STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor SOČ: 10. Elektrotechnika, elektronika a telekomunikace

Mikroprocesorová vykreslovací jednotka

(Rendering microprocessor unit)

Autoři: Tomáš Pazdiora

Škola: VOŠ a SPŠE Olomouc

Kraj: Olomoucký kraj

Konzultant: Mgr. Roman Babička

Olomouc 2015

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v seznamu vloženém v práci SOČ.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Olomouci dne …… podpis: ……

Poděkování

Chtěl bych vyslovit poděkování panu Mgr. Romanovi Babičkovi za odborné konzultace a poskytnuté informace.

ANOTACE

Tato práce vznikla za účelem získat nové zkušenosti a poskytnout ostatním vývojářům informace z vývoje a poskytnout jim části práce pro volné použití. Celá tato práce řeší vykreslování a zobrazování obrazu na monitoru pomocí mikroprocesoru. Tato problematika je velmi rozsáhlá a zasahuje jak do hardwarové, tak do softwarové oblasti. Je tedy na pomezí mezí elektronikou a informatikou.

Klíčová slova: mikroprocesory, AVR ,SD, VGA, RAM, double buffering, dithering, D/A, DAC, R-2R, procedurální generování, BMP, UART, SPI, CRC, ASM, C

ANNOTATION

This project was created for the purpose of getting a new experience and information and to provide the information from development and the parts of this project for free usage for any purpose to the other developers. This entire project is about rendering and showing images on the monitor. It's widespread and it covers both hardware and software areas. So it's on the border between electronics and information technology.

Keywords: microprocessors, AVR ,SD, VGA, RAM, double buffering, dithering, D/A, DAC, R-2R, procedural generation, BMP, UART, SPI, CRC, ASM, C

OBSAH

Úvod

Zadáním práce bylo navrhnout zařízení, které je schopné zobrazovat obraz na monitoru. Při návrhu jsem se zaměřil převážně na to, aby byly komponenty pro výrobu dostupné a celková cena zařízení byla přijatelná.

Mým cílem nebyla snaha vytvořit zařízení pro koncové uživatele, ale zařízení, které by sloužilo jako studijní materiál a jeho části by mohly být kýmkoliv libovolně využívány pro další projekty (viz Dodatek). Vzhledem k tomu, že se tato práce zabývá širokým spektrem různých problematik, myslím, že může být dobrým zdrojem informací téměř pro každého, který se pohybuje na poli elektroniky a mikroprocesorů.

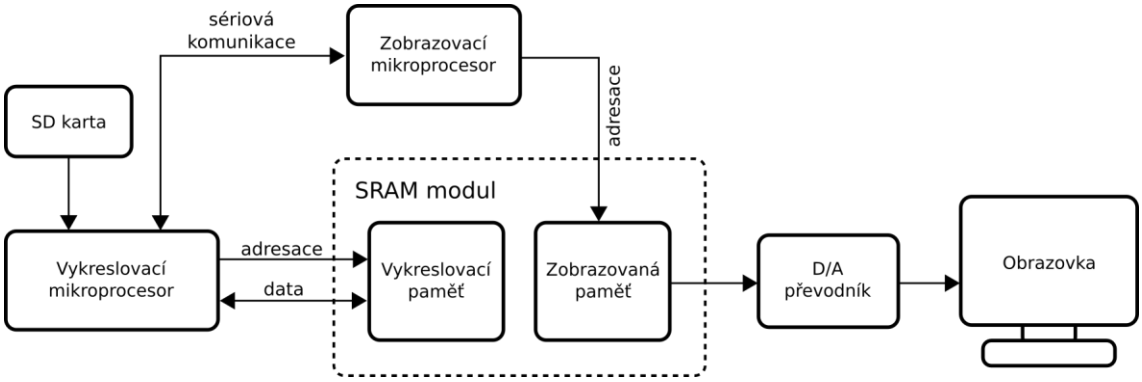
Kapitoly jsou řazeny tak, aby odpovídaly postupnému řešení dílčích částí při vývoji. V první části je rozebrán hardware zařízení a v druhé je popsán jeho software.

1. STRUČNÝ POPIS ZAŘÍZENÍ

Mikroprocesorová vykreslovací jednotka je zařízení určené k zobrazování obrazu na monitoru. Tento obraz je buďto načtený z externí paměti, nebo vygenerovaný vykreslovacím mikroprocesorem.

Vykreslovací jednotka není brána jako finální produkt, ale je určena k modifikaci a zkoušení jejích možností.

Obsahuje dva mikroprocesory, mezi které jsou rozděleny potřebné úkony, SD kartu, která poskytuje vstupní data, modul s paměťmi, do kterých se ukládá obraz a převodník, který převádí digitální výstup na analogový signál.



Obr. 1 – 1: Zjednodušené principiální blokové schéma celého zařízení

2. POUŽITÉ MIKROPROCESORY

Pro tuto práci jsem chtěl použít mikroprocesory, které jsou v ČR snadno dostupné a za rozumnou cenu.

Bylo možné využít mikroprocesory od firem Microchip, NXP a Atmel. Od těchto tří výrobců můžeme získat mikroprocesory s různými architekturami. Od všech tří výrobců je v ČR dostupná klasická architektura 8051. Firmy Microchip a Atmel v ČR dále nabízí mikroprocesory s 8bitovou architekturou RISC. Poslední dostupnou architekturou v ČR je architektura ARM (konkrétně Cortex-M0) od výrobce NXP.

Tito výrobci dále nabízí 32bitové mikroprocesory s architekturou RISC, nebo ARM architekturu vyšších tříd. Tyto produkty ale dnes téměř není možné v kamenných obchodech ČR sehnat.

Nakonec jsem se rozhodl pro 8bitové mikroprocesory architektury RISC od firmy Atmel, jelikož s nimi mám již zkušenosti.

Při rozhodování byla velkým soupeřem architektura ARM. Ta je pro tuto práci přímo ideální. Mikroprocesory této architektury mají větší velikosti pamětí a mohou běžet na vyšších frekvencích. Vyšší řady architektury ARM se dnes používají v různých mobilních zařízeních a jsou velmi populární. Pro tuto architekturu jsem se ale nerozhodl z důvodu nízké dostupnosti v ČR.

2.1. MIKROPROCESORY AVR

AVR je označení upravené Harvardské RISC architektury mikroprocesorů firmy Atmel. Označení AVR zahrnuje jak 32bitové, tak 8bitové mikroprocesory. 8bitové se dále dělí do tří tříd.

První třídou jsou mikroprocesory AVR XMEGA, z celé řady těchto 8bitových mikroprocesorů jsou nejvýkonnější. Dosahují rychlostí až 32 MHz. Nejsou ovšem dostupné v ČR.

Další třídou jsou mikroprocesory megaAVR, spolu s architekturou tinyAVR jsou jedny z nejdostupnějších mikroprocesorů v ČR. Jejich výhodou může být velké množství pinů (28 – 100), další výhodou je hardwarová implementace hojného počtu periferních obvodů jako komunikační protokoly SPI, TWI (I^2C), UART/USART, A/D převodníky, analogové komparátory, čítače, časovače a další. Dosahují frekvencí až 20 MHz.

Poslední třídou jsou již výše zmíněné mikroprocesory tinyAVR. Tyto mikroprocesory jsou navrhovány s důrazem na jejich cenu a velikost. Mají menší velikosti pamětí než mikroprocesory megaAVR , mají integrovány jen některé periferní

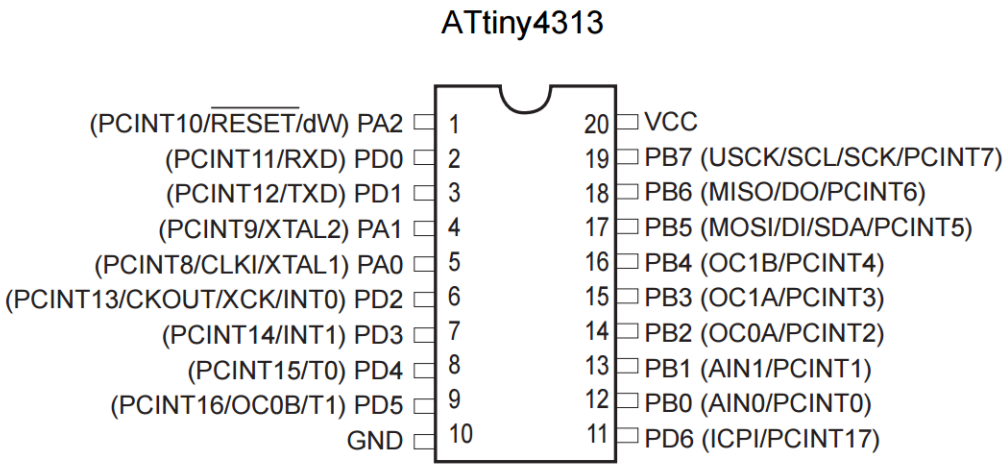
obvody a mívají menší počet pinů (6 – 32). S rychlostí jsou na tom stejně tak dobře jako třída megaAVR. Dosahují frekvencí až 20 MHz.

Tyto tři třídy mikroprocesorů mají maximální výkon 1.0 MIPS/MHz (MIPS – milion instrukcí za sekundu). Tento výkon platí pro většinu základních instrukcí. Některé instrukce ale trvají více strojových cyklů.

2.2. Zobrazovací mikroprocesor

Jako zobrazovací mikroprocesor byl zvolen mikroprocesor **ATtiny4313** řady tinyAVR. Je nástupcem mikroprocesoru ATtiny2313. Tyto dva mikroprocesory jsou kompatibilní. Rozdílem je velikost pamětí, ta je u ATtiny4313 dvojnásobná (flash, SRAM i EEPROM). Mezi jeho hlavní parametry potřebné v této práci patří:

- frekvence až 20 Mhz při napětí 4,5 – 5,5 V,
- 4 Kb programové paměti flash,
- 256 bytů statické RAM,
- hardwarová podpora sériové komunikace pomocí UART/USART,
- jeden 8bitový a jeden 16bitový časovač/čítač,
- 18 programovatelných vstupně výstupních pinů.



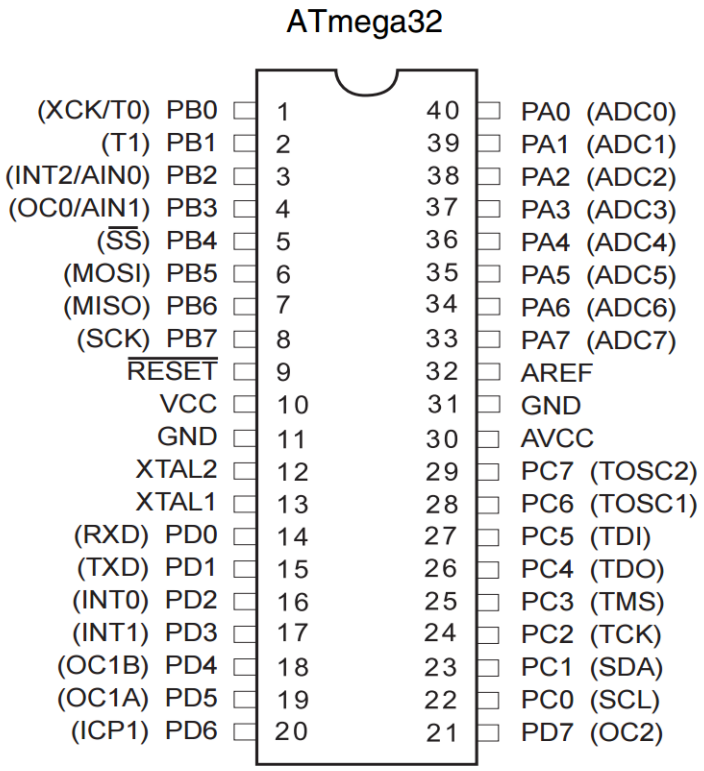
Obr. 2 – 1: ATtiny4313 – rozložení pinů

2.3. Vykreslovací mikroprocesor

Vykreslovacím mikroprocesorem je mikroprocesor **ATmega32** řady megaAVR. Byl zvolen kvůli ulehčení a urychlení návrhu hardwaru a softwaru, vzhledem k jeho poměrně velké paměti a velkému počtu vstupně výstupních pinů. Mohl být nahrazen jiným, levnějším mikroprocesorem, jako například ATmega8. To by ovšem mohlo

zkomplikovat a prodloužit vývoj zařízení. Mezi hlavní parametry mikroprocesoru ATmega32 potřebné v této práci patří:

- frekvence až 16 Mhz při napětí 4,5 – 5,5 V,
- 32 Kb programové paměti flash,
- 2 Kb statické RAM,
- hardwarová podpora protokolů UART/USART a SPI,
- 32 programovatelných vstupně výstupních pinů.



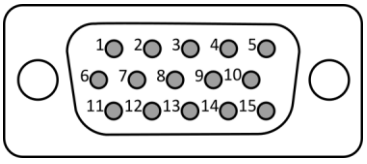
Obr. 2 – 2: ATmega32 – rozložení pinů

3. ROZHRANÍ VGA

VGA (Video Graphics Array) standard byl vydán v roce 1987 společností IBM. Jde o standard popisující velikost video paměti, hloubku barev, obnovovací frekvenci, rozlišení a další parametry. VGA standard definuje jak stranu generující analogový signál, tak stranu přijímající analogový signál. Vzhledem k tomu, že jsou tyto dvě strany odděleny právě analogovým signálem, není nutné dodržet tento standard kompletně. Pro zobrazení obrazu na přijímací straně stačí dodržet pouze časování, velikost napětí analogového signálu a impedanci.

VGA obsahuje signály pro barvu, synchronizaci a identifikaci monitoru. Barva se skládá ze tří signálů: červené, zelené a modré barvy. Každý signál barvy má impedanci 75 Ω a maximální rozkmit napětí 0,7 V. Všechny ostatní signály mají napětí na TTL úrovni (5 V). Synchronizační signály se dělí na horizontální a vertikální. Jejich polarita je dána použitým režimem zobrazení. V této práci je použit režim 640x480 pixelů při 60 snímcích za sekundu. V tomto režimu mají oba synchronizační signály negativní polaritu (tzn. 0 V – synchronizační puls, jinak 5 V). Identifikační signály jsou dle VGA standardu signály ID0 – ID3, které jsou dnes již nahrazeny signály pro sériovou komunikaci pomocí protokolu I^2C (standard VESA E-DDC).

Běžným konektorem je DE-15, což je třířadý, 15pinový D-Sub, také známý jako Cannon, podle firmy která jej zavedla (ITT-Cannon).



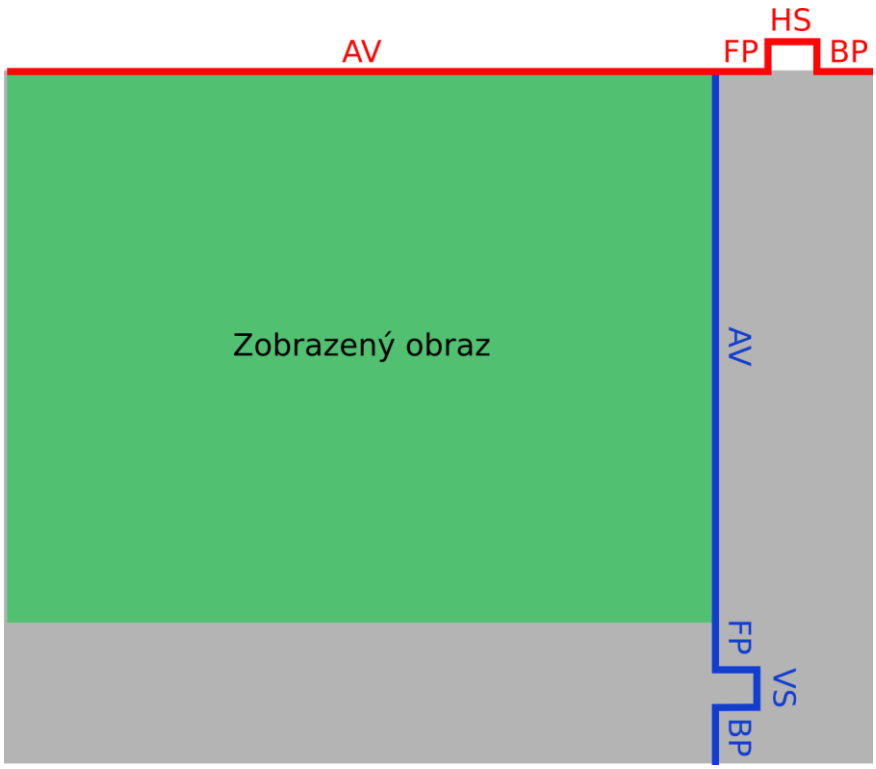
Obr. 3 – 1: VGA konektor

Pin	Popis
1	**Červená**
2	**Zelená**
3	**Modrá**
4	ID2 / Rezervován (E-DCC)
5	Zem
6	Červená – zem
7	Zelená – zem
8	Modrá – zem
9	Bez pinu / +5V z PC (E-DCC)
10	Synchronizace – zem
11	ID0 / Rezervován (E-DCC)
12	ID1 / Sériová data (E-DCC)
13	**Horizontální synchronizace**
14	**Vertikální synchronizace**
15	ID3 / Hodinový signál (E-DCC)

Tab. 3 – 1: Rozložení pinů VGA konektoru

3.1. Časování

Průběhy signálů jsou rozděleny do několika částí. Tyto části nám říkají, kdy se má zobrazovat video a kdy se mají sepínat synchronizační signály.



Obr. 3 – 2: VGA synchronizační signály

Zkratka	Název	Popis
AV	Active video	Část, ve které je na RGB kanály přiveden signál. V ostatních částech musí být na všech kanálech nulová intenzita.
FP	Front porch	Časový interval před synchronizačním pulsem
HS/VS	Sync pulse	Synchronizační puls. Vertikální i horizontální pulsy jsou sepínány nezávisle na sobě. (tzn. horizontální puls je na každé lince, i v průběhu vertikálního pulsu)
BP	Back porch	Časový interval po synchronizačním pulsu

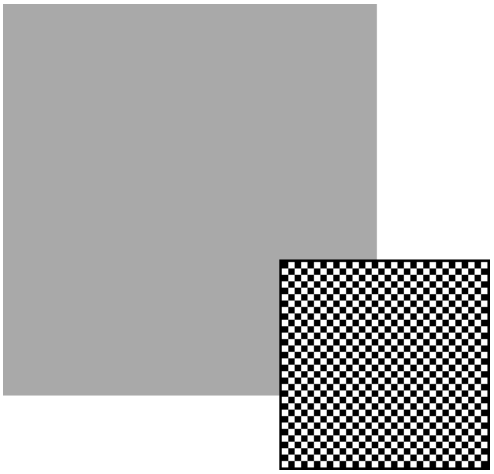
Tab. 3 – 2: Vysvětlení zkratek částí synchronizačních signálů

VGA signál může mít několik různých režimů. Každý režim se liší rozlišením obrazu, obnovovací frekvencí (počet snímků za sekundu) a jsou pro ně definovány důležité parametry: frekvence pixelu, horizontální frekvence (frekvence řádku), polarity synchronizačních signálů a doby trvání jejich částí.

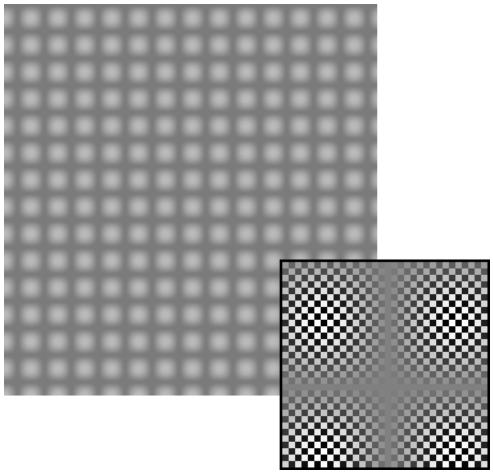
V této práci je použit režim 640x480 při 60Hz. Zde nastává problém, jelikož tento režim definuje frekvenci pixelu na 25.175 MHz a maximální frekvence použitého mikroprocesoru generujícího synchronizační signály je 20 MHz. Z toho vyplývá, že nebude možné generovat správnou velikost pixelu, pixely budou roztáhlé a nastane zde jev zvaný overscan. Neznamená to ještě ovšem, že nebude možné obraz správně synchronizovat. Časy všech částí synchronizačních impulsů se dají s rozumnou odchylkou přepočítat na frekvenci pixelu 20MHz. Většina zobrazovacích zařízení by si měla s touto odchylkou poradit. Novější zařízení jsou ale na přesnost signálu náročnější.

3.1.1. OVERSCAN

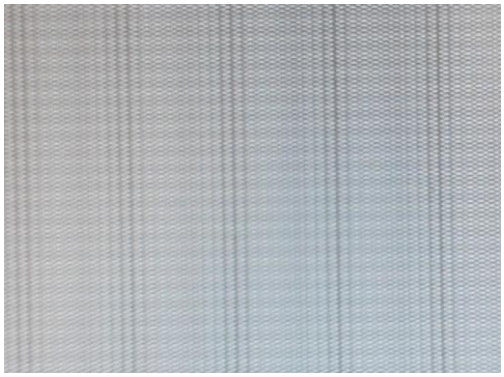
Jev zvaný overscan nastává při situaci, když se mapuje obraz s určitým rozlišením na odlišné rozlišení monitoru a podíl těchto rozlišení není celočíselný. Na monitoru je poté značně viditelná mřížka. V této práci je overscan pouze horizontální, a proto jsou vidět jen vertikální pruhy (viz Obr. 3 – 5).



Obr. 3 – 3: Mapování pixelu 1:1



Obr. 3 – 4: Overscan 10%



Obr. 3 – 5: Overscan vykreslovací jednotky

3.1.2. Přepočet časování

Pro správnou synchronizaci je důležité dodržet hodnoty časování. Všechny hodnoty vycházejí z času trvání jednoho pixelu. Při snížení frekvence pixelu se snažíme najít takový počet pixelů v určitých částech synchronizace, aby se doby trvání těchto částí co nejméně lišily od originálních hodnot. V následujících tabulkách jsou originální hodnoty, přepočtené hodnoty a jejich odchylky.

Hodnoty časování pro režim 640x480 při 60Hz		
Vertikální frekvence	59,94047619 Hz	
Horizontální frekvence	31,46875 kHz	
Frekvence pixelu	25,175 MHz	
Linková část	**Pixely**	**Čas [µs]**
Active video	640	25,42204568
Front porch	16	0,635551142
Sync pulse	96	3,813306852
Back porch	48	1,906653426
Celá linka	800	31,7775571
Snímková část	**Linky**	**Čas [ms]**
Active video	480	15,25322741
Front porch	10	0,317775571
Sync pulse	2	0,063555114
Back porch	33	1,048659384
Celý snímek	525	16,68321748

Tab. 3 – 3: Hodnoty časování pro režim 640x480 při 60Hz

Hodnoty časování pro nižší frekvenci pixelu		
Vertikální frekvence	59,99250094 Hz	
Horizontální frekvence	31,49606299 kHz	
Frekvence pixelu	20 MHz	
Linková část	**Pixely**	**Čas [µs]**
Active video	508	25,4
Front porch	13	0,65
Sync pulse	76	3,8
Back porch	38	1,9
Celá linka	635	31,75
Snímková část	**Linky**	**Čas [ms]**
Active video	480	15,24
Front porch	10	0,3175
Sync pulse	2	0,0635
Back porch	33	1,04775
Celý snímek	525	16,66875

Tab. 3 – 4: Hodnoty časování pro nižší frekvenci pixelu

Odchylky v hodnotách časování		
	Procentní odchylka	**Absolutní odchylka**
Vertikální frekvence	+0,0868 %	+0,052024747 Hz
Horizontální frekvence	+0,0868 %	+0,027312992 kHz
Frekvence pixelu	-20,5561 %	-5,175 MHz
Linková část	**Procentní odchylka**	**Absolutní odchylka [µs]**
Active video	-0,0867 %	-0,02204568
Front porch	+2,2734 %	+0,014448858
Sync pulse	-0,3490 %	-0,013306852
Back porch	-0,3490 %	-0,006653426
Celá linka	-0,0867 %	-0,0275571
Snímková část	**Procentní odchylka**	**Absolutní odchylka [ms]**
Active video	-0,0867 %	-0,013227408
Front porch	-0,0867 %	-0,000275571
Sync pulse	-0,0867 %	$-5{,}51142 \cdot 10^{-5}$
Back porch	-0,0867 %	-0,000909384
Celý snímek	-0,0867 %	-0,014467478

Tab. 3 – 5: Odchylky v hodnotách časování

4. Externí paměť RAM

RAM (Random Access Memory) je rychlá paměť s libovolným přístupem. Slouží k dočasnému ukládání dat. Jedná se o paměť volatilní, neboli nestálou – pro uchování dat musí být napájena.

V této práci je externí RAM využita pro ukládání obrazových dat, která se nevejdou do operační paměti použitých mikroprocesorů.

4.1. Statická a dynamická RAM

V dnešní době se ujaly dva typy RAM – dynamické, které jsou velmi populární díky své ceně a relativně dobrým vlastnostem, a statické, které jsou znatelně dražší, zato ale rychlejší, nepotřebují podpůrné obvody pro uchování informace a jednodušeji se ovládají.

Většina dnešních dynamických pamětí má podpůrné obvody integrované v sobě. Tyto paměti se ovládají pomocí příkazů. Nejdříve se musí inicializovat dle dané sekvence a až poté mohou přijímat ostatní příkazy pro čtení a zápis

Prvotním plánem bylo využít pro tuto práci paměť dynamickou, konkrétně synchronní dynamickou RAM (SDRAM) a její dávkový režim (Burst mode). Tento režim funguje tak, že se paměti pošle příkaz pro zapnutí dávkového režimu a paměť automaticky při každém taktu vydává data sekvenčně za sebou a není potřeba poskytovat adresu dat. Tato paměť mohla odlehčit složitosti hardwaru a softwaru, bohužel se mi ji ale nepodařilo zprovoznit, kvůli chybějící technické dokumentaci a pouhému odhadování funkce a rozložení pinů z podobných (pravděpodobně kompatibilních) dokumentací.

Použil jsem tedy paměť statickou. Oproti dynamické paměti nepodporuje žádné příkazy, ale jen jednoduchý zápis a čtení. Pro rychlé sekvenční čtení bylo potřeba implementovat hardwarové řešení.

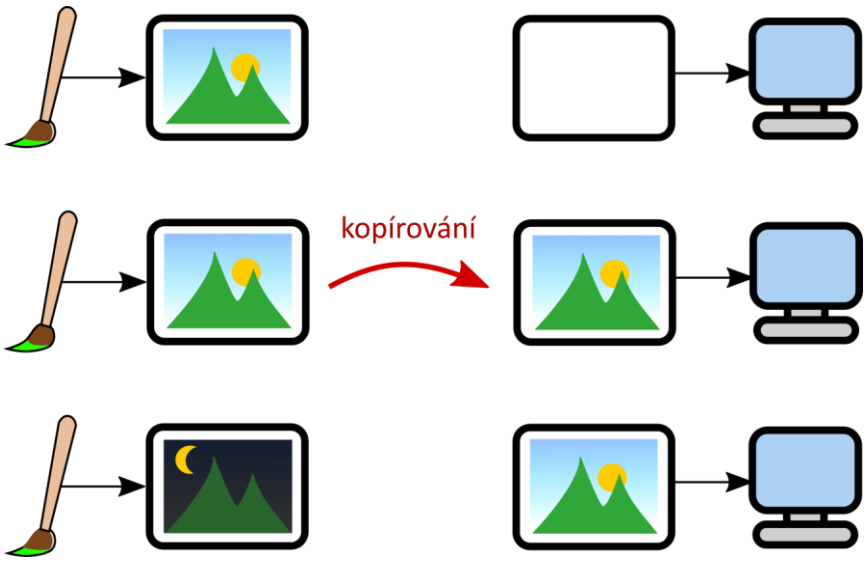
4.2. Double buffering

Double buffering je v počítačové grafice technika využívající dvou pamětí pro vykreslování a zobrazování obrazu. Celkové vykreslování se tak značně urychlí a zabrání se tím trhání obrazu a případným artefaktům. Je to metoda používaná dodnes a pravděpodobně se bude používat ještě dlouho, protože nejsou závažné důvody, proč by se měla nahrazovat.

V této práci bylo použití dvou nezávislých pamětí jediné rozumné řešení, protože rychlosti mikroprocesorů nejsou příliš vysoké a zobrazování obsahu paměti na monitor zabírá přes 80 % celkového času práce s pamětí.

4.2.1. Klasický double buffering

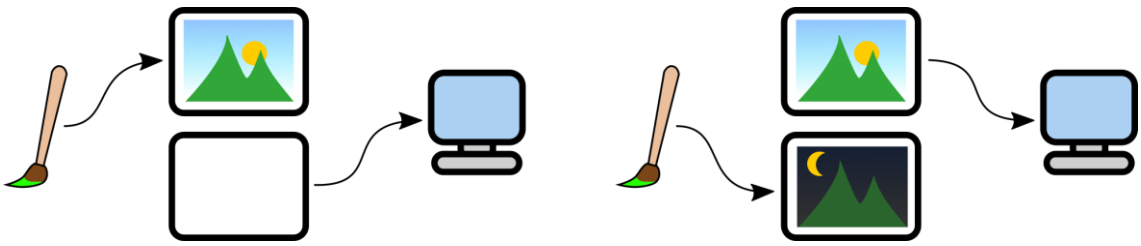
Klasický double buffering je založen na tom, že jedna paměť je primární, z té se obraz zobrazuje přímo na monitor, a druhá, tzv. „back buffer“, do které se obraz vykresluje. Po dokončení vykreslování se celý obsah back bufferu rychle překopíruje do primární paměti a back buffer je připraven pro vykreslování dalšího snímku.



Obr. 4 – 1: Ilustrace double bufferingu

4.2.2. Page flipping

Další možností realizace double bufferingu je page flipping. Ten využívá toho, že má dostupné dvě ekvivalentní paměti, ze kterých lze zobrazovat obsah na monitor. V počátečním stavu se z jedné paměti zobrazuje obraz a do druhé se zapisuje. Po dokončení zápisu se obě paměti prohodí, vykreslený snímek se zobrazí na monitoru a starý snímek je přepisován snímkem následujícím.



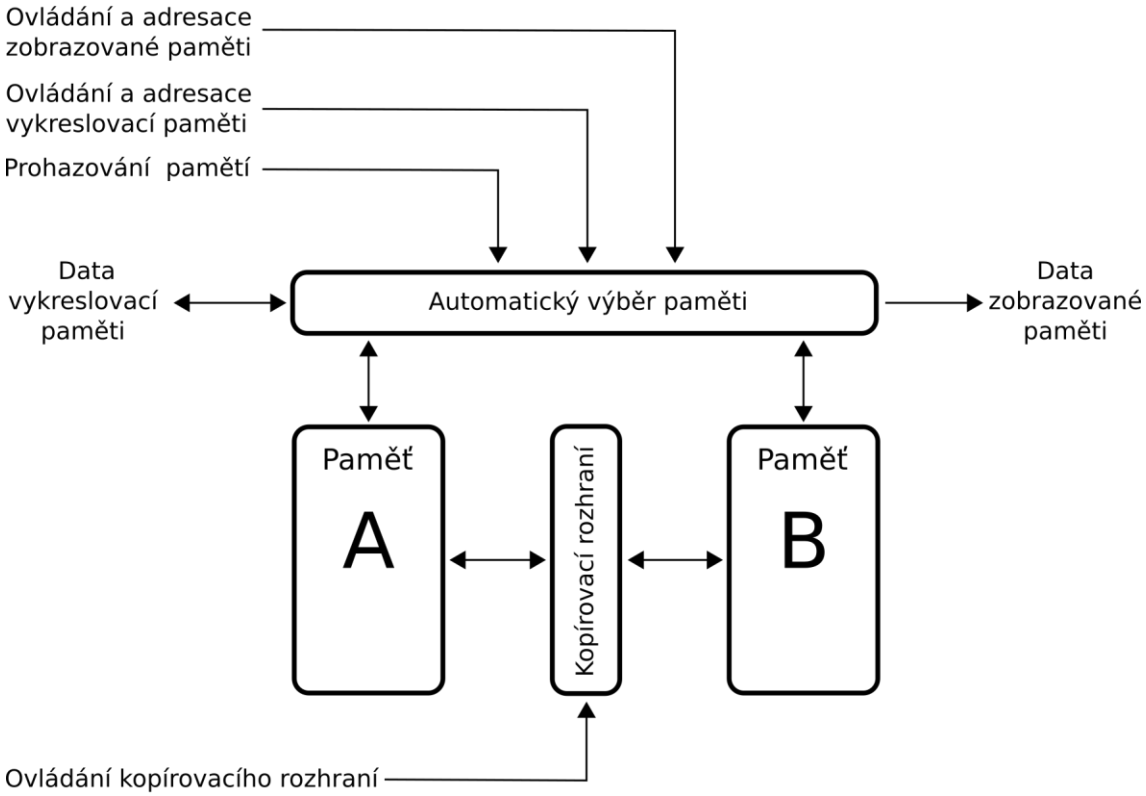
Obr. 4 – 2: Ilustrace page flippingu

4.3. HARDWAROVÁ REALIZACE

RAM paměť vykreslovací jednotky jsem realizoval jako modul se dvěma nezávislými statickými paměťmi. Tento modul podporuje jak klasický double buffering, tak page flipping.

Na page flipping jsem se při návrhu modulu zaměřil. Výsledkem je možnost prohození pamětí velmi rychle, téměř kdykoliv, v řádu jednotek µs. Z praktických důvodů je ale prohazování omezeno na obnovovací frekvenci displeje (60 Hz), aby se zabránilo trhání obrazu.

Klasický double buffering je jen jako nadstavba, protože je možné kopírovat jen omezený počet linek za dobu jednoho snímku. Lze jej ale využít pro postupné zobrazování snímku v průběhu jeho vykreslování. Kopírování je implementováno oboustranně, kvůli page flippingu. Je tedy teoreticky možné kopírovat obsah zobrazovaného snímku do back bufferu.



Obr. 4 – 3: Blokové schéma SRAM modulu

4.3.1. Realizace rychlého sekvenčního čtení

Hlavním nedostatkem statické RAM (SRAM) byla absence rychlého sekvenčního čtení, při kterém by v průběhu nebylo nutné zadávat adresu dat. Dále bylo potřeba redukovat množství adresových pinů, kvůli nedostatku pinů u zobrazovacího mikroprocesoru. Zřejmým a logickým řešením bylo použití čítače impulsů, který by adresu dat postupně zvyšoval. Při testování paměťového modulu s čítačem se ale objevily problémy. Tyto problémy byly spojené s nedostatečnou rychlostí čítačů impulsů. Ty totiž v průběhu přičítání impulsu mají na výstupu nekorektní hodnoty, které mohly ovlivnit zobrazení až 7 pixelů. Příčinou nekorektních hodnot je to, že se při změně hodnota neaktualizuje naráz, ale postupně po jednom bitu od LSB po MSB, jak můžeme vidět na příkladu v Tab. 4 – 1.

Hodnota na výstupu		Poznámka
binárně	dekadicky	
00011111	**31**	**Správná hodnota před přičtením impulsu**
00011110	30	Nekorektní hodnota v průběhu přičítání
00011100	28	Nekorektní hodnota v průběhu přičítání
00011000	24	Nekorektní hodnota v průběhu přičítání
00010000	16	Nekorektní hodnota v průběhu přičítání
00000000	0	Nekorektní hodnota v průběhu přičítání
00100000	**32**	**Správná hodnota v průběhu přičítání**
00100000	32	Správná hodnota v průběhu přičítání
00100000	32	Správná hodnota po přičtení impulsu

Tab. 4 – 1: Příklad postupné aktualizace hodnoty čítače impulsů

Tento problém se mohl vyřešit přidáním bufferu. Tím by se ale zbytečně zvýšila složitost ovládání. I přes vyřešení chyby by ale měla nedostatečná rychlost čítače velký vliv na maximální možné rozlišení, a proto jsem se pro tuto cestu řešení problému nerozhodl.

Jako finální řešení se ukázalo použití posuvného registru. Ten umožnuje jak rychlé sekvenční čtení, tak i o něco pomalejší čtení z libovolných adres. Toho by nešlo pomocí čítače impulsů dosáhnout.

Klíčem k rychlému sekvenčnímu čtení je posouvání hodnoty adresy jen o jeden bit. Z toho důvodu nešlo použít standardní posloupnost adres od 0 do „n“. Posloupnost

těchto adres musela splňovat dvě podmínky. Následující hodnota se musela shodovat (kromě LSB) s předchozí hodnotou posunutou o jeden bit doleva, osekanou na délku slova posloupnosti. LSB mohl být pro následující hodnotu libovolně zvolen. Matematicky lze tuto podmínku vyjádřit takto:

$$a_{n+1} = [(2 \cdot a_n + 1) \bmod 2^x] \vee [(2 \cdot a_n) \bmod 2^x],$$

kde proměnná *x* značí délku slova posloupnosti v bitech.

Druhou podmínkou bylo, že posloupnost musí obsahovat všechny možné hodnoty pro danou délku slova. Z toho také vyplývá, že se hodnoty v posloupnosti nesmí opakovat. *Příklad: Pro 8bitovou délku slova musí posloupnost obsahovat 256 různých hodnot.*

Tato posloupnost byla hledána pomocí počítačového programu metodou pokus – omyl, dokud neplatily obě podmínky. Tuto metodu jsem zvolil pro její jednoduchost a také proto, že pro délku adresy 8 bitů byl maximální počet iterací přijatelný.

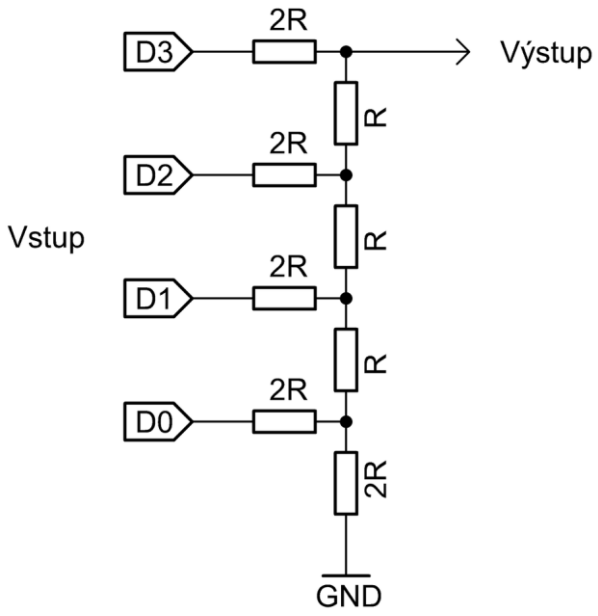
5. D/A PŘEVODNÍK

Úkolem D/A převodníku, dále jen DAC (digital-to-analog converter), je převod digitální hodnoty na hodnotu analogovou.

Existují 3 základní typy převodníků. Prvním je DAC využívající pulzní modulaci, je možné ho používat jen na nízkých frekvencích a není příliš přesný. Jeho hlavní výhoda je možnost softwarové implementace v mikroprocesoru s minimálním množstvím přidaného hardwaru. Dalším typem je binárně vyvážený DAC. Je poměrně jednoduchý a rychlý. Jeho nevýhodou je ovšem nutnost použití velkého rozsahu různých hodnot odporů. To může přinášet problémy jak s šumem u velkých hodnot odporů, tak s dostupností všech potřebných hodnot dostatečně precizních odporů. Posledním typem je převodník s odporovou sítí R-2R. Tomuto typu je věnována následující kapitola.

5.1. R-2R DAC

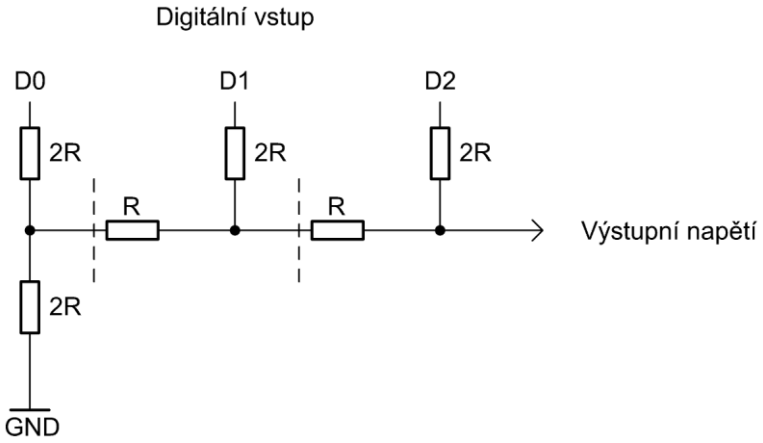
Tento typ převodníku je jedním z nejpoužívanějších. Mezi jeho výhody patří rychlost, jednoduchost a přesnost. Využívá pouze dvou hodnot odporů R a 2R. Pokud hodnotu 2R nahradíme sériovou kombinací dvou odporů hodnoty R, můžeme celý převodník realizovat jen s jednou hodnotou odporu. Další výhodou je to, že výstupní impedance je rovna hodnotě R, proto je velmi jednoduché zahrnout tento převodník do jakéhokoliv návrhu. Je také velmi snadno škálovatelný na libovolnou šířku vstupních dat.



Obr. 5 – 1: Schéma 4bitového R-2R DAC

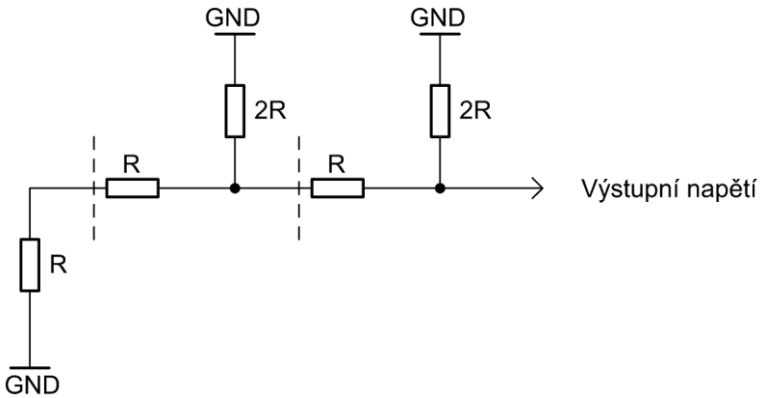
5.1.1. ANALÝZA FUNKCE PŘEVODNÍKU

Zde si na 3bitovém R-2R převodníku ukážeme jeho funkci a zjistíme velikost jeho impedance.



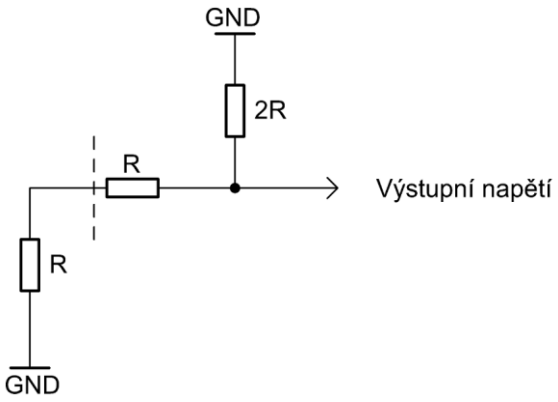
Obr. 5 – 2: Schéma 3bitového R-2R DAC určeného pro analýzu

Chceme-li vypočítat výstupní impedanci převodníku, můžeme zanedbat všechna vstupní napětí. Impedance na nich není závislá. Pokud tedy budeme předpokládat, že na všech vstupech je nulové napětí, můžeme začít zjednodušovat schéma nahrazováním sériových či paralelních kombinací odporů jiným odporem. Vycházíme z toho, že odpor 2R má dvojnásobnou hodnotu odporu R. Nahradíme-li tak část před první přerušovanou čarou, získáme následující schéma.



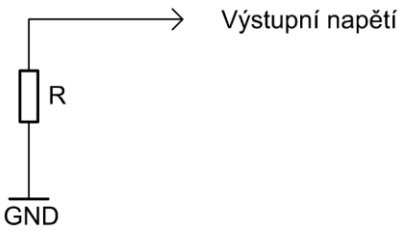
Obr. 5 – 3: Analýza velikosti impedance 3bitového R-2R DAC (1. krok)

Můžeme si všimnout, že vznikla stejná situace jako při nahrazení první kombinace. Pokračujeme ve zjednodušování části před další přerušovanou čarou.



Obr. 5 – 4: Analýza velikosti impedance 3bitového R-2R DAC (2. krok)

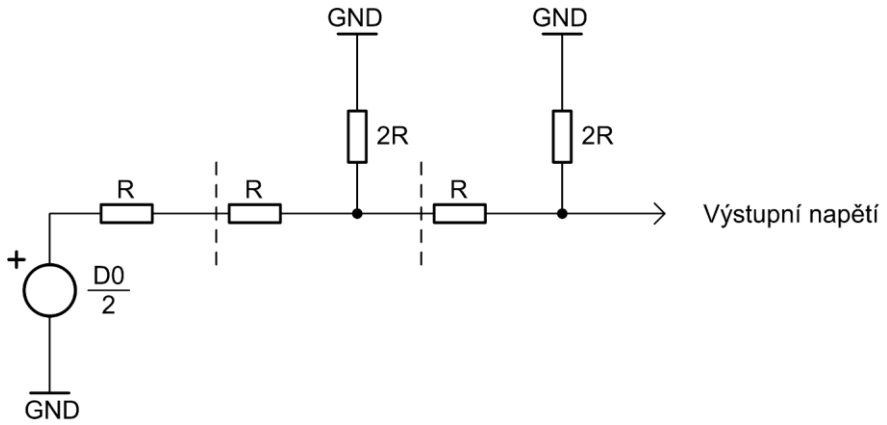
Opět nastává stejná situace. Zbývá zde poslední sériová a paralelní kombinace odporů. Provedeme poslední krok zjednodušení.



Obr. 5 – 5: Analýza velikosti impedance 3bitového R-2R DAC (3. krok)

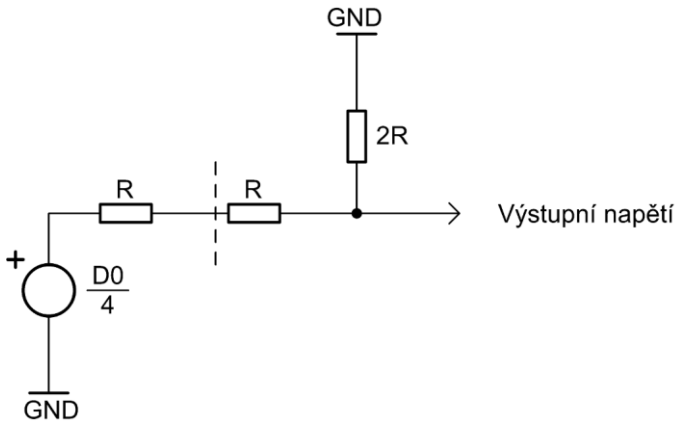
Získali jsme zcela zjednodušené schéma převodníku, a můžeme tedy konstatovat, že výstupní **impedance je rovna odporu R**.

Dále si ukážeme obecné zjednodušení schématu převodníku a zjistíme vztah pro velikost výstupního napětí. Využijeme k tomu Théveninovu poučku o náhradním zdroji napětí a princip superpozice. Vycházíme opět ze stejného schématu (Obr. 5 – 2) jako v předchozí analýze impedance. Nejdříve spočítáme příspěvek vstupního napětí D0 do celkového výstupního napětí. Pomocí Théveninovy poučky zjednodušíme část před první přerušovanou čarou. Získáváme následující náhradní schéma.



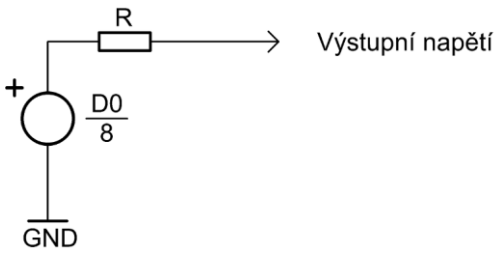
Obr. 5 – 6: Analýza příspěvku napětí D0 do výstupního napětí (1. krok)

Pokračujeme zjednodušením části před druhou přerušovanou čarou.



Obr. 5 – 7: Analýza příspěvku napětí D0 do výstupního napětí (2. krok)

Provedeme poslední krok zjednodušení, a získáme tak velikost příspěvku napětí D0 do celkového výstupního napětí.



Obr. 5 – 8: Analýza příspěvku napětí D0 do výstupního napětí (3. krok)

Zjistili jsme, že vstupní napětí D0 přispívá do celkového výstupního napětí jednou osminou. Stejnou metodou spočítáme příspěvky napětí D1 a D2. Příspěvek D1 bude

dvakrát větší než D0 a příspěvek D2 bude dvakrát větší než D1. Sečtením těchto příspěvků získáváme vzorec pro celkové výstupní napětí.

$$U_{out} = \frac{D0}{8} + \frac{D1}{4} + \frac{D2}{2}$$

Když budeme na digitální vstup přivádět postupně se zvyšující binární hodnotu, výstupní napětí se bude zvyšovat úměrně vzhledem k nastavené hodnotě. Tento typ DAC převodníku je tedy **lineární**.

Pro převodník s libovolnou šířkou vstupních dat můžeme pro výstupní napětí použít následující vzorec, kde n je šířka slova, D_0 značí LSB a D_{n-1} značí MSB.

$$U_{out} = \sum_{i=0}^{n-1} \frac{D_i}{2^{n-i}}$$

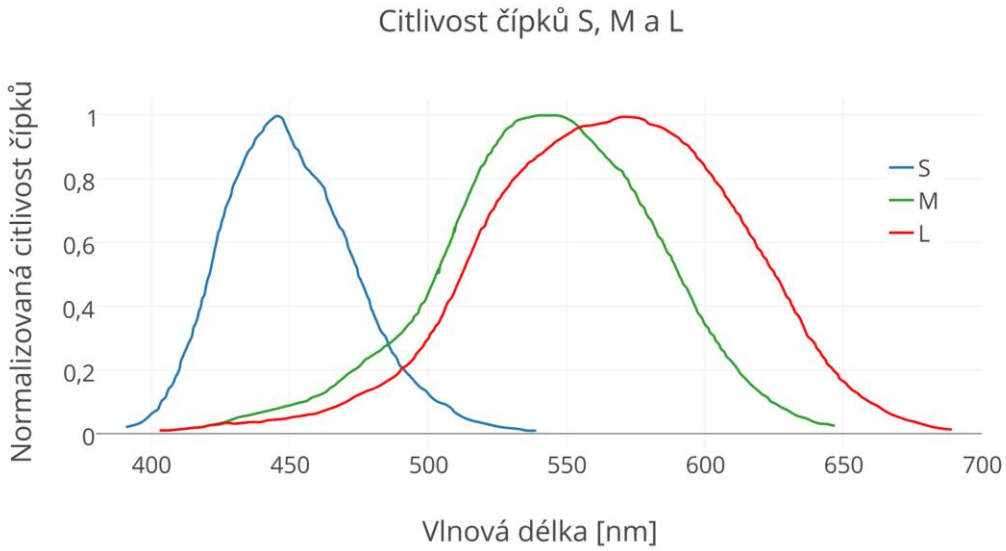
5.2. Barevná hloubka

Při návrhu převodníku bylo potřeba se rozhodnout pro určitou barevnou hloubku. Paměť má místo jen 8 bitů pro jeden pixel. Byla možnost rozdělit počet bitů rovnoměrně mezi červenou, zelenou a modrou barvu. Zůstaly by ale 2 bity nevyužité. Nastala otázka, pro které barvy tedy použít 3 bity a pro kterou jen 2.

5.2.1. Lidské oko

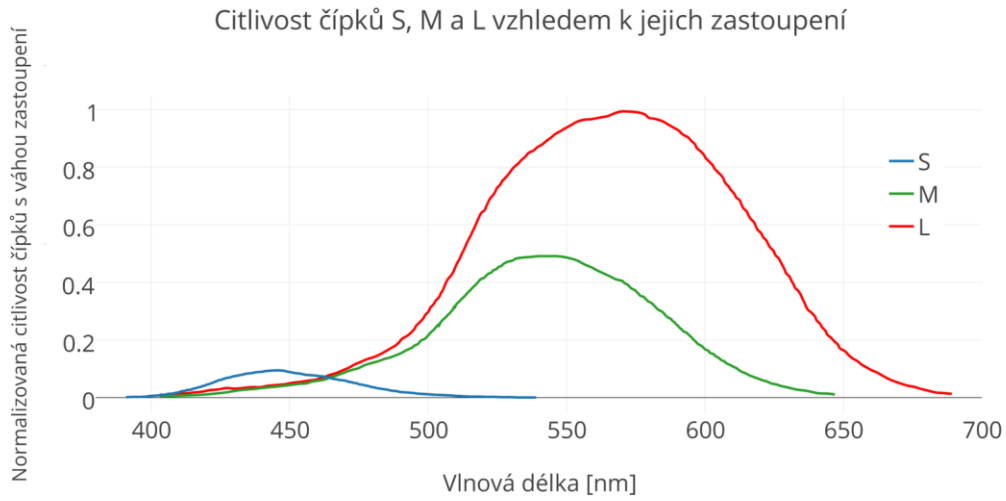
Lidské oko obsahuje dva typy receptorů, které reagují na světlo: tyčinky a čípky. Tyčinky reagují nejvíce za špatných světelných podmínek. Neposkytují informaci o barvě, ale jen o intenzitě světla. Slouží tedy pouze k černobílému nočnímu vidění. Dále nám zajišťují detekci pohybu a periferní vidění. Naopak čípky jsou nejcitlivější na jasném světle a jsou citlivé na barvu. V lidském oku je podstatně více tyčinek (120 milionů) než čípků (6 až 7 milionů).

Lidské čípky se vyskytují ve třech variantách: S, M a L. Čípky S jsou citlivé na krátké vlnové délky (modrá barva), M na střední vlnové délky (zelená barva) a L na dlouhé vlnové délky (červená barva).



Obr. 5 – 9: Citlivost čípků S, M a L

Varianty čípků ale nejsou v lidském oku zastoupeny ve stejném počtu. Lidské oko obsahuje přibližně 63 % čípků L, 31 % čípků M a 6% čípků S.[1]



Obr. 5 – 10: Citlivost čípků S, M a L vzhledem k jejich zastoupení

Dle tohoto grafu můžeme konstatovat, že je lidské oko nejméně citlivé na modrou barvu. Proto jsem v převodníku použil 3 bity pro zelenou a červenou barvu a jen 2 bity pro barvu modrou.

[1] http://www.chm.davidson.edu/vce/coordchem/color.html (volně přeloženo pro potřeby práce)

6. ZOBRAZOVÁNÍ OBRAZU

Zobrazování obrazu řeší zobrazovací mikroprocesor. Jeho prací je generovat synchronizační signály, zajišťovat sekvenční čtení dat ze zobrazované paměti, ovládat double buffering a přijímat příkazy od vykreslovacího mikroprocesoru.

Všechny tyto úkony se musí provádět velmi rychle a ve správný čas (viz kapitola 3.1. Časování). I nepřesnost jediného instrukčního cyklu může rozsynchronizovat celé zobrazování.

Sériová komunikace mezi mikroprocesory probíhá na protokolu UART (viz kapitola 10.1. UART), zabezpečeným paritou (viz kapitola 11.1. Parita).

Kvůli potřebné rychlosti a přesnosti byl psán firmware mikroprocesoru v jazyku symbolických adres (assembly language).

V následujících podkapitolách budou popsány významné části firmwaru.

6.1. KOMPENZACE CYKLŮ PŘERUŠENÍ

Pokud chceme využívat v programu přerušení, musíme počítat s určitou nepřesností při zavolání přerušení. Tato nepřesnost vzniká tím, že při zavolání přerušení je potřeba dokončit poslední instrukci. Instrukce použité v programu mají délku 1 – 4 cyklů. To znamená, že největší odchylka v délkách přerušení může být až 3 cykly.

Řešení, které kompenzuje tyto rozdíly, využívá toho, že jde o přerušení vytvořená časovačem. Po přetečení časovače vzniká přerušení, ale časovač čítá dál při každém cyklu. Načteme-li hodnotu časovače po přerušení, můžeme zjistit přesnou dobu, která uběhla mezi přetečením časovače a skutečným začátkem přerušení. Z této doby si můžeme odvodit, jak dlouho trvala instrukce, která předcházela přerušení. Poté jen stačí kompenzovat cykly tak, aby přerušení trvalo vždy stejnou dobu.

Pokud předpokládáme, že nejdelší instrukce trvá 4 cykly, tak:

- trvá-li předcházející instrukce 1 cyklus, pozastavíme program na 3 cykly,
- trvá-li předcházející instrukce 2 cykly, pozastavíme program na 2 cykly,
- trvá-li předcházející instrukce 3 cykly, pozastavíme program na 1 cyklus,
- trvá-li předcházející instrukce 4 cykly, pokračujeme v programu.

Následuje výňatek z programu, který řeší tuto kompenzaci cyklů.

```
;**** Kompenzace cyklů přerušení ****; - celkem 12 cyklů

 IN   r19,TCNT1L          ; načtení hodnoty časovače
 SUBI r19,9               ; odečtení pozice předchozí instrukce
                          ; (konstanta 9 je zde pouze jako příklad)

 LDI  ZH,high(ukazatel)  ; načtení hodnoty ukazatele pro nepřímý skok
 LDI  ZL,low(ukazatel)

 ADD  ZL,r19              ; přičtení délky instrukce předcházející
 CLR  r19                 ; přerušení k adrese ukazatele
 ADC  ZH,r19

 IJMP                     ; nepřímý skok na adresu ukazatele

ukazatel:                 ; délka instrukce předcházející přerušení:
 NOP                      ; 1 cyklus
 NOP                      ; 2 cykly
 NOP                      ; 3 cykly
                          ; 4 cykly

;**********************************;
```

6.2. RYCHLÉ SEKVENČNÍ ČTENÍ

Rychlé sekvenční čtení je kombinací softwarového a hardwarového řešení (viz kapitola 4.3.1. Realizace rychlého sekvenčního čtení). Úkolem softwaru je provádět co nejrychleji přičítání adres zobrazovaných dat. Je několik metod jak program urychlit. První metodou je rozepsat všechny iterace smyček. Tato metoda sice zabírá velké množství paměti (v konkrétním případě došlo přibližně ke zdvojnásobení objemu celého programu), ale smyčky, které v sobě mají jen pár instrukcí, může významně urychlit. Dalšího urychlení jsem dosáhl tím, že jsem si předem připravil do registrů všechny možné výstupní hodnoty, které může program v daném úseku potřebovat. Celkem se mi povedlo redukovat přičtení adresy zobrazovaných dat na pouhé dvě instrukce. Zobrazovací mikroprocesor pracuje na frekvenci 20 MHz a tyto dvě instrukce trvají dohromady dva cykly. Z toho vyplývá, že se dostáváme na maximální frekvenci pixelu 10 MHz, a to omezuje horizontální rozlišení přibližně na **250 pixelů**. Toto rozlišení se roztáhne a vyplní originálních 640 pixelů.

Jednou instrukcí se na vstup posuvného registru přivede daná hodnota z posloupnosti bitů rychlého sekvenčního čtení (viz kapitola 4.3.1. Realizace rychlého sekvenčního čtení) a druhá instrukce vytvoří hodinový impuls pro posuvný registr, který provede aktualizaci hodnoty adresy.

V následujícím výňatku z programu vidíme začátek rychlého sekvenčního čtení.

```
;r20   adr=0 clk=0
;r21   adr=0 clk=1
;r22   adr=1 clk=0
;r23   adr=1 clk=1

; nastavení hodnot registrů r20-r23
IN   r20,sync_port2
ANDI r20,~((1<<adr_x_bit)|(1<<clk_x_bit))
MOV  r21,r20
ORI  r21,(1<<clk_x_bit)
MOV  r22,r20
ORI  r22,(1<<adr_x_bit)
MOV  r23,r22
ORI  r23,(1<<clk_x_bit)

; postupné zvyšování adresy dat
OUT  sync_port2,r20        ; vstup registru = 0, clock = 0
OUT  sync_port2,r21 ;0     ; vstup registru = 0, clock = 1

OUT  sync_port2,r22        ; vstup registru = 1, clock = 0
OUT  sync_port2,r23 ;1     ; vstup registru = 1, clock = 1

OUT  sync_port2,r22        ; vstup registru = 1, clock = 0
OUT  sync_port2,r23 ;1     ; vstup registru = 1, clock = 1

OUT  sync_port2,r22        ; vstup registru = 1, clock = 0
OUT  sync_port2,r23 ;1     ; vstup registru = 1, clock = 1

OUT  sync_port2,r22        ; vstup registru = 1, clock = 0
OUT  sync_port2,r23 ;1     ; vstup registru = 1, clock = 1

OUT  sync_port2,r22        ; vstup registru = 1, clock = 0
OUT  sync_port2,r23 ;1     ; vstup registru = 1, clock = 1

OUT  sync_port2,r22        ; vstup registru = 1, clock = 0
OUT  sync_port2,r23 ;1     ; vstup registru = 1, clock = 1

OUT  sync_port2,r22        ; vstup registru = 1, clock = 0
OUT  sync_port2,r23 ;1     ; vstup registru = 1, clock = 1

OUT  sync_port2,r22        ; vstup registru = 1, clock = 0
OUT  sync_port2,r23 ;1     ; vstup registru = 1, clock = 1

OUT  sync_port2,r20        ; vstup registru = 0, clock = 0
OUT  sync_port2,r21 ;0     ; vstup registru = 0, clock = 1

...
```

7. Vykreslování obrazu

Vykreslování obrazu je prací vykreslovacího mikroprocesoru. Oproti zobrazování obrazu se u vykreslování nemusí brát příliš ohled na přesnost. Dále taky není stanovena žádná minimální rychlost vykreslování.

Celé vykreslování je ale komplexní a je potřeba ho rozdělit do několika vrstev. Proto je psán software pro vykreslovací mikroprocesor v jazyce C.

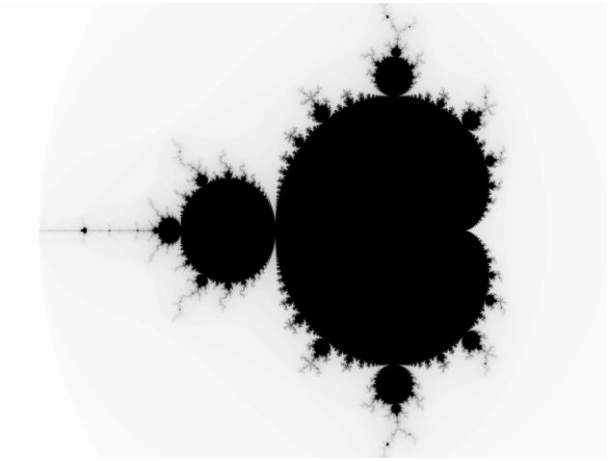
„Vykreslováním“ zde není myšleno pouhé zapisování obrazu do paměti, ale i jeho načítání z externí paměti, popřípadě generování.

Dále musí tento mikroprocesor posílat příkazy zobrazovacímu mikroprocesoru pro ovládání double bufferingu.

7.1. Generování obrazu

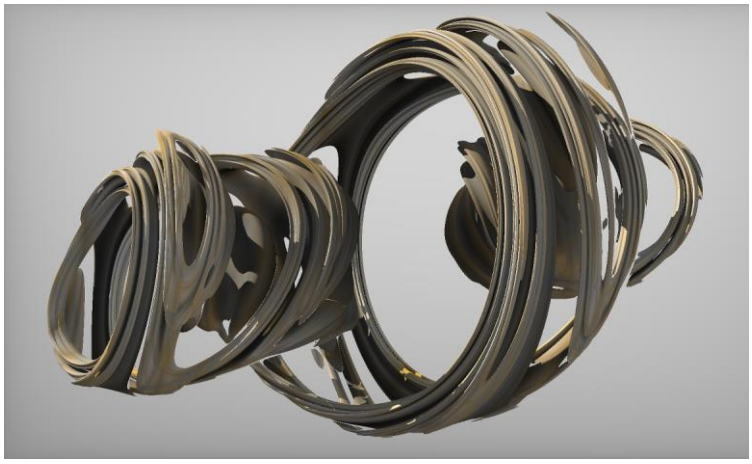
Pod pojmem generování se rozumí, že jde o algoritmické vytváření obrazu, na rozdíl od manuálního přístupu (např. načtení předem vytvořených dat z externího média a následné zobrazení).

Při generování obrazu v této práci máme k dispozici dvě základní proměnné – Jednu vstupní, což je pozice aktuálně vykreslovaného pixelu, a výstupní, kterou je výsledná barvu pixelu. Ačkoliv se pozice pixelu může jevit jako nulová informace, opak je pravdou. S pouhou informací o pozici pixelu se dá vygenerovat spoustu, někdy až nepředstavitelných věcí. Například to mohou být takzvané fraktály. Můžeme říct, že jde o nekonečné, sobě podobné vzory na různých úrovních. Pojem „fraktál“ poprvé definoval matematik Benoit Mandelbrot. Podle něj je také pojmenován jeden typ fraktálu, který objevil – Mandelbrotova množina. Pro výpočet nám stačí znát vstupní pozici pixelu.



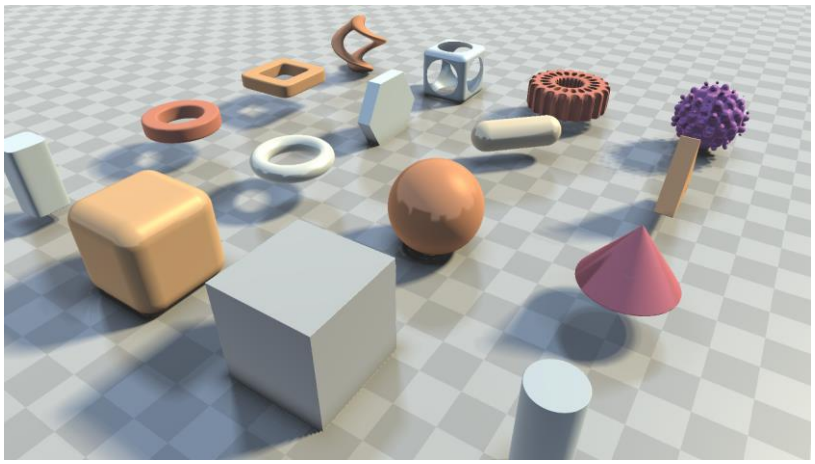
Obr. 7 – 1: Mandelbrotova množina

Z pozice pixelu jdou pomocí matematických výpočtů generovat například i trojrozměrné fraktály.



Obr. 7 – 2: Trojrozměrný fraktál
(„Quaternion“, ©2013 Iñigo Quílez, CC BY-NC-SA 3.0)[2]

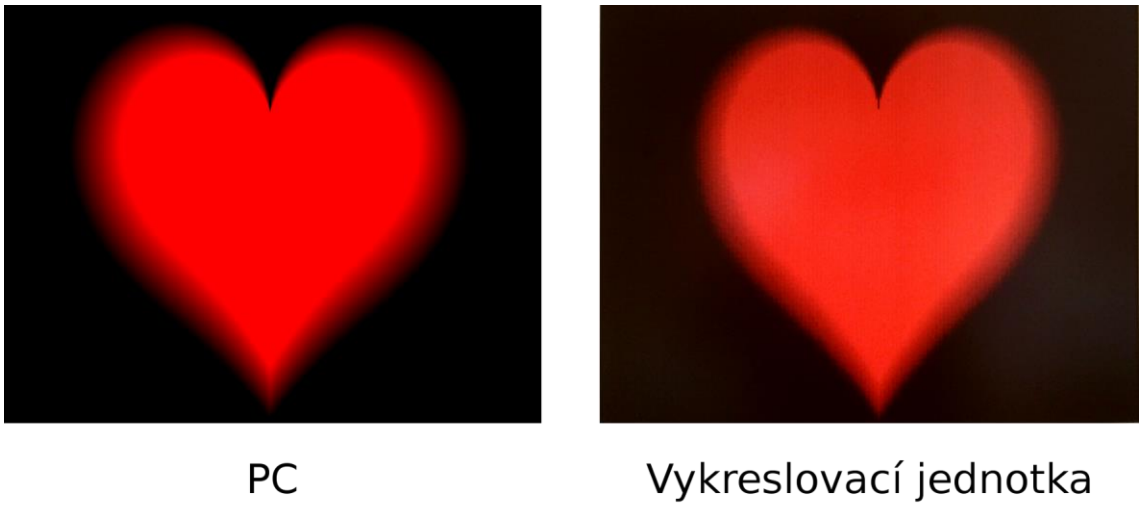
Nebo různé další prostorové objekty metodou zvanou „raymarching“.



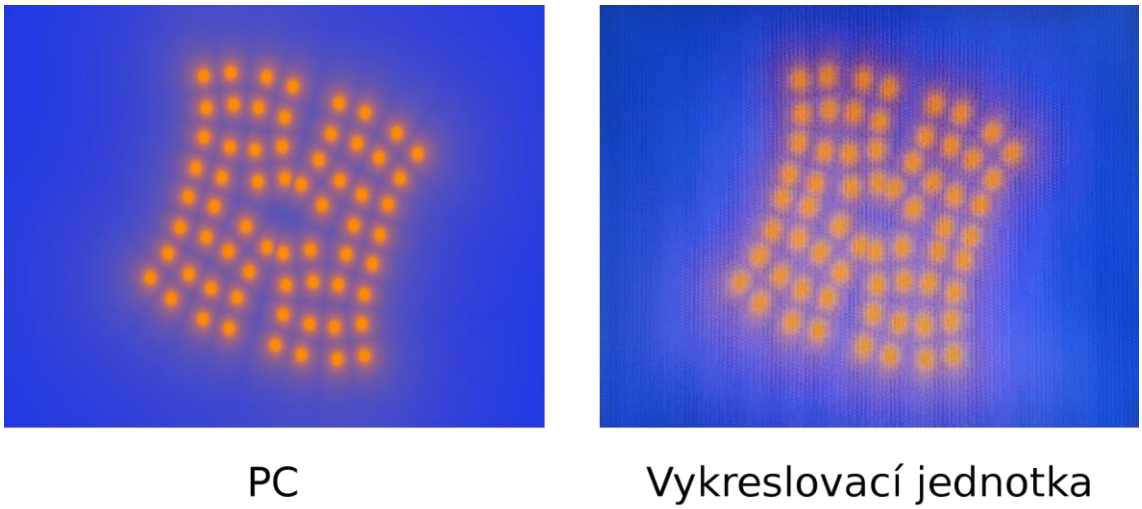
Obr. 7 – 3: Ukázka raymarchingu
(„Raymarching - Primitives“, ©2013 Iñigo Quílez, CC BY-NC-SA 3.0)[3]

V této práci je teoreticky možné tyto obrazy generovat, prakticky ale ne, kvůli nízké přesnosti použité matematické knihovny a nedostatku operační paměti. Navíc by toto náročné vykreslování trvalo velmi dlouho. Některé jednodušší útvary ale vykreslovací jednotka zvládne.

[2] https://www.shadertoy.com/view/MsfGRr (citováno dne: 4.4. 2015)
[3] https://www.shadertoy.com/view/Xds3zN (citováno dne: 4.4. 2015)



Obr. 7 – 4: Generování obrazu na vykreslovací jednotce (srdce)



Obr. 7 – 5: Generování obrazu na vykreslovací jednotce (základní fraktál)

Dále můžeme za cenu nižší rychlosti vykreslování využít zápisu nebo čtení z jakékoliv pozice v paměti určené pro vykreslování. Můžeme tak například využít volnou paměť mimo obraz, která se nezobrazuje, což je 32 x 256 pixelů (to odpovídá 8kilobytům volné paměti), nebo můžeme při výpočtech využívat informace z již vykreslených pixelů. Je nespočet možností, jak tento zápis a čtení z libovolné pozice využít.

7.2. DITHERING

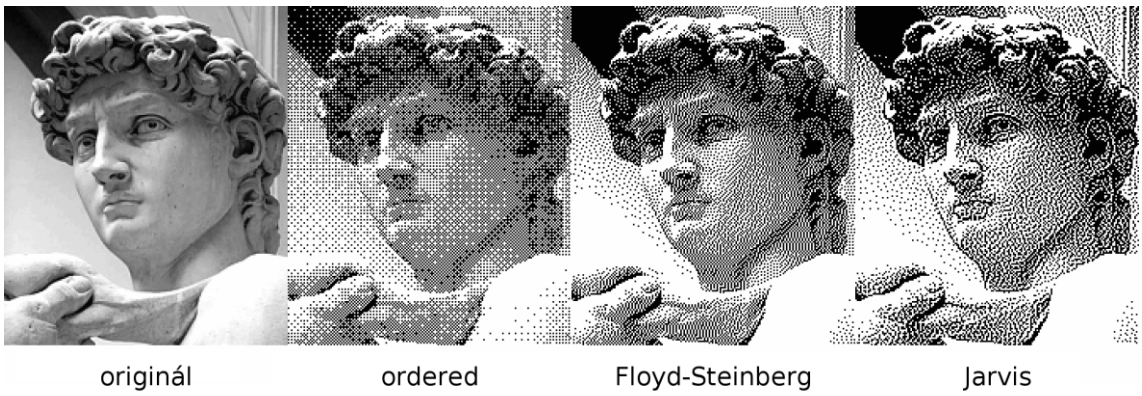
Jelikož většina vstupních či vygenerovaných dat má větší barevnou hloubku, než je možné zobrazit pomocí použitého D/A převodníku, je potřeba tuto barevnou hloubku nějak redukovat.

Nejjednodušší možností je osekat barevnou hloubku na potřebný počet bitů. Tím ale původní barevnou hloubku zcela ztrácíme.

Jedním z řešení tohoto problému je takzvaný dithering. Jde o techniku úmyslného přidávání šumu pro randomizaci kvantizační chyby. Tuto techniku lze použít při jakémkoliv kvantování nějaké hodnoty. Proto najde využití například i v audiu. Dále se ale budeme bavit jen o ditheringu používaném pro převod obrazu do nižší barevné hloubky.

Existuje několik algoritmů ditheringu. Jedním z nejznámějších algoritmů je Floyd-Steinberg dithering. V porovnání s jinými algoritmy má asi nejlépe vypadající výsledky. Pro jeho implementaci jsem se ale nerozhodl, protože by tento algoritmus využíval velké množství operační paměti mikroprocesoru. Vizuálně by na tom byl sice lépe než použitý ordered dithering, ale zato by byl o poznání pomalejší.

Následující obrázek ukazuje porovnání algoritmů ditheringu při redukci z 8bitového obrázku ve stupních šedi na černobílý 1bitový obrázek.

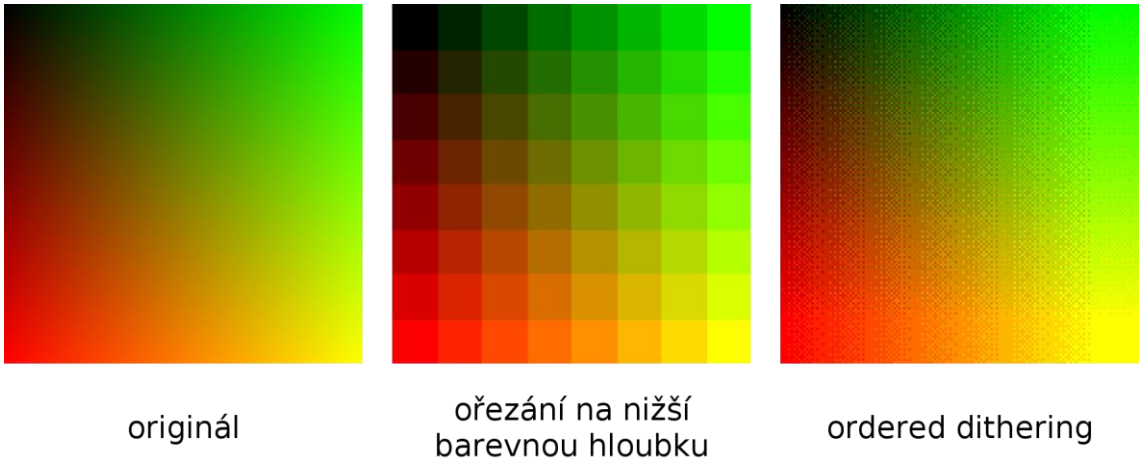


Obr. 7 – 6: Porovnání různých typů algoritmů ditheringu

7.2.1. Ordered dithering

Ordered dithering je velmi jednoduchý a rychlý algoritmus pro realizaci ditheringu. Je často využívaný právě v případech, kdy je potřeba zobrazit obraz větší barevné hloubky, než dokáže dané zařízení zobrazit.

Principem algoritmu je přičtení hodnoty z prahové mapy (threshold mapy) k pixelu. Tento pixel je následně převeden do požadované nižší barevné hloubky. Přidaná hodnota z prahové mapy může zapříčinit kvantování na vyšší hladinu. Tímto se kvantizační chyba rozprostře a výsledný obraz vypadá o poznání lépe.



Obr. 7 – 7: Ordered dithering

Pro ordered dithering můžeme využít několik velikostí prahových map. Ty ovlivňují výsledný vzhled ditheringu. Nejčastěji se používá mapa o velikosti 8x8 nebo 4x4. V této práci je použita následující prahová mapa 4x4.

$$\frac{1}{17}\begin{bmatrix} 1 & 9 & 3 & 11 \\ 13 & 5 & 15 & 7 \\ 4 & 12 & 2 & 10 \\ 16 & 8 & 14 & 6 \end{bmatrix}$$

Pseudokód pro ordered dithering s prahovou mapou 4x4 vypadá takto:

```
q = velikost kvantizační hladiny

pro všechna y:
  pro všechna x:
     pixel = pixel[x][y] + prahová mapa[x mod 4][y mod 4] * q
     nový pixel = osekat_na_nižší_barevnou_hloubku(pixel)
     pixel[x][y] = nový pixel
```

8. NAČÍTÁNÍ DAT Z SD KARTY

Secure Digital karta, nebo také zkráceně SD karta, je nevolatilní paměťové zařízení. SD karta byla zavedena v roce 1999 jako nástupce karet MMC.

Kapacita těchto karet se dnes pohybuje v rozmezí od jednotek až po stovky gigabytů. Dle kapacity se dělí do 3 tříd: Standard-Capacity (SDSC) pro velikosti od 1MB do 2GB včetně, High-Capacity (SDHC) od 2GB do 32GB včetně, eXtended-Capacity (SDXC) od 32GB do 2TB včetně.

V této práci je podporována pouze třída SDSC, ale úpravou softwaru je možné přidat podporu pro třidu SDHC.

SD karty se vyrábí ve třech standardních velikostech: „originální velikost“, mini a micro. Nejčastěji se setkáme s originální velikostí a velikostí micro. Tato práce je kompatibilní s velikostí micro.

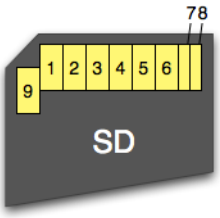
Rychlosti karet se udávají pomocí rychlostních tříd. Rychlosti se v těchto třídách pohybují od 2 MB/s nahoru. Vzhledem k hardwarovým problémům je v této práci bohužel podporována pouze rychlost 16KB/s.

Rychlostní třída	Rychlost
Class 2	2 MB/s
Class 4	4 MB/s
Class 6	6 MB/s
Class 10	10 MB/s
UHS Speed Class 1 (U1)	10 MB/s
UHS Speed Class 3 (U3)	30 MB/s

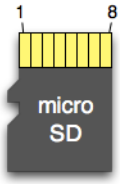
Tab. 8 – 1: Rychlostní třídy SD karet

Chceme-li zařízení podporující SD karty používat ke komerčním účelům, musíme podepsat smlouvu s SD Association (https://www.sdcard.org/) a platit roční poplatky ve výši až 3000 USD.

8.1. Ovládání SD karty



Pin	SD	SPI
1	CD/DAT3	CS
2	CMD	DI
3	VSS1	VSS1
4	VDD	VDD
5	CLK	SCLK
6	VSS2	VSS2
7	DAT0	DO
8	DAT1	X
9	DAT2	X



Pin	SD	SPI
1	DAT2	X
2	CD/DAT3	CS
3	CMD	DI
4	VDD	VDD
5	CLK	SCLK
6	VSS	VSS
7	DAT0	DO
8	DAT1	X

Obr. 8 – 1: Rozložení pinů SD a microSD karty

SD karty jsou ovládány pomocí příkazů, které můžeme posílat pomocí dvou podporovaných protokolů. Prvním protokolem je protokol SD. Ten je buď 1bitový, nebo 4bitový. Je nezbytné ho použít, chceme-li komunikovat s kartou na vysokých rychlostech. Druhým protokolem je protokol SPI, který je hojně rozšířený, jednoduchý a často je hardwarově implementován v mikroprocesorech. V této práci je využit právě protokol SPI, který je podrobněji popsán v kapitole 10.2. SPI.

Pro ovládání SD karty máme k dispozici přibližně 50 příkazů. Většinou si ale vystačíme jen s pár příkazy. Příkazy jsou chráněny proti chybám kódem CRC-7, popsaným v kapitole 11.3. CRC. Struktura příkazu poslaného SD kartě vypadá následovně:

Pozice bitu	47	46	[45:40]	[39:8]	[7:1]	0
Šířka v bitech	1	1	6	32	7	1
Hodnota	0	1	x	x	x	1
Popis	Start bit	Vysílací bit	Číslo příkazu	Argument	CRC-7	Stop bit

** Hodnota „x“ značí proměnnou závislou na posílaném příkazu.*

Tab. 8 – 2: Struktura příkazu pro SD kartu

Ke každému příkazu navrací karta odpověď (response). Jejím hlavním účelem je informovat o případných problémech při přijímání příkazu. Odpověď může také poskytovat informace o kartě (ID karty, ID výrobce, kapacitu karty atd.). Je několik druhů odpovědí. Základním z nich je odpověď R1.

Bit	Význam
7	Pokaždé „0“ – jde o platnou odpověď
6	Chyba argumentu
5	Chyba adresy
4	Chyba v sekvenci příkazů mazání
3	Chyba v CRC
2	Neplatný příkaz
1	Reset mazání
0	V nečinném (idle) režimu

Tab. 8 – 3: Odpověď R1

Proto, aby karta přijímala jakékoliv příkazy, ji musíme nejdříve inicializovat. Pro správnou inicializaci musíme komunikovat přes SPI protokol maximálně s rychlostí 400kHz. Nejdříve kartě pošleme alespoň 74 hodinových impulsů bez jakýchkoliv dat. Poté pošleme příkaz CMD0, který kartě řekne, ať se přepne do nečinného (idle) režimu. Následuje příkaz CMD1, který zahájí inicializační proces. Dále pošleme příkaz CMD59, pro aktivaci CRC7, který se při přechodu do SPI režimu automaticky vypíná. Posledním potřebným příkazem (jde-li o SDSC kartu) je příkaz CMD16, kterým nastavíme velikost jednoho bloku v bytech (zápis a čtení je totiž orientováno blokově). Je vhodné nastavit tuto hodnotu na 512 bytů, jelikož u karet SDHC a SDXC je velikost bloku fixně nastavena na právě na hodnotu 512 bytů.

Tento způsob inicializace karty neodpovídá poslední specifikaci a neřeší detekci typu tříd karet (SDSC, SDHC, SDXC). Dále neposkytuje kartě informaci o napájecím napětí. Proto je vhodné vědět, jakou kartu chceme vložit do vykreslovací jednotky.

Jelikož je tento způsob inicializace funkční a pro tuto práci jsou dostačující karty SDSC, rozhodl jsem se kód inicializace nepřepisovat.

V této práci je pro čtení dat využíváno jen dvou příkazů: CMD18 – zahájení čtení více bloků a CMD12 – ukončení víceblokového čtení. Po poslání příkazu CMD18 vyčkáme na takzvaný „Start Blok Token“, který má hodnotu 0xFE, následuje 512 bytů dat a po nich 2 byty CRC.

Pokud v průběhu posílání dat přijme karta příkaz CMD12 pro zastavení přenosu, přenos je v tomto bodě ukončen a karta může přijímat jakékoliv příkazy. Pokud příkaz CMD12 karta nepřijala, pokračuje posíláním dalších 512 bytů dat s 2byty CRC kódu. Tento proces se opakuje, dokud není příkazem CMD12 zastaven, nebo nenastane-li překročení maximální možné adresy.

CRC kód pro detekci chyb v přenosu dat (CRC-16-CCITT) je podrobněji popsán v kapitole 11.3. CRC.

8.2. Souborový systém FAT

Souborový systém je určitý způsob organizace dat na paměťovém médiu. Je navržen proto, aby bylo snadné přistupovat k potřebným informacím. Říká nám, kde daná informace začíná a končí, jak je velká, nebo pod jakým názvem je uložena.

Mezi souborové systémy patří například tyto: NTFS, ext4, HFS+, FAT32, exFAT a další. FAT je standardním souborovým systémem SD karet a USB flash disků. Pro SD karty do 16MB se používá FAT12, do 2GB FAT16, do 32GB FAT32 a nad 32GB exFAT. Každá verze souborového systému FAT se v něčem mírně liší, ale všechny fungují na stejném principu.

Záznam v oddílu disku se souborovým formátem FAT začíná „Boot sektorem“. Boot sektor je 512 bytů dlouhá část, která sděluje základní informace, podle kterých se dále při čtení či zápisu řídíme.

Souborový systém FAT je adresován dvěma způsoby. Buď je adresován po sektorech, nebo po clusterech. V sektorech se udává například začátek a velikost alokační tabulky. Velikost sektoru se udává v bytech a můžeme ji zjistit v Boot sektoru, ale zpravidla je nastavena na 512 bytů. Soubory a složky jsou adresovány pomocí clusterů. Cluster je definován určitým počtem sektorů (1, 2, 4, 8, 16, 32, 64 nebo 128 sektorů). Jeho velikost vyčteme z Boot sektoru. Clustery se číslují od čísla 2 nahoru.

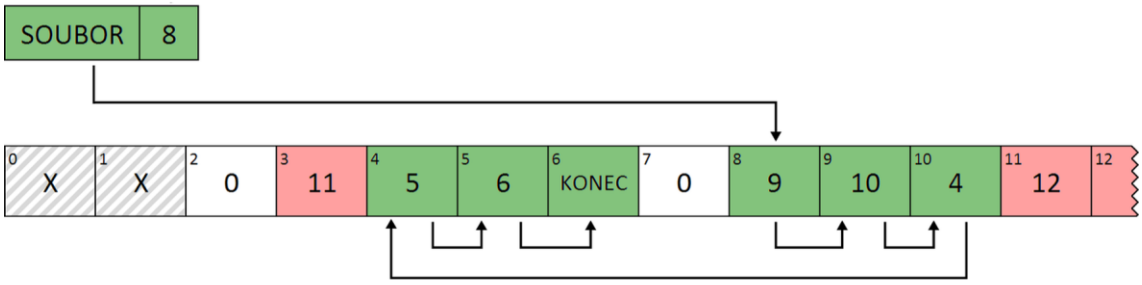
8.2.1. Alokační tabulka

Alokační tabulka je jádrem souborového systému FAT. V alokační tabulce jsou uloženy hodnoty, podle kterých se řídíme při čtení nebo zápisu. Souborový systém exFAT je výjimkou a neřídí se alokační tabulkou, ale takzvanou alokační bitmapou, proto je tento souborový systém v této podkapitole vynechán.

Alokační tabulka obsahuje tolik hodnot, kolik má celý souborový systém clusterů. Každá z těchto hodnot má pevně danou šířku v bytech. Tato šířka se liší podle použitého souborového systému FAT. Šířka této hodnoty je 1,5 bytů pro FAT12 (dvě

hodnoty jsou uloženy do trojice bytů), 2 byty pro FAT16 a 4 byty pro FAT32. Tyto hodnoty číslujeme od 0 až po „n - 1“, kde n je celkový počet clusterů v souborovém systému.

Princip funkce je následující: když čteme data z nějakého clusteru, a dojdeme na konec tohoto clusteru, vyhledáme v alokační tabulce hodnotu s indexem právě dočteného clusteru. Z této hodnoty se dozvíme hodnotu následujícího clusteru, anebo zjistíme, že byl právě dočtený cluster posledním clusterem v řetězci (došli jsme na konec čteného souboru). Při zápisu tento princip funguje podobně. V alokační tabulce hledáme volný cluster, který je označen hodnotou „0“. Po zapsání obsahu tohoto clusteru zapíšeme do hodnoty v alokační tabulce s indexem právě zapsaného clusteru číslo následujícího volného clusteru, který najdeme. Až zapíšeme poslední cluster, do hodnoty v alokační tabulce s indexem tohoto clusteru zapíšeme, že jde o poslední cluster v tomto řetězci clusterů. Jde o speciální hodnotu, která se liší dle použitého typu souborového systému FAT.



Obr. 8 – 2: Ilustrace postupného čtení pořadí clusterů z alokační tabulky.

8.2.2. Souborové záznamy

Souborový systém FAT obsahuje složky a soubory. Záznamy souborů, ale i dalších podsložek se nacházejí ve složkách. Jde v podstatě o stejné záznamy s tím rozdílem, že záznam souboru ukazuje na začátek souboru a záznam složky na začátek složky, která obsahuje další záznamy. V těchto záznamech se dočteme informaci o názvu a příponě souboru (nebo názvu složky), o jeho velikosti a o první pozici clusteru, na které se tento soubor (nebo složka) nachází.

Odsazení (hexadecimálně)	Velikost v bytech	Popis
00	8	Jméno
08	3	Přípona
0B	1	Příznakový byte
0C	8	*Využito pro nadstavbu VFAT*
14	2	První cluster (horní 2 byty)
16	2	Čas
18	2	Datum
1A	2	První cluster (dolní 2 byty)
1C	4	Velikost souboru

Tab. 8 – 4: Struktura souborového záznamu

Bit	Význam
7	*Nevyužito*
6	*Nevyužito*
5	Soubor je archivován
4	Jedná se o složku
3	Nejedná se o záznam ale o název svazku
2	Jde o systémový soubor / složku
1	Jde o neviditelný soubor / složku
0	Soubor je určen jen pro čtení

Tab. 8 – 5: Struktura příznakového bytu

Jelikož tyto záznamy podporují pouze krátká jména souborů s formátováním ASCII, označená jako formát 8.3 (8 znaků jména, tečka a 3 znaky přípony), byl také zaveden speciální formát záznamů LFN (long filename), který umožnuje používat jména (dohromady s příponou a tečkou před příponou) o velikosti až 255 znaků s formátováním UCS-2 (z tohoto formátu vznikl dnešní formát UTF-16). Formát LFN je ale chráněn patentem a proto nemůže být komerčně využíván.

Formát LFN funguje tak, že se využije několik záznamů pro uložení dlouhého jména a ty jsou pak následovány záznamem odkazujícím na samotný soubor nebo složku. U LFN záznamů jsou nastaveny první 4 bity u příznakového bytu na jedničku a tím se zamezí zobrazování těchto záznamů ve většině operačních systémů a programů.

Odsazení (hexadecimálně)	Velikost v bytech	Popis
00	1	Číslo LFN záznamu
01	10	5 znaků UCS-2
0B	1	Příznakový byte
0C	1	*Rezervováno – vždy „0“*
0D	1	Kontrolní součet
0E	12	6 znaků UCS-2
1A	2	*Vždy „0“*
1C	4	2 znaky UCS-2

Tab. 8 – 6: Struktura LFN záznamu

V čísle LFN záznamu je 7. bit (MSB) využit pro označení smazaného záznamu. Pokud jde o formát 8.3, tak ten využívá pro označení smazaného záznamu prvního znaku ve jméně, který při smazání souboru přepíše na hexadecimální hodnotu E5. Dále je v čísle LFN záznamu využit 6. bit pro označení posledního LFN záznamu. Ve skutečnosti je ovšem poslední záznam první, protože jsou LFN záznamy číslovány sestupně.

Záznamy ve složce:
Normální záznam 1
Normální záznam 2
LFN pro normální záznam 3 (3. část)
LFN pro normální záznam 3 (2. část)
LFN pro normální záznam 3 (1. část)
Normální záznam 3
Normální záznam 4

Tab. 8 – 7: Příklad záznamů LFN ve složce

9. Zpracování vstupních dat

Načtená data z SD karty jsou dále zpracovávána. V této práci je implementováno zobrazování obrázků formátu BMP a interpretace jednoduchých příkazů.

9.1. Obraz formátu BMP

BMP (bitmapa), známá také jako DIB, je formát pro ukládání rastrových obrázků. BMP obrázek obsahuje hlavičku a obrazová data. Dále může případně obsahovat paletu indexovaných barev a barevný profil (ICC). Hlavička definuje základní parametry obrázku, jako je šířka, výška, bitová hloubka, celková velikost obrázku bytech atd.

Odsazení (hexadecimálně)	**Velikost v bytech**	**Popis**
00	2	Identifikátor – „BM“ v ASCII (42, 4D hexadecimálně)
02	4	Velikost BMP souboru v bytech
06	4	*Rezervováno*
0A	4	Odsazení, kde začínají obrazová data
Následuje druhá část hlavičky, která může mít více typů. Zde je popsán typ „BITMAPINFOHEADER“		
0E	4	Velikost této části hlavičky (40 bytů)
12	4	Šířka bitmapy v pixelech
16	4	Výška bitmapy v pixelech
1A	2	Počet barevných ploch (musí byt „1“)
1C	2	Počet bitů na pixel (barevná hloubka)
1E	4	Použitá komprese („0“ – bez komprese)
22	4	Velikost obrázku (stačí zadat „0“ pro nekomprimovaný obrázek)
26	4	Horizontální rozlišení (pixely na metr)
2A	4	Vertikální rozlišení (pixely na metr)
2E	4	Počet barev v barevné paletě
32	4	Počet důležitých barev (zpravidla ignorována)

Tab. 9 – 1: Hlavička BMP souboru

Po přečtení informací z hlavičky se můžeme přesunout k obrazovým datům. Pixely jsou zapisovány zleva doprava a linky zespod nahoru. Na každé lince se může vyskytovat výplň (padding). Každá linka je vyplněna tak, aby byl počet bytů na lince dělitelný čtyřmi beze zbytku.

Obrazová data					
Pixel [0, v-1]	Pixel [1, v-1]	Pixel [2, v-1]	…	Pixel [š-1, v-1]	Výplň
Pixel [0, v-2]	Pixel [1, v-2]	Pixel [2, v-2]	…	Pixel [š-1, v-2]	Výplň
…					
Pixel [0, 7]	Pixel [1, 7]	Pixel [2, 7]	…	Pixel [š-1, 7]	Výplň
Pixel [0, 6]	Pixel [1, 6]	Pixel [2, 6]	…	Pixel [š-1, 6]	Výplň
Pixel [0, 5]	Pixel [1, 5]	Pixel [2, 5]	…	Pixel [š-1, 5]	Výplň
Pixel [0, 4]	Pixel [1, 4]	Pixel [2, 4]	…	Pixel [š-1, 4]	Výplň
Pixel [0, 3]	Pixel [1, 3]	Pixel [2, 3]	…	Pixel [š-1, 3]	Výplň
Pixel [0, 2]	Pixel [1, 2]	Pixel [2, 2]	…	Pixel [š-1, 2]	Výplň
Pixel [0, 1]	Pixel [1, 1]	Pixel [2, 1]	…	Pixel [š-1, 1]	Výplň
Pixel [0, 0]	Pixel [1, 0]	Pixel [2, 0]	…	Pixel [š-1, 0]	Výplň

Tab. 9 – 2: Znázornění rozložení obrazových dat

9.2. INTERPRETACE PŘÍKAZŮ

V této práci je využito jednoduchých interpretovaných příkazů. Je načítán textový soubor, který tyto příkazy obsahuje. Syntax vychází ze zápisu volání funkcí v jazyce C. Všechny implementované příkazy jsou popsány v přiložené technické dokumentaci. Jsou podporovány jednořádkové a víceřádkové komentáře. Syntax vypadá následovně:

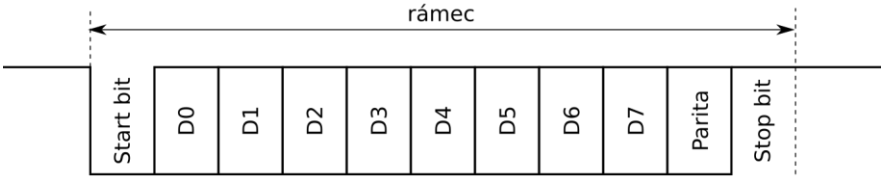
```
// Jednořádkový komentář

/*    Víceřádkový komentář
      Víceřádkový komentář
      Víceřádkový komentář    */

// příkaz (argument1, argument2, ... );

// následují příklady zápisu příkazů

bmp_config(0);
clear_color(0,0,0);
draw_bmp ("images/22.bmp");
repeat();
```

10. POUŽITÉ KOMUNIKAČNÍ PROTOKOLY

10.1. UART

Pod zkratkou UART se skrývá pojem „univerzální asynchronní přijímač/vysílač“ (Universal Asynchronous Receiver/Transmitter). Někdy se využívá synchronní verze UARTu označovaná jako USART. Tento protokol je obyčejně používán ve spojení s komunikačními standardy RS-232, RS-422 nebo RS-485. Jde o protokol sériové komunikace. Pro jednosměrný přenos se využívá jeden vodič a pro obousměrný přenos vodiče dva. Data se posílají v rámci, který začíná start bitem a končí stop bitem. Rámec může obsahovat (zpravidla) 5 až 8 datových bitů a případnou paritu. Parita je podrobněji popsána v kapitole 11.2. Parita.

Takzvaný „Baud rate“ udává rychlost přenosu (bity za sekundu). Zpravidla se využívá několik standardních rychlostí, mezi které patří například baud 1200, 2400, 4800, 9600 a další.

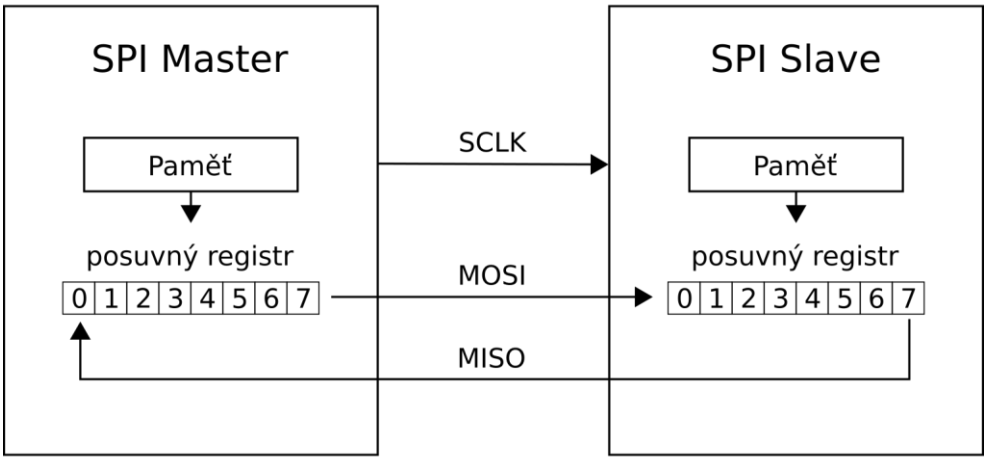


Obr. 10 – 1: Rámec UARTu

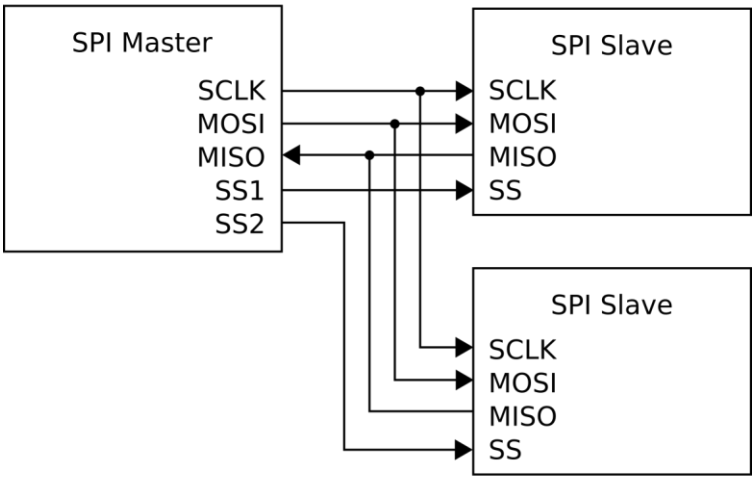
10.2. SPI

SPI (Serial Peripheral Interface) je sériové synchronní rozhraní využívající architekturu master-slave. Používají se vodiče označované jako SCLK – serial clock, SS – slave select, MOSI – master out, slave in, MISO – master in, slave out. Na toto rozhraní je zpravidla připojen jeden „Master“ a několik dalších zařízení „Slave“ (jejich maximální počet závisí na počtu použitých vodičů SS). Master distribuuje hodinový signál a tím také určuje, kdy se budou vysílat informace. Informace se vysílají a přijímají zároveň, jde tedy o režim full duplex. SPI může fungovat i v režimu simplex, je-li zapojen pouze jeden komunikační vodič (MOSI nebo MISO).

Hardwarově je rozhraní SPI řešeno pomocí dvou posuvných registrů, které vodiči MOSI a MISO vzájemně propojíme.



Obr. 10 – 2: Hardwarová realizace SPI



Obr. 10 – 3: Typické propojení více SPI zařízení

11. POUŽITÁ ZABEZPEČENÍ PROTI CHYBÁM

11.1. MAJORITNÍ VYHODNOCENÍ

Nejjednodušším principem zabezpečení je majoritní vyhodnocení. Je založeno na vícenásobném posílání stejné informace. Informace která převažuje, je vyhodnocena jako správná. Například, pošleme-li 5 bitů stejné informace a přijde nám dvakrát „0“ a třikrát „1“, považujeme „1“ za pravdivou informaci.

Pokud si můžeme informaci vyžádat znovu, je to pro nás výhodou. Můžeme tak vyhodnocovat vícebitovou informaci. Tuto informaci si budeme vyžadovat tak dlouho, dokud nám nepřijde dvakrát (nebo i vícekrát) stejná informace po sobě.

Tato metoda zabezpečení chyb je v této práci zkombinována se zabezpečením CRC (více v kapitole 11.3. CRC). Vyžadujeme zde informaci tak dlouho, dokud za sebou nepřijde dvakrát stejné CRC.

11.2. PARITA

Parita je dalším jednoduchým zabezpečením proti chybám. Detekuje až 50% chyb a využívá jen jednoho redundantního bitu. Paritní bit neposkytuje možnost pro korekci dat. Parita může být buď sudá, nebo lichá a vysílač i přijímač musí vědět, který typ parity je používán. Paritní bit je nastaven tak, aby součet všech „1“ datových bitů včetně bitu paritního byl buďto sudý (sudá parita), nebo lichý (lichá parita).

Data	**Přidaný paritní bit**	
	Lichá parita	**Sudá parita**
10010100	0	1
01111110	1	0

Tab. 11 – 1: Příklad paritního bitu

Hodnotu parity lze vypočítat pomocí exkluzivní disjunkce (XOR) mezi všemi bity v daném slově. Parita je vlastně speciálním případem CRC (CRC-1).

11.3. CRC

CRC, neboli cyklický redundantní kód, je komplexní řešení detekce chyb. Využívá dělení vstupního polynomu (posílané zprávy) polynomem generačním v algebře modula 2. Zbytek, který po dělení vznikne, se připojí k posílané zprávě. Přijímač přijatá data (zpráva + zbytek po dělení) znovu vydělí generačním polynomem. Je-li zbytek po tomto dělení nulový, zpráva neobsahuje detekovatelnou chybu.

Pokud je generační polynom zvolen správně, může tento kód detekovat všechny jednobitové chyby, dvoubitové chyby vzdálené od sebe až po délku rovné řádu generačního polynomu, všechny chyby s lichým počtem chybných bitů a většinu shluků chyb.

Je 34 druhů nejpoužívanějších CRC generačních polynomů. V této práci jsou využity polynomy CRC-7:

$$x^7 + x^3 + 1$$

a CRC-16-CCITT:

$$x^{16} + x^{12} + x^5 + 1$$

Vstupní data reprezentují koeficienty vstupního polynomu. Váha bitu odpovídá exponentu členu „x“. Převod tedy vypadá následovně:

$$10011 => 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0 = x^4 + x + 1$$

V následujícím příkladu využijeme vstupní polynom z předchozího příkladu a jako generační polynom si zvolíme $x^2 + 1$. Výpočet CRC je následující: polynomy mezi sebou vydělíme a zjistíme zbytek po dělení,

$$\frac{x^4 + x + 1}{x^2 + 1} = x^2 - 1 + \frac{x + 2}{x^2 + 1}$$

vypíšeme si koeficienty každého členu

$$x + 2 = 1 \cdot x^1 + 2 \cdot x^0$$

a na těchto koeficientech provedeme operaci modulo s číslem 2.

$$(1 \bmod 2) \cdot x^1 + (2 \bmod 2) \cdot x^0 = 1 \cdot x^1 + 0 \cdot x^0 => 10$$

Tyto koeficienty představují výsledek CRC s generačním polynomem $x^2 + 1$ v binární podobě.

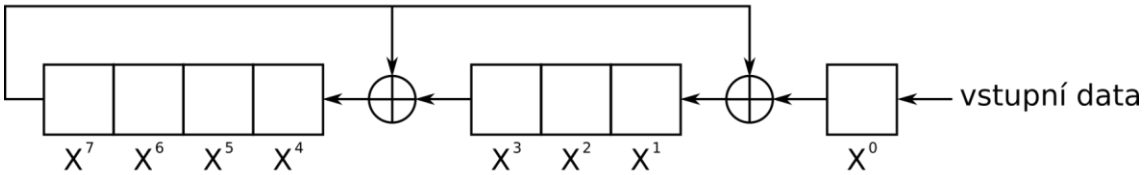
V praxi se pro získání zbytku po dělení polynomu polynomem v algebře modula 2 využívá exkluzivní disjunkce (XOR). V takovém případě pracujeme jen s koeficienty a nemusíme si tak zprávu převádět na polynom.

Výpočet tímto způsobem je následující: Nejdříve ke zprávě přidáme nuly, aby se nám zbytek po dělení „neztratil“. Počet těchto nul je roven řádu generačního polynomu bez jedné. Zprávu a generační polynom zarovnáme tak, aby byly MSB na stejné pozici.

Pokud je ve zprávě na pozici MSB generačního polynomu „1“ provedeme exkluzivní disjunkci mezi zprávou a generačním polynomem, pokud „0“, tak tento krok přeskočíme. Posuneme generační polynom o bit doprava. Tyto kroky opakujeme, dokud není původní zpráva rovna nule. Následně získáváme požadované CRC.

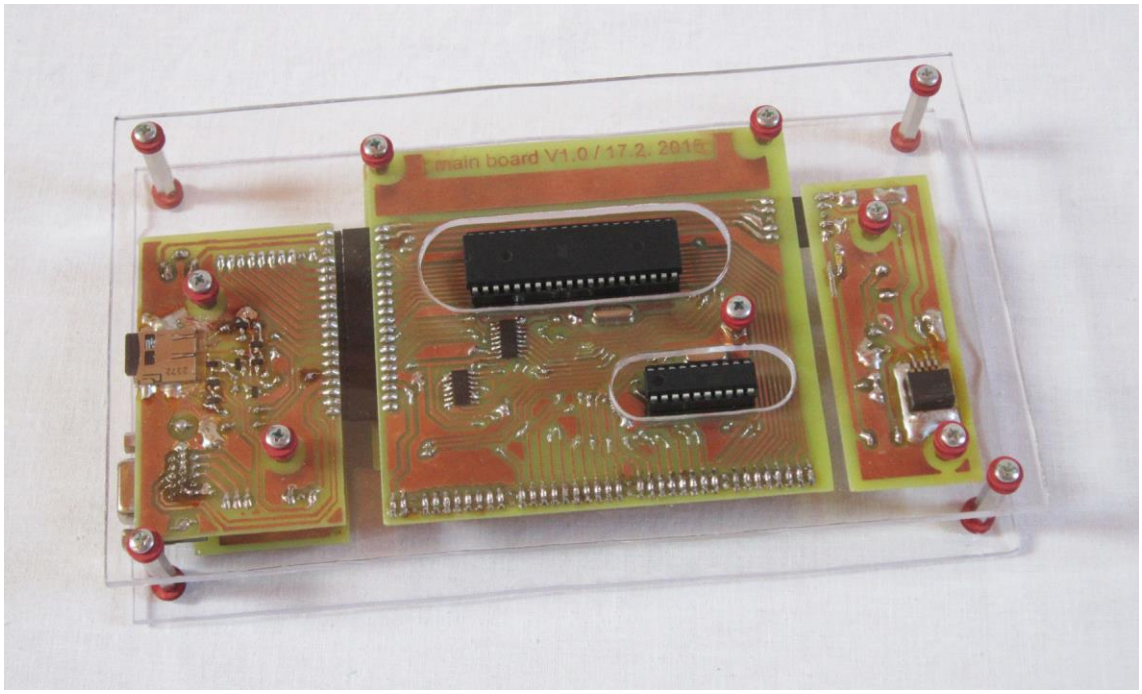
```
1001100     - zpráva
101         - generační polynom
-------
0011100
 101
-------
0001000
  101
-------
0000010     - CRC
```

Celý tento algoritmus se dá realizovat lineárním zpětnovazebným posuvným registrem se zpětnou vazbou zavedenou do členů s koeficientem „1“ přes hradlo XOR.



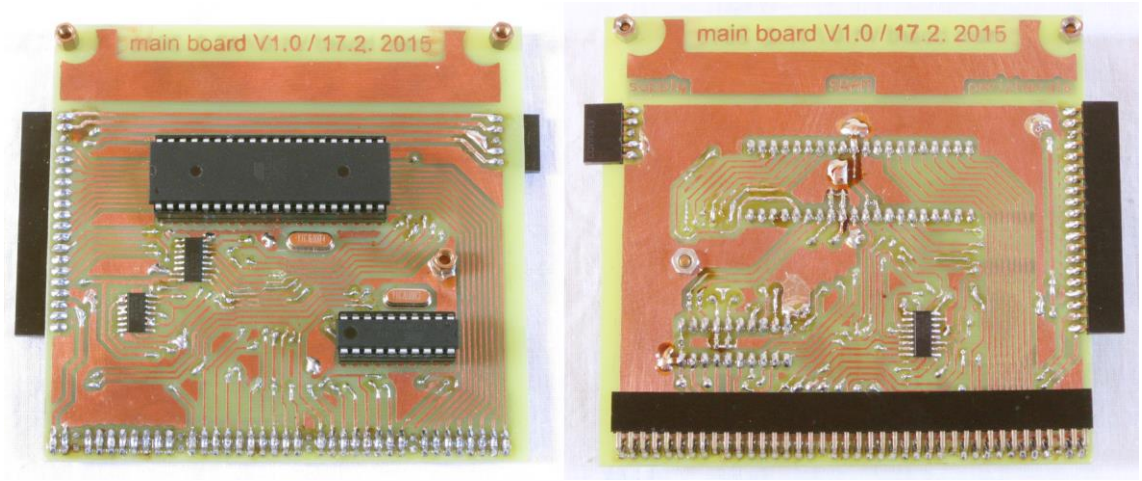
Obr. 11 – 1: Hardwarová realizace generátoru CRC-7

12. FOTODOKUMENTACE

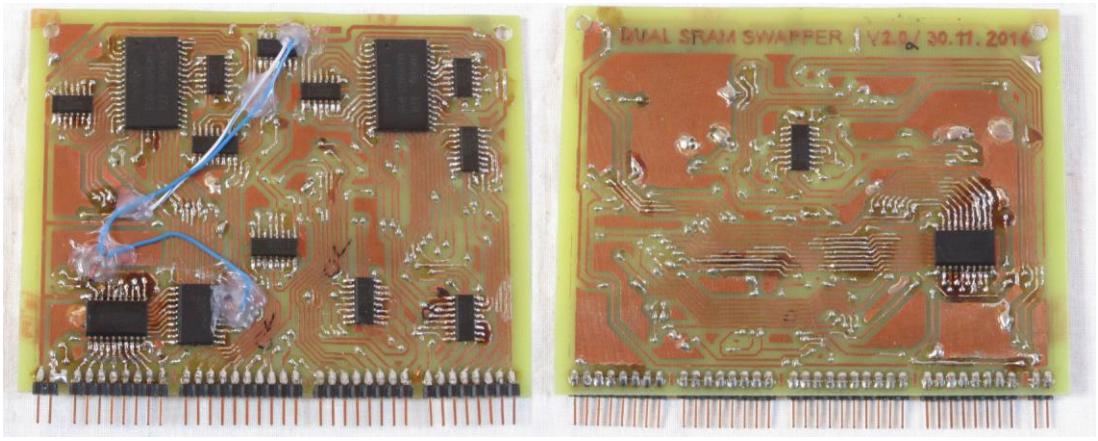


Obr. 12 – 1: Fotografie celého zařízení

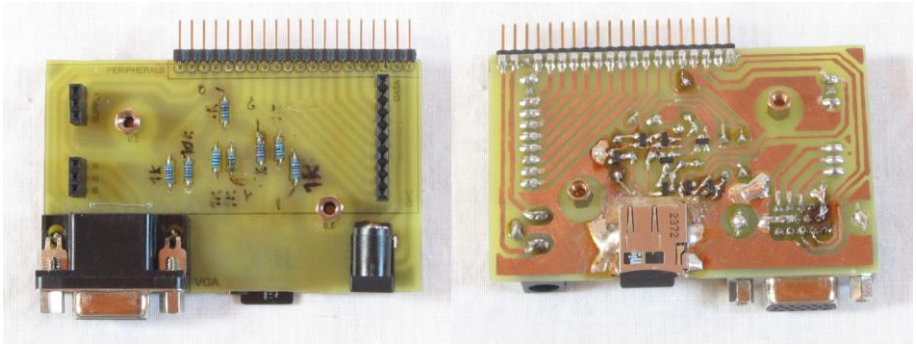
12.1. JEDNOTLIVÉ DESKY



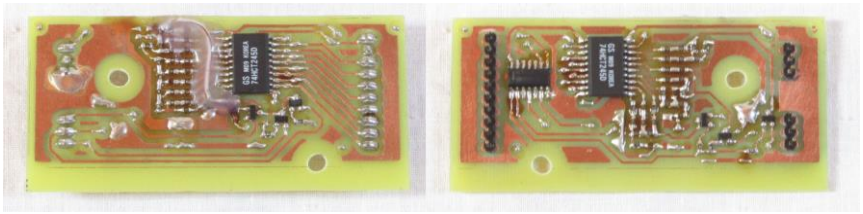
Obr. 12 – 2: Hlavní deska



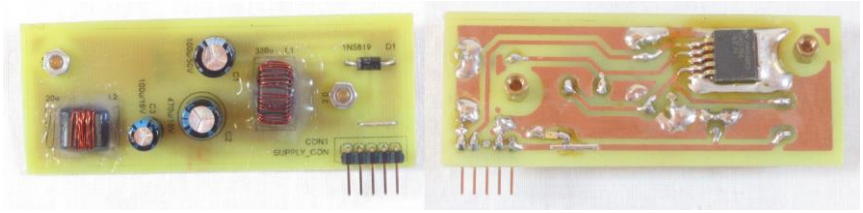
Obr. 12 – 3: SRAM modul



Obr. 12 – 4: Deska se vstupy a výstupy



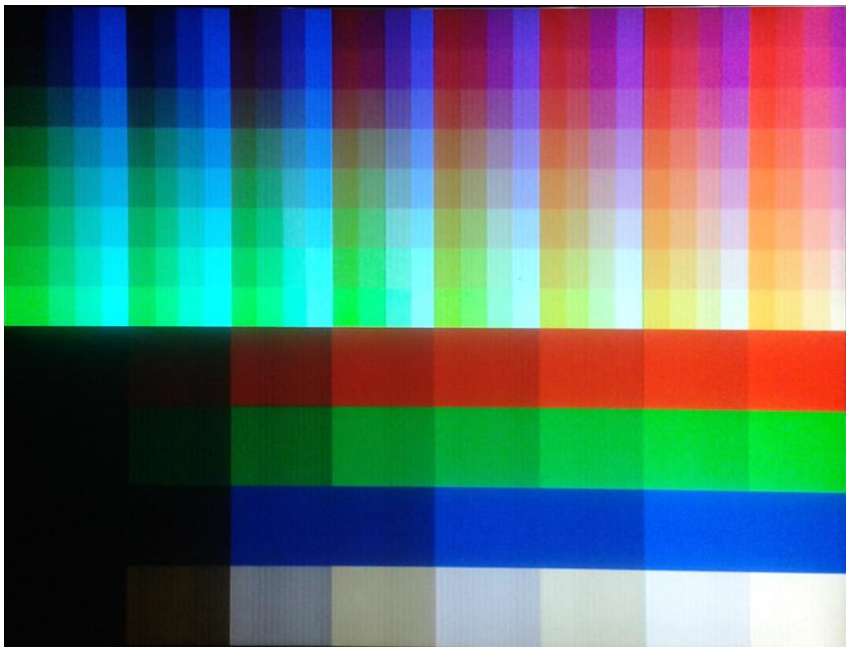
Obr. 12 – 5: DAC



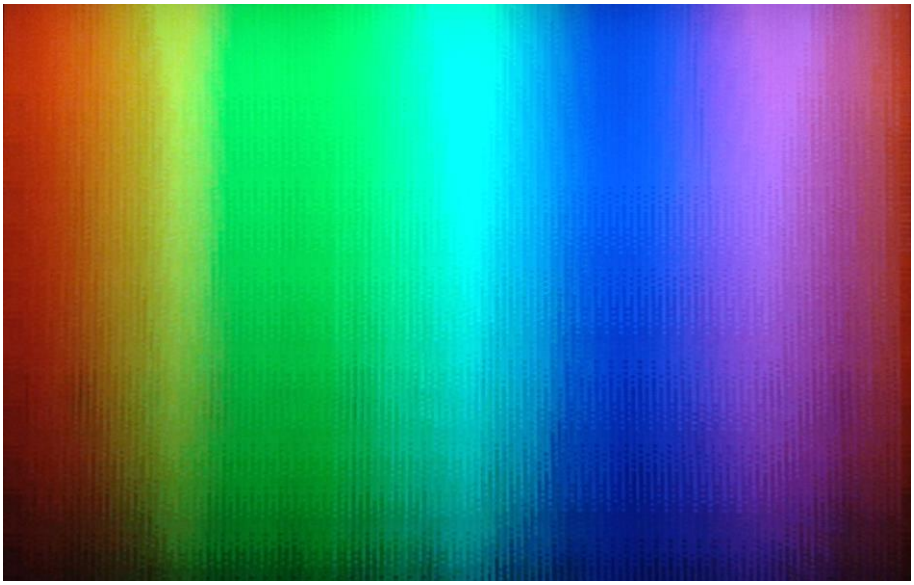
Obr. 12 – 6: Deska s regulátorem napětí

12.2. Činnost zařízení

Obr. 12 – 7: Načtený a zobrazený obrázek

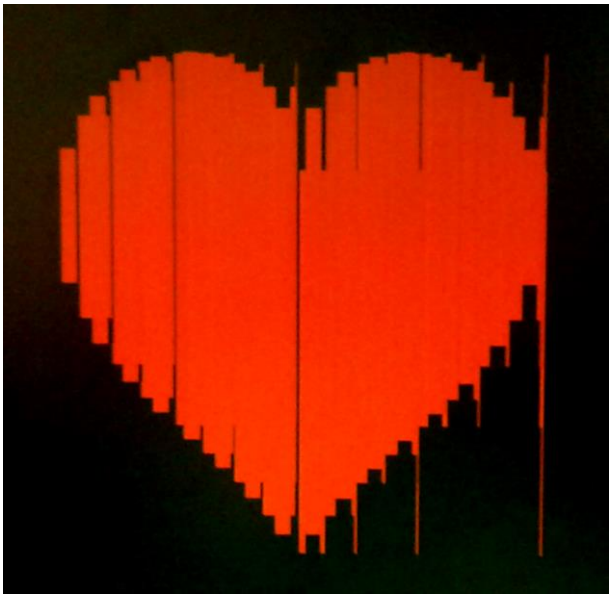


Obr. 12 – 8: Paleta všech zobrazitelných barev

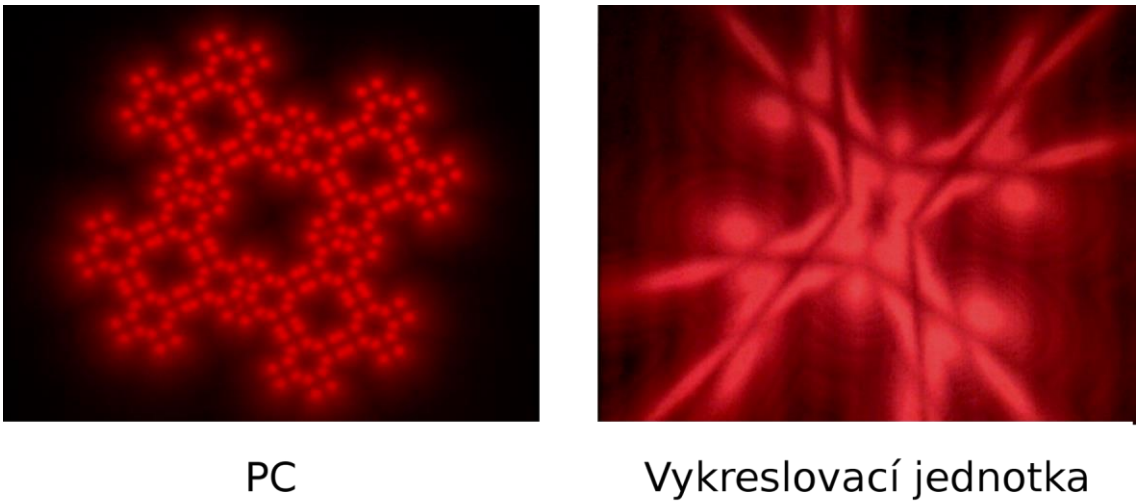


Obr. 12 – 9: HSV paleta se zapnutým ditheringem

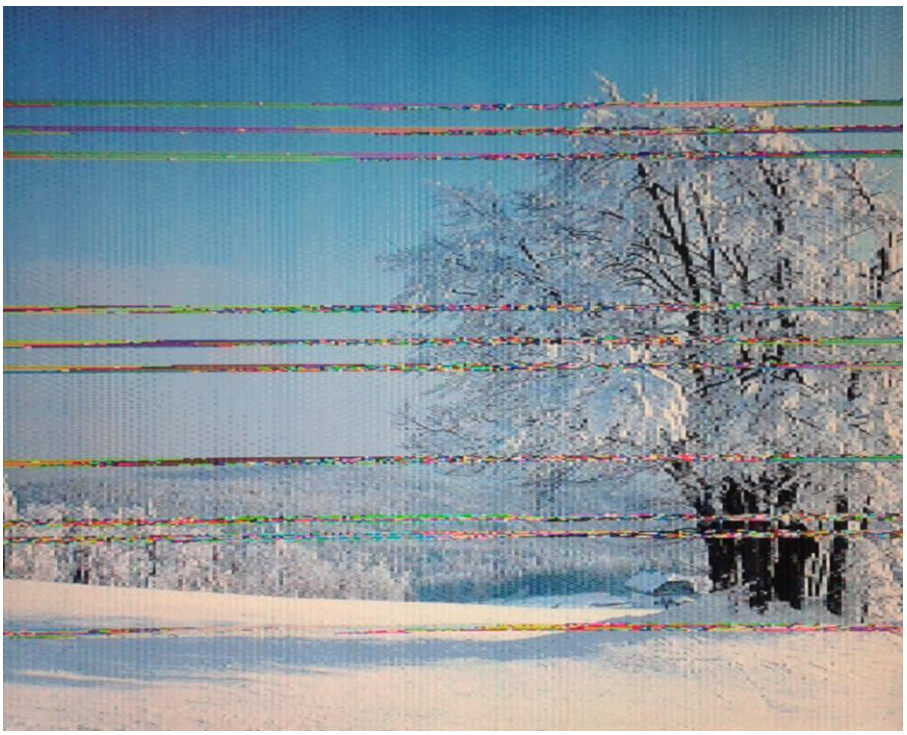
12.3. Chyby při vývoji



Obr. 12 – 10: Chyby v obraze před použitím adresace pomocí posuvných registrů



Obr. 12 – 11 : Problém nepřesnosti matematické knihovny při složitějších výpočtech



Obr. 12 – 12: Chyby v obraze před implementací CRC

ZÁVĚR

Při vývoji tohoto zařízení se naskytlo více problémů, než jsem původně očekával. Proto jsem musel určité části více rozvinout a jiné zase zkrátit. S tímto se ale musí počítat, protože jde o vývoj, a ne o výrobu. Pokud něco vyvíjíme, nevíme, s čím vším se můžeme v průběhu vývoje setkat, a je těžké určit termín dokončení, na rozdíl od výroby, při které jsou všechny postupy předem přesně naplánovány.

Celkově se ale vývoj zařízení vydařil a jeho poslední verze obsahuje jen malé množství chyb, které se nepovedlo odstranit, převážně proto, že jde o chyby, jež se projevují náhodně.

Celá tato práce by mohla být řešena jediným mikroprocesorem s minimálním množstvím přidaného hardwaru, ale mikroprocesory s dostatečným výkonem a pamětí (dnes sice běžně vyráběné) bohužel nejsou dostupné v ČR, nebo jsou dostupné za extrémně vysokou cenu.

Zato byla ale práce pestřejší a věnovala se širšímu okruhu problémů a díky tomu je zde sepsáno více rozmanitých informací pro volné použití v jiných projektech.

DODATEK

Tato práce obsahuje technickou dokumentaci, zdrojové kódy firmwaru obou mikroprocesorů, schémata a návrhy desek plošných spojů.

Celá tato práce včetně veškerých souborů je dostupná také na GitHubu – https://github.com/Aroidzap/mikroprocesorova-vykreslovaci-jednotka.

Zdrojové kódy jsou komentovány v anglickém jazyce, aby byly srozumitelné pro větší okruh lidí, kteří mohou části kódu využít pro svou potřebu.

Celá tato práce, včetně technické dokumentace, schémat a návrhu desek plošných spojů je k dispozici pod licencí CC-BY 4.0. Zdrojové kódy jsou licencovány pod licencí MIT. Přesnější informace o licencích najdete na GitHubu.

SEZNAM POUŽITÉ LITERATURY A STUDIJNÍCH MATERIÁLŮ

Dokumentace a technické specifikace:

1) Atmel: Datasheet, ATmega32(L) Complete, revision Q, 02/2011. Dostupné z: http://www.atmel.com/Images/doc2503.pdf

2) Atmel: Datasheet, ATtiny2313A/4313 Complete, revision B, 10/2011. Dostupné z: http://www.atmel.com/Images/doc8246.pdf

3) Samsung semiconductor: Datasheet, K6X1008C2D, revision 1.0, 09/2003. Dostupné z: http://pdf1.alldatasheet.com/datasheet-pdf/view/77267/SAMSUNG/K6X1008C2D.html

4) NXP Semiconductors: Datasheet, 74HC164; 74HCT164, revision 7, 06/2013. Dostupné z: http://www.nxp.com/documents/data_sheet/74HC_HCT164.pdf

5) NXP Semiconductors: Datasheet, 74HC157; 74HCT157, revision 6, 08/2012. Dostupné z: http://www.nxp.com/documents/data_sheet/74HC_HCT157.pdf

6) NXP Semiconductors: Datasheet, 74HC245; 74HCT245, revision 3, 01/2005. Dostupné z: http://www.nxp.com/documents/data_sheet/74HC_HCT245.pdf

7) SD Group: SD Specifications, Part 1, Physical Layer Simplified Specification, Version 4.10, January 22, 2013. Dostupné z: https://www.sdcard.org/downloads/pls/simplified_specs/part1_410.pdf

Wikipedia – otevřená encyklopedie:

8) http://www.wikipedia.org/

Elementární C knihovny pro AVR:

9) http://www.nongnu.org/avr-libc/user-manual/

Podobná zařízení, VGA:

10) http://tinyvga.com/

11) http://www.lucidscience.com/

12) http://quinndunki.com/blondihacks/?page_id=1761

13) http://www.linusakesson.net/scene/craft/index.php

VGA:

14) http://www.javiervalcarce.eu/html/vga-signal-format-timming-specs-en.html

15) http://pinouts.ru/Video/Vga15_pinout.shtml

Elektronika obecně, převodníky:

16) http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/

17) http://www.maximintegrated.com/en/design/techdocs/app-notes/index.mvp

18) http://www.tek.com/blog/tutorial-digital-analog-conversion-%E2%80%93-r-2r-dac

Double buffering, page flipping:

19) https://docs.oracle.com/javase/tutorial/extra/fullscreen/doublebuf.html

Dithering:

20) http://www.tannerhelland.com/4660/dithering-eleven-algorithms-source-code/

Vnímání barev lidským okem:

21) http://www.chm.davidson.edu/vce/coordchem/color.html

SD karty:

22) http://codeandlife.com/2012/04/02/simple-fat-and-sd-tutorial-part-1/

23) http://elm-chan.org/

24) http://stackoverflow.com/questions/8080718/sdhc-microsd-card-and-spi-initialization

CRC:

25) http://www.barrgroup.com/Embedded-Systems/How-To/CRC-Calculation-C-Code

26) https://www.ccsinfo.com/forum/viewtopic.php?t=24977

FAT:

27) http://www.maverick-os.dk/FileSystemFormats/FAT32_FileSystem.html

28) http://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html

BMP:

29) http://www.fastgraph.com/help/bmp_header_format.html

Seznam obrázků, grafů a tabulek