(char iData[140][19], char iDirection,
                    char iStep, char iInterval, char iRepetition);
```

Při zavolání funkce se přejde do smyčky *do-while*. Účel této smyčky je opětovné vykreslování dat na displej, tedy tvorba animace. Při každé iteraci smyčky je inkrementována proměnná *Repetition*. Při dosažení požadovaného počtu opakování je smyčka ukončena, je proveden návrat do hlavní smyčky programu *main* a opětovně se čeká na příjem dalších dat. Výhodou smyčky *do-while* je fakt, že bude provedena vždy alespoň jednou. Není tedy třeba vytvářet funkci zvlášť pro statický obraz a zvlášť pro animaci. K ukončení smyčky *do-while* dojde i za stavu, kdy je směr animace nastaven do hodnoty *None*.

Vykreslení probíhá po pixelech. K vykreslení jednoho pixelu slouží funkce *ChangePixel*. Vstupem této funkce je hodnota a souřadnice pixelu. Ve funkci se poté nastaví příslušné hodnoty registrů výběrem z globálních proměnný *Column* a *Row*. Po nastavení registrů je přiveden impuls zapsáním logické hodnoty jedna na 6. pozici v registru *PORTC*.

```
void ChangePixel(char iPixel, int iColumn, int iRow)
        {
        if (iPixel == '1')
                PORTC = 0b00000000;
        else
                PORTC = 0b10000000;
        PORTB = Column[iColumn];
        PORTA = Row[iRow];
        PORTC = PORTC | 0b01000000;
        _delay_us(500);
        PORTC = !(PORTC & 0b01000000);
        }
```

Při vykreslování statického obrazu se pouze postupně projde pole hodnot a na každou pozici je volána funkce *ChangePixel*. Při vykreslení animace je však

funkce volána vždy na posunutou pozici. Posunutí záleží na směru animace. Velikost posunutí závisí na vstupní proměnní *iStep*. Při vykreslení animace je tedy nejprve vykreslen statický obraz. Při další iteraci smyčky *do-while* probíhá vykreslení pixelů na pozici posunuté o hodnotu proměnné *Step*. Tato proměnná je každou iterací zvětšená/zmenšená o vstupní hodnotu *iStep*. Tímto způsobem by ihned došlo k přetečení indexů pole mimo rozsah. Je tedy nutno kontrolovat hodnotu *Step* a příslušně ji normalizovat do definovaného rozsahu. Následující kontrola slouží pro horizontální animace.

```
if (i + Step < 0)
        ChangePixel(iData[i + Step + (-Step/139 + 1)*140][j], i, j);
```

K lepší čitelnosti zobrazené animace je použito dvojího typu vykreslování. Je-li posun v horizontálním směru, je vhodnější, aby se obraz vykresloval po sloupcích. Pro vertikální posun je naopak vhodné vykreslovat informace po řádcích. Rozlišení těchto stavů probíhá pomocí porovnávání vstupní hodnoty *iDirection*, ve které je uložen směr animace. Z vykresleného obrazu, viz Obr. 10.1 je patrné, že spodní řádek displeje je nefunkční.

Obr. 10.1 Vykreslený text na displeji

Rozhodl jsem se tedy změřit, zda se chyba nachází v logických obvodech pro volbu řádku nebo přímo v panelu. Pomocí osciloskopu jsem měřil enable impulsy. Zjistil jsem, že při volbě spodního řádku je z dekodéru v první vrstvě korektně generován impuls. Tento impuls vede do dekodéru druhé vrstvy, na jehož výstupu na odpovídajícím řádku také naměřím impuls. Za druhým dekodérem následuje tranzistorové pole. Konkrétně toto tranzistorové pole bylo vyměněno při oživování displeje, mělo by být tedy funkční. Jelikož jsem naměřil impuls i za tímto polem, chyba s vykreslováním spodního řádku bude pravděpodobně přímo na panelu.

ZÁVĚR

Z literární rešerše vyplývá hlavní výhoda terčíkových displejů. Je to možnost mít stále zobrazenou informaci i po odpojení napájení. Touto vlastností disponují také například elektroforetické displeje. Ostatní zmíněné displeje potřebují k zobrazení informace stálý odběr energie. Terčíkové displeje nejsou čitelné za sníženého osvětlení, čímž se liší od LED, OLED a LCD displejů.

Při oživení panelu musela být nejprve vyměněna sada elektronických součástek, mezi které patřily především tranzistorová pole. Díky kterým docházelo ke zkratům a k ničení snižujících regulátorů. Regulátor typu 78S24 musel být také v důsledku zkratu osazen nový. Jako největší problém při oživování panelu se ukázal ne příliš dobrý návrh desek plošných spojů a výběr elektronických komponent, což však nebylo součástí mé práce na displeji. Ve schématech, která jsem měl k dispozici byly chaoticky popsány jednotlivé spoje, především tedy u jednoho z konektorů, který musel být propojen přes drátové propoje, neboť rozložení konektoru na jedné desce neodpovídalo zapojení konektoru na desce druhé. Při prvotním vykreslování na displej jsem si tak stále myslel, že je chyba buď ve vytvořeném programu nebo v osazených součástkách. Ač mělo oživení displeje zabrat nepatrnou část, oživením displeje se zabývala značná část semestrálního projektu.

Poslední část zadání byl návrh a realizace řídicího systému. Komunikace probíhá přes internetovou síť pomocí síťových socketů. Systém je řízen z operačního systému Windows, na kterém je vytvořena aplikace pro návrh a odesílání obrazu a animací. Odeslání navrženého obrazu na Rapsberry Pi je umožněno po zadání IP adresy tohoto zařízení.

Značnou část okna aplikace zabírá vykreslená bitmapa, přes kterou systém umožňuje návrh obrazovek změnou jednotlivých pixelů nebo zadáváním znaků z klávesnice. Dále je implementováno jednak uložení, ale také načtení celé obrazovky ze souboru. Volba animace probíhá přes posuvné prvky, které nastavují velikost posunu animace, interval mezi posuny a počet opakování posunů. Dalším prvkem je výběr ze seznamu. Ze seznamu se vybere směr animace. Po vykreslení obrazu je stále nefunkční spodní řádek panelu. Pomocí osciloskopu jsem však zjistil, že problém se nenachází na deskách s logickým výběrem řádku. Proto bych vyřešení tohoto problému zařadil mezi další možný vývoj této práce.

LITERATURA

[1] LATTENBERG, Ivo. *Moderní zobrazovací jednotky*. AUTOMA [online]. 2016 [cit. 2018-12-29]. Dostupné z: http://automa.cz/cz/casopis-clanky/moderni-zobrazovaci-jednotky-2000_03_27628_582/

[2] DUDÁČEK, Karel. Alfanumerické displeje. In: *Domovské stránky uživatelů* [online]. [cit. 2019-04-10]. Dostupné z: http://home.zcu.cz/~dudacek/PZ/alfanum_displeje.pdf

[3] DIVIŠ, Jozef. Základy elektroniky: LED dioda. In: *Elektronika: Základy elektroniky* [online]. [cit. 2019-04-10]. Dostupné z: http://old.spsemoh.cz/vyuka/zel/diody.htm#ledlasd

[4] DIVIŠ, Jozef. Zobrazovací jednotky: LED technologie. In: *Elektronika: Základy elektroniky* [online]. [cit. 2019-04-10]. Dostupné z: http://old.spsemoh.cz/vyuka/zel/zobrazovaci-jednotky.htm#LED

[5] DIVIŠ, Jozef. Zobrazovací jednotky: Princip LCD displeje. In: *Elektronika: Základy elektroniky* [online]. [cit. 2019-04-10]. Dostupné z: http://old.spsemoh.cz/vyuka/zel/zobrazovaci-jednotky.htm#LCDprincip

[6] OLED technology: introduction and basics. In: *OLED-info* [online]. Dec 23, 2018 [cit. 2019-04-24]. Dostupné z: https://www.oled-info.com/oled-technology

[7] BONHEUR, Kristoffer. Electrophoretic display: Advantages and disadvantages. In: *Version daily* [online]. 26 December 2017 [cit. 2019-04-10]. Dostupné z: http://www.versiondaily.com/advantages-disadvantages-electrophoretic-display/

[8] How it works. In: *Flip dots* [online]. [cit. 2019-04-10]. Dostupné z: https://flipdots.com/en/electromagnetic-flip-disc-technology-how-it-works/

[9] Zobrazovací prvky a moduly: Zobrazovací moduly "flip DOT" a "flip DOT-LED". In: *Buse* [online]. [cit. 2019-04-11]. Dostupné z: http://www.buse.cz/cs/zobrazovaci-moduly-flip-dot-a-flip-dot-led

[10] DOT-LED display of a bus Irisbus Citybus 18M [online]. 2014 [cit. 2018-12-29]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Irisbus%2C_informa%C4%8Dn%C3%AD_panel.jpg/800px-Irisbus%2C_informa%C4%8Dn%C3%AD_panel.jpg?1543004853305

[11] *Raspberry Pi 1 model B* [online]. [cit. 2018-12-29]. Dostupné z: https://www.raspberrypi.org/app/uploads/2012/09/sony-rasp-pi.jpg

[12] RPi Hardware: Specifications. *Embedded Linux Wiki* [online]. 10 February 2019 [cit. 2019-04-12]. Dostupné z: https://elinux.org/RPi_Hardware

[13] GPIO. *Raspberry Pi* [online]. [cit. 2019-04-12]. Dostupné z: https://www.raspberrypi.org/documentation/usage/gpio/
[14] GIRLING, Gray. *Raspberry PI: A practical guide to the revolutionary small computer*. Sparkford: Haynes Publishing, 2013. ISBN 978-0-85733-295-0.
[15] *ATMEL. 8-bit AVR Microcontroller ATmega128A: DATASHEET COMPLETE*. San Jose, 2015. Dostupné také z: https://www.microchip.com/
[16] PHILIPS SEMICONDUCTORS. *74HC/HCT238: 3-to-8 line decoder/demultiplexer* [online]. 1990 [cit. 2019-04-12]. Dostupné z: https://www.gme.cz/data/attachments/dsh.425-059.1.pdf
[17] TEXAS INSTRUMENTS. *LM2596 SIMPLE SWITCHER: Step-Down Voltage Regulator* [online]. [cit. 2019-04-12]. Dostupné z: http://www.ti.com/lit/ds/symlink/lm2596.pdf
[18] *Wiring Pi: GPIO Interface library for the Raspberry Pi* [online]. 2013 [cit. 2018-12-29]. Dostupné z: http://wiringpi.com/
[19] Basics of UART comunnication. *Circuit Basics* [online]. [cit. 2019-05-05]. Dostupné z: http://www.circuitbasics.com/basics-uart-communication/
[20] VÁŇA, Vladimír. *Mikrokontroléry ATMEL AVR - Programování v jazyce C: Popis a práce ve vývojovém prostředí CodeVisionAVR C*. Praha: BEN, 2003. ISBN 80-7300-102-0.
[21] *L78S: 2 A positive voltage regulator IC*. 2014. Dostupné také z: https://www.gme.cz/data/attachments/dsh.330-021.2.pdf
[22] WinMain function. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-winmain
[23] Windows Data Types: HINSTANCE. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/winprog/windows-data-types
[24] GetMessage function: Remarks. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-getmessage
[25] TranslateMessage function: Remarks. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-translatemessage
[26] CreateWindow. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/ms959988(v%3Dmsdn.10)
[27] Device Contexts: About Device Context. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/cs-cz/windows/desktop/gdi/about-device-contexts

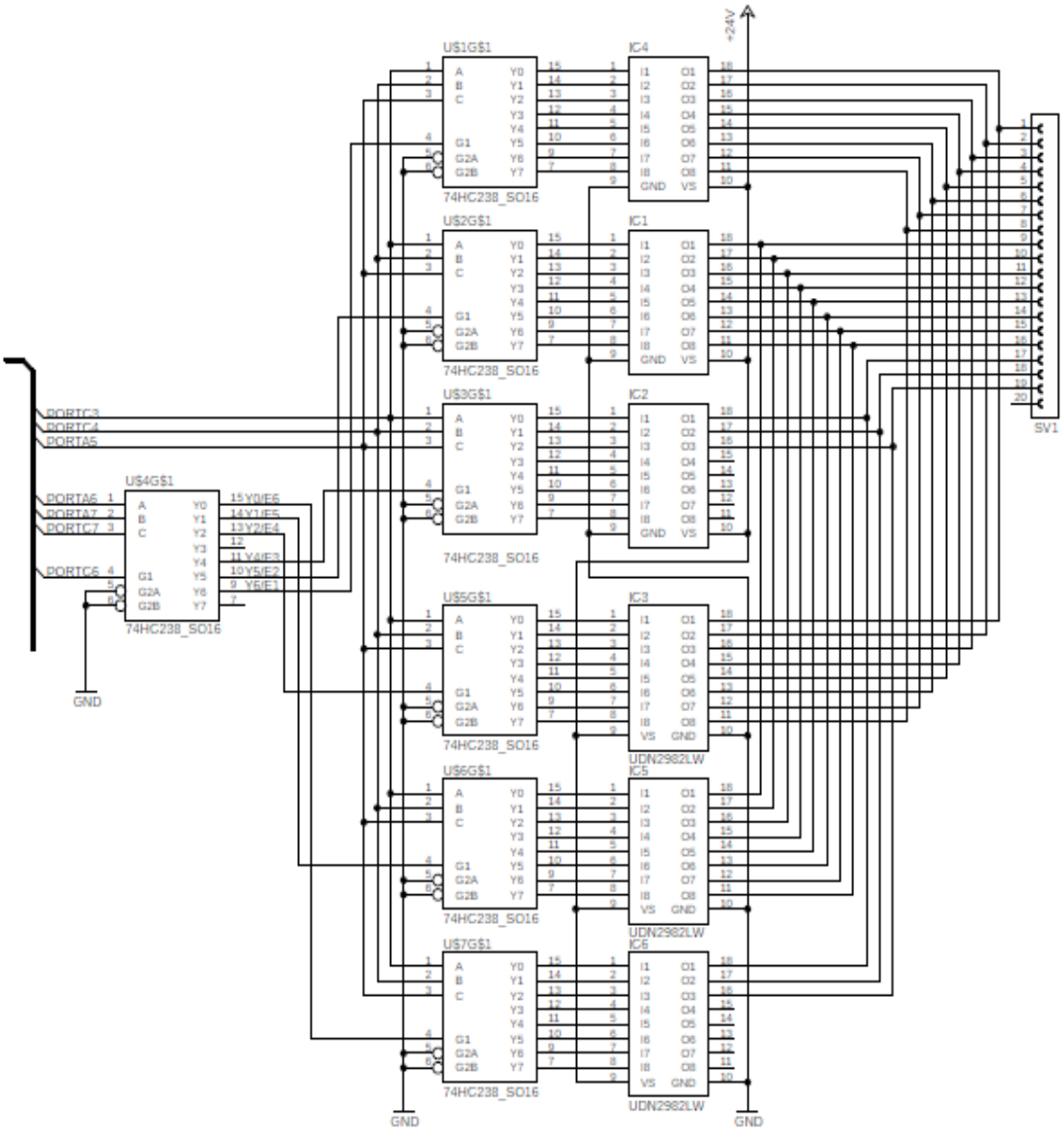
[28] CreateBitmap. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/api/wingdi/nf-wingdi-createbitmap
[29] StretchBlt function. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/api/wingdi/nf-wingdi-stretchblt
[30] WM_LBUTTONDOWN message. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/inputdev/wm-lbuttondown
[31] Virtual-Key Codes. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/inputdev/virtual-key-codes
[32] OPENFILENAMEA structure. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/api/commdlg/ns-commdlg-tagofna
[33] CreateFileA function: CREATE_ALWAYS. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-createfilea
[34] Initializing Winsock. *Microsoft Docs* [online]. [cit. 2019-04-13]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/winsock/initializing-winsock
[35] UART Programing with Atmega128. *Explore Embedded* [online]. [cit. 2018-12-29]. Dostupné z: https://exploreembedded.com/wiki/UART_Programming_with_Atmega128

Seznam symbolů, veličin a zkratek

A	-	Ampér
ARM	-	Advanced RISC Machine
Bd	-	Baud
DC	-	Device Context
DDR	-	Data Direction Register
EEPROM	-	Electrically Erasable Programmable Read-Only Memory
GND	-	Ground
I2C	-	Inter-Integrated Circuit
IP	-	Internet Protocol
JTAG	-	Joint Test Action Group
LCD	-	Liquid Crystal Display
LED	-	Light-Emitting Diode
MCU	-	Micro Controller Unit
MHz	-	Megahertz
ms	-	Milisekunda
OLED	-	Organic Light Emitting Diode
PC	-	Personal Computer
RISC	-	Reduced Instruction Set Computers
SD	-	Secure Digital
SPI	-	Serial Peripheral Interface
SRAM	-	Static Random Access Memory
SSH	-	Secure Shell
TCP	-	Transmission Control Protocol
TFT	-	Thin Film Transistor
UART	-	Universal Asynchronous Receiver and Transmitter
USB	-	Universal Serial Bus
V	-	Volt
WSA	-	Windows Sockets API

Seznam příloh

1 Schéma korektního zapojení dekodérů



2 CD

1. **Text práce**
2. **Program na mikrokontrolér**
3. **Program na PC**
4. **Program na Raspberry Pi**
5. **Schéma**