VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

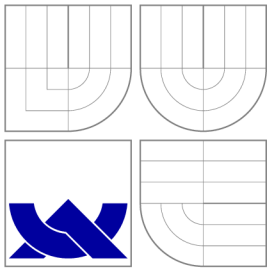
IMPLEMENTACE ALGORITMŮ ZPRACOVÁNÍ OBRAZOVÉHO RASTRU V FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

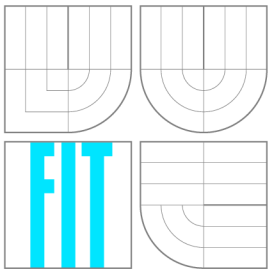
AUTOR PRÁCE
AUTHOR

Bc. VÍT ŠIROKÝ

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

IMPLEMENTACE ALGORITMŮ ZPRACOVÁNÍ OBRAZOVÉHO RASTRU V FPGA

IMPLEMENTATION OF RASTER PROCESSING ALGORITHMS IN FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VÍT ŠIROKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. PAVEL ZEMČÍK

BRNO 2010

Abstrakt

V této práci jde o nezvyklý pohled na implementaci grafických algoritmů v FPGA v kontextu počítačového vidění. Je zde možné najít informace o rastrovém obrazu a jeho zpracování, jeho segmentaci s využitím prahování a adaptitního prahování a také o platformách FPGA a DSP. Také je zde návrh konkrétní realize projektu v kameře Unicam2D a popis jiných možností realizace. Následuje popis implementovaných testů a demonstrace a diskuze jejich výsledků v závěru práce

Abstract

This thesis is about unusal view of implementation of graphic algorithms in FPGA in computer vision context. There are some informations about raster image and raster image operations, raster image segmentation usign threhsholding and adaptive thresholding and FPGA and DSP platforms. Next, there is a concept of the concrete project realization in the Unicam2D camera and description other ways of implementation. Next, there is a description of implemented tests with some demonstration followed by discussion of ressults in the end of the work.

Klíčová slova

Rastrový obraz, Segmentace, Prahování, Adaptivní prahování, FPGA, UnicamD2

Keywords

Raster image, Segmentation, Thresholding, Adaptive-Thresholdinf, FPGA, UnicamD2

Citace

Vít Široký: Implementace algoritmů zpracování
obrazového rastru v FPGA, diplomová práce, Brno, FIT VUT v Brně, 2010

Implementace algoritmů zpracování obrazového rastru v FPGA

Prohlášení

Prohlašuji, že jsem tuto diplomou práci vypracoval samostatně pod vedením pana doc. Dr. Ing Pavla Zemčíka. Veškeré obrázky v této práci jsou buď mým vlastním dílem, nebo uvádím, odkud jsem je převzal.

. . . . . .
Vít Široký
25. května 2010

Poděkování

Mé poděkování patří vedoucímu práce za odborné a pedagogické vedení a Jirkovi Šustkovi z firmy CAMEA za ochotné poskytnutí informací o HW platformě používané při práci na tomto projektu a pomoc s testováním v kameře.

Zadání diplomové práce/2767/2009/xsirok03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2009/2010

Zadání diplomové práce

Řešitel: **Široký Vít, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Implementace algoritmů zpracování obrazového rastru v FPGA**
Implementation of Raster Processing Algorithms in FPGA
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte literaturu se zaměřením na reprezentaci rastrového obrazu, operace nad rastrovým obrazem, využití takových operací v počítačové grafice a v počítačovém vidění, struktura a programování FPGA a DSP.
2. Vyberte vhodnou množinu operací a vhodnou platformu obsahující FPGA a DSP pro implementaci těchto operací.
3. Implementujte vybranou množinu operací ve vhodném jazyce (VHDS, Handle C, apod.).
4. Simulujte vybrané operace a demonstrujte výsledky.
5. Diskutujte dosažené výsledky a možnost pokračování práce.

Literatura:

- dle konzultace s vedoucím

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1-2 zadání

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zemčík Pavel, doc. Dr. Ing.**, UPGM FIT VUT
Konzultant: Fučík Otto, Dr. Ing., UPSY FIT VUT
Datum zadání: 21. září 2009
Datum odevzdání: 26. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2
L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Obsah

Kapitola 1

Úvod

Tato práce se zabývá jedním z mnoha problémů řešitelných v oblasti výpočetní techniky. Jde tu zejména o práci s obrazem, ale trochu v jiném duchu, než jak je známa z běžné praxe. Nepůjde zde ani o retuše fotografií, ani o tvorbu počítačové grafiky. Prestože se bude hovořit o fotografiích, tak zde nejsou chápány jako památka na rodinnou oslavu nebo dovolenou u moře, ale jako zdroj dat pro zpracování v počítači. Taková fotografie nese řadu informací, které jsou pro člověka naprosto zřejmé. Není problém říci, že na fotce je auto vyfocené u moře, vedle auta stojí člověk se zmrzlinou. Výpočentí technika něco takového říci nemůže, chybí ji k tomuto účelu lidské vnímání okolí.

Existuje zde ovšem snaha naučit počítač obraz vyhodnocovat podobně, jako jej vyhodnocuje člověk. V současnosti už existuje celá řada postupů, které toto řeší a podávají slušné výsledky, poměrně dobře známá je detekce obličeje v digitálních fotoaparátech, která slouží k optimálnímu nastavení expozice při focení osob. Tato problematika není jednoduchá, a proto má význam se s ní zabývat a hledat v ní různé cesty přinášející buď zcela nové možnosti nebo vylepšující stávající. V této práci se zabývám možností druhou, jde zde o zlepšení stávajících řešení.

Digitální technika zaznamenala velmi masivní rozvoj v celé řadě oblastí. Jako příklad může posloužit ve spotřební elektronice přechod nejdříve od MC k CD, o něco později od VHS k DVD. Ve fotografii byl také klasický film a jeho vyvolávání odsunuto na okraj zájmu široké veřejnosti a naopak digitální fotografie je dnes pro svou dostupnost něčím naprosto samozřejmým. Digitální televizní vysílání postupně analogové vytlačí.

Pro účely této práce je důležitý digitální obraz. Z běžné uživatelské praxe je známo, že s digitálním obrazem lze s pomocí vhodného softwaru v počítači dále pracovat. Digitální obraz není přitom v počítači nic jiného, než uspořádaná množina čísel. Z toho plyne, že ať už se jedná o různé úpravy barev, ořezávání, dodatečné doostřování nebo cokoliv dalšího, vždy jde obecně o realizaci nějaké výpočetní operace.

Jak bude podrobněji uvedeno v dalších částech, některé operace zpracování obrazu jsou výpočetně velmi složité. Hrubě zjednodušeno lze říci, že čím složitější operace je, tím více času je třeba k jejímu provedení.

Čas je ovšem v některých případech drahý či nedostačující, a všeobecným trendem v oblasti výpočetní techniky je zrychlování výpočtu. Cest zrychlování je několik, jednou z nich je překotný vývoj stále rychlejších počítačů. Tento způsob využívají často například hráči počítačových her. Další možností je vývoj speciálních čipů navržených a optimalizovaných pro určitou aplikaci. Také optimalizace výpočtů přináší v této oblasti posun vpřed.

Tato práce se zabývá další variantou, a to možností zpracování digitálního obrazu s využitím zrychlení výpočtů pomocí hardwarového předzpracování. Principem je výpočet části

operace ještě před započetím výpočtu v softwaru. Software pak využije předpočítané mezivýsledky. Tím bude zaručeno, že výpočet v softwaru bude jednodušší a snad i rychlejší.

Cílem práce je tedy nalezení a implementace vhodného způsobu předzpracování obrazu v hardwaru. K tomu účelu poslouží FPGA.

V následující části práce, v kapitole 2, je popis v současnosti známých faktů týkajících se rastrového obrazu, programování aplikací v HW a SW. Popisovány jsou vybrané operace s ohledem na počítačovou grafiku a počítačové vidění. Pro obě (SW i HW) části platí, že vzhledem k rozsahu a množství možných informací popisuji pouze problematiku nějakým způsobem související s touto prací, ať už proto, že jsem se s ní musel seznámit, abych získal patřičný přehled v oblasti, nebo proto, že jsem zmiňované věci skutečně využil. Pojednává se zde zejména o FPGA a DSP. Část práce 3 diskutuje získané poznatky a na základě vzniklých závěrů specifikuje konkrétní představy o realizaci. Dále je představeno několik variant HW platforem a zvážena jejich vhodnost použití pro účely této práce. Je zde brán ohled na dostupnost platformy a také na obtížnost práce s danou platformou. Kapitola 4 postupně popisuje všechny zásadní etapy vývoje projektu. Podrobně jsou popsány veškeré experimentální implementace a jejich výsledky. Samozřejmě nechybí jejich kompletní vyhodnocení. V kapitole 5, závěru , následuje shrnutí dosažených výsledků a popis směru dalšího vývoje.

Kapitola 2

Současný stav

V této části práce se zabývám shrnutím známých a pro tuto práci potřebných faktů týkajících se rastrového obrazu a programování aplikací v HW a SW. Na začátek uvádím základní informace o digitálním obrazu, jak ho získat, jak vypadá, co se s ním dá dělat včetně příkladu. Následuje popis, jak se dá s obrazem pracovat v kontextu použité techniky sloužící k práci s obrazem.

2.1 Digitální obraz

V současnosti je digitální obraz něčím naprosto běžným a snad každý se se ním setkal. Každý obraz ale musí nějak vzniknout.

Snímání digitálního obrazu

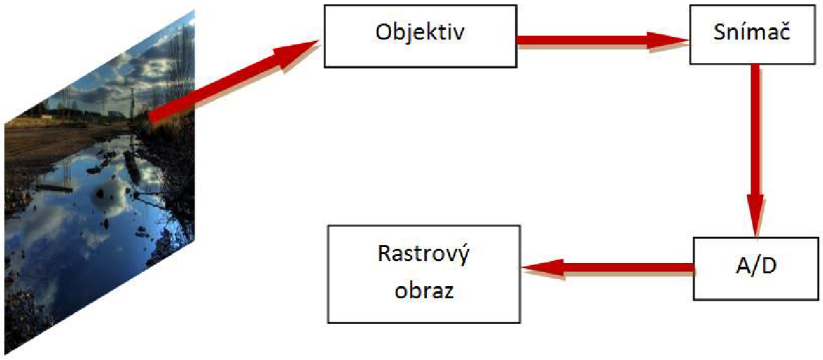
Pro snímnání digitálního obrazu je potřeba digitální fotoaparát nebo kamera. Díky mobilním telefonům, které jsou dnes v drtivé většině vybaveny malou kamerou, má dnes možnost získat digitální obraz téměř každý. Samozřejmě, nelze očekávat vysokou kvalitu takových zařízení. Snímače jsou v nich miniaturní a objektivy (může-li o nich být v případě mobilů vůbec řeč) také.

Použil jsem pojmy objektiv a snímač. Jedná se o dvě důležité součásti digitální kamery nebo fotoaparátu. Prostřednictvím objektivu se zaznamenávaný obraz dostane správně zaostřený, a se správnou clonou na snímač. Snímač může být technologie CCD nebo CMOS. Úkolem snímače je převést vizuální hodnotu na elektrickou, kterou následně převodník AD (analog->digital) navzorkuje. Tím je digitální obraz vytvořen. V praxi je ještě nutné data vhodně uložit, filtrovat a zkomprimovat. Proces je naznačen na obrázku 2.1.

Rastrový obraz

Pro popis rastrového obrazu bude vhodné povšimnout si samotného názvu, kdy pojem rastrový obraz v sobě nese dvě slova: rastr a obraz. Nejprve je třeba uvážit, co to je vlastně obraz. Zde je uveden pohled na věc vycházející z [7] .

O obrazu lze hovořit třeba v souvislosti s uměleckým dílem na plátně. Další možností je fotografie v albu, nebo třeba to, co lze sledovat na obrazovce televize či PC. Jistě by bylo možné nalézt řadu dalších příkladů. Všechny mají společného jmenovatele, kterým je zachycení nějaké více či méně reálné scény.



Obrázek 2.1: Blokové schéma digitalizace

Z formálního hlediska lze obraz považovat za funkci dvou proměnných.

$$o = f(x, y) \tag{2.1}$$

Proměnné x a y jsou rozměry obrazu. Je ovšem třeba počítat s tím, že obraz je digitální, tedy nespojitý, a tak definičním oborem obrazové funkce nejsou čísla reálná. Dále musí být definiční obor omezený, protože není možné zobrazit obraz záporných rozměrů a také nelze pracovat s obrazem nekonečným nebo jen příliš velkým.

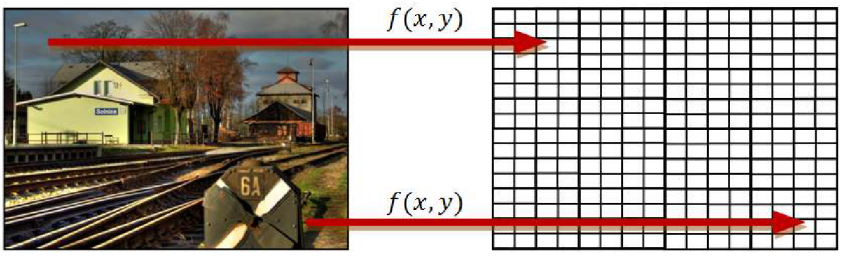
$$f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0 \tag{2.2}$$

Nyní už je specifikováno, co je v kontextu této práce považováno za obraz, a díky upřesnění formálního zápisu rovnicí 2.2 je rovněž popsán rastr.

Rastr je matice bodů o rozměrech x a y, kde x představuje šířku obrazu a y je výška obrazu. Každý bod rastru se nazývá pixel. Rovnice 2.2 odpovídá obrazu monochromatickému, v případě obrazu barevného může být hodnota pixelu obvykle vyjádřena jako trojice barevných složek RGB a tedy zobrazení f potom odpovídá rovnici 2.3 .

$$f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \tag{2.3}$$

Princip rastrového obrazu je schematicky znázorněn na obrázku 2.2.

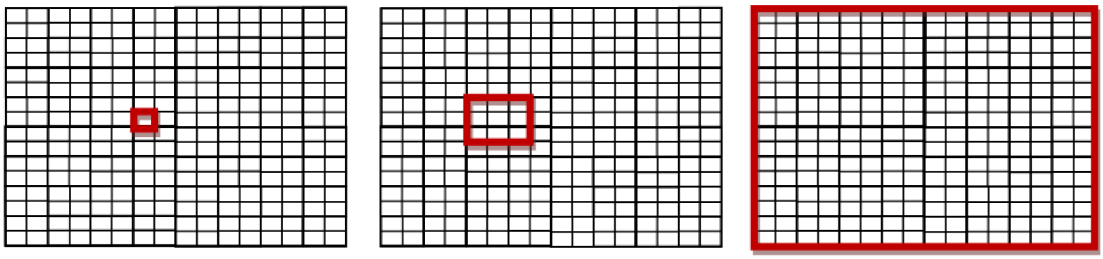


Obrázek 2.2: Rastrový obraz

2.2 Operace nad rastrovým obrazem

Operace nad rastrovým obrazem lze dělit z hlediska množství zpracovávaných pixelů na bodové, blokové a globální.

Bodová operace pracuje na vstupu s diskrétní hodnotou jednoho pixelu. Bodovou operací může být třeba prahování (viz dále), kdy se určí jasová hodnota pixelu a v porovnání s prahovou hodnotou se buď prohlásí za popředí, nebo pozadí. Okolní body přitom nemají na nic vliv. Bloková operace pracuje se vstupním blokem několika sousedních pixelů. Představitelem je třeba mediánový filtr (hodnota pixelu se vypočítá jako medián bloku dat okolo pixelu. Operace globální pracuje s celým obrazem, tj. se všemi pixely obrazu, například výpočet průměrné intenzity obrazu. Nákres problematiky ukazuje obrázek 2.3.



Obrázek 2.3: Zleva: bodová, bloková a globální operace

Z pohledu použití operace lze dělit operace do dalších kategorií, je možné uvést operace transformační pro různé rotace, zkosení apod., operace pro skládání obrazů (např. panoramatické snímky), filtrační operace (odstranění šumu, detekce hran...) nebo také segmentační operace.

2.3 Segmentace

Slovník cizích slov říká, že slovo segment znamená část nebo díl. Segmentace je potom proces rozdělování na části nebo díly.

Segmentace v oblasti počítačového vidění je chápána jako oddělení objektů zájmů v obraze od pozadí [6] [4]. To je naprostý základ, analýzy obrazu z hlediska jeho obsahové stránky. Jak jinak by software mohl například přečíst text v obraze, když neví, kde ten text v obraze je?

Segmentace prahováním [6] [4]

Segmentace prahováním je segmentační metoda vhodná v případě, že hledáme objekty podobné barvy, na kontrastním pozadí. Metoda je to velmi jednoduchá, možná nejjednodušší ze všech.

Mějme hodnotu T, a říkejme jí práh (z anglického Threshold). Dále mějme funkci $f(x, y)$, jejíž výsledek porávnáme s prahem. Funkce $g(x, y)$ nabývá pouze dvou hodnot v závislosti na porovnání $f(x, y)$ s T.

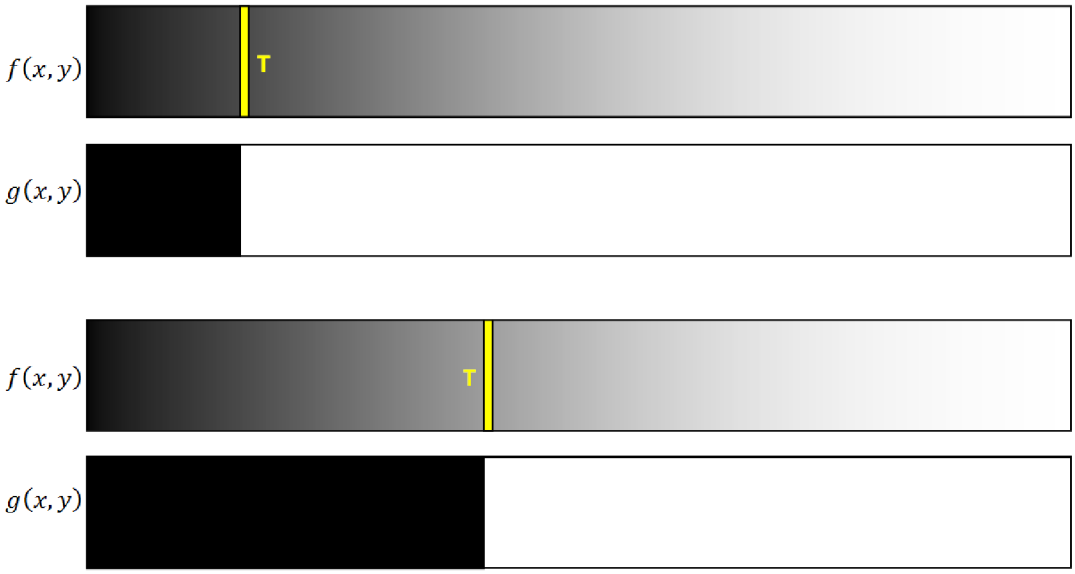
$$g(x, y) = Max \wedge f(x, y) \geq T \tag{2.4}$$

$$g(x, y) = Min \wedge f(x, y) < T \tag{2.5}$$

V případě šedotónového obrazu platí, že $f(x,y) \in \mathbb{N}_0 \wedge 0 \leq f(x,y) \leq 255$. Pro práh T platí $T \in \mathbb{N}_0 \wedge 0 \leq T \leq 255$.

Tedy práh T je nějaká jasová hodnota, s kterou se porovnává jasová hodnota zkoumaného pixelu $f(x,y)$. Dle rovnic 2.4 a 2.5 se nastaví nová obrazová hodnota $g(x,y)$. Min a Max jsou hodnoty minimálního jasu 0 a maximálního jasu 255. Existují i další modifikace prahování. Cílem může být pouze potlačení pozadí a zachování vzhledu objektů, v takovém případě hovoříme o semi-thresholdingu. Jiná verze, pásmové prahování může stanovit toleranci hodnot okolo prahu. Také je možné stanovit prahů i výstupních hodnot několik.

Názorná ilustrace funkce prahování je na obrázku 2.4.



Obrázek 2.4: Princip prahování

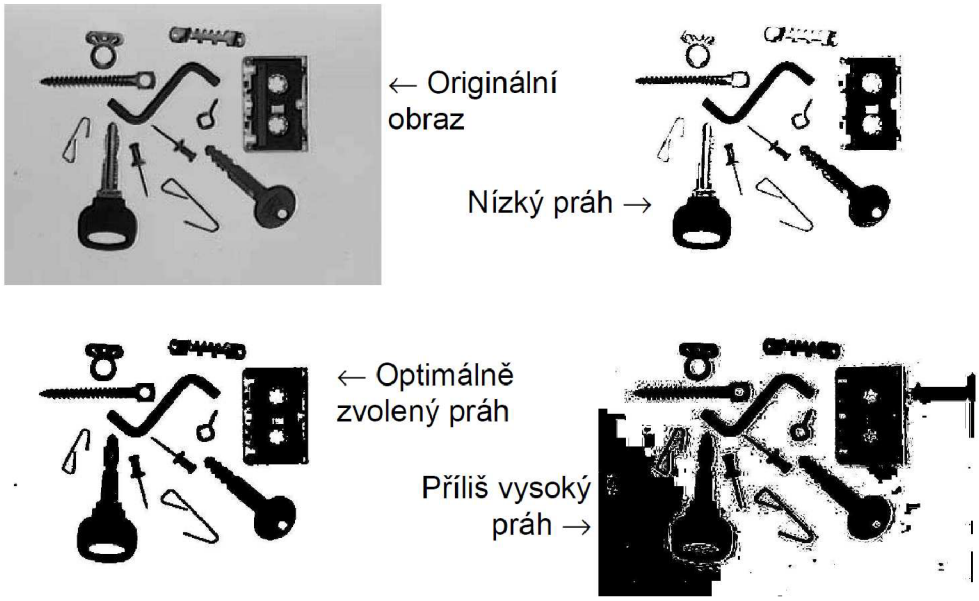
Práh

Z obrázku 2.4 i z uvedených vztahů je zřejmé, že hodnota $g(x,y)$ je závislá na nastavení prahu T. Z toho plyne problém s nastavením prahu T. Když se nastaví práh správně, výsledek prahování bude pravděpodobně takový, jaký je očekáván. Při nesprávném nastavení prahu bude ale výsledek špatný. Ilustruje to obrázek 2.5 převzatý z přednášky [6].

Jak je vidět na obrázku 2.5, je velmi důležité nastavit správný práh. Uživatel to svede metodou "kouknu a vidím"bez větších potíží, případně metodou "pokus-omyl"práh vyladí. Jak to ale dokázat automatizovaně v nějakém segmentačním procesu ?

Možnosti automatického optimálního nastavení prahu jsou dvě. Lze analyzovat histogram nebo použít metodu statistickou

Další problém přináší scéna nerovnoměrně osvětlená. Bude-li snímaný objekt nerovnoměrně osvětlen, bude se optimální práh lišit pro světlejší i pro tmavší části snímku. Obrázek 2.6 prevzatý z [6] krásně ukazuje problém nehomogenního osvětlení scény. Práh je vhodně nastavený přibližně pro dvě třetiny obrazu, ve spodní třetině už je segmenatace špatná. Zároveň je úplně vpravo vidět, že s více prahy se segmenty detekují lépe. Na druhou stranu jsou ale některé segmenty detekovány dvojitě.



Obrázek 2.5: Vliv nastavení prahu na výsledky prahování, převzato z [4]

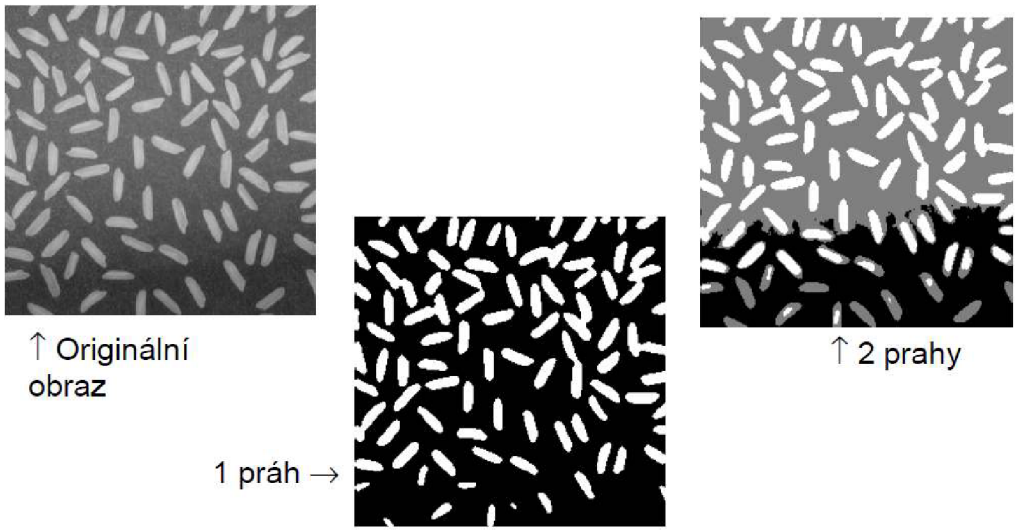
Řešením může adpativní prahování. Narozdíl od obyčejného prahování, kdy se stanová práh globálně pro celý obraz, se při adaptivním prahování stanoví práh pro bloky obrazu. Obraz se rozdělí na vhodně velké bloky (např. 8×8), pro které se stanoví automaticky optimální práh některým z výše uvedených způsobů. Poté se při segmentaci používá práh odpovídající danému bloku.

2.4 Technické způsoby zpracování

Klasickým přístupem k vykonávání obecně libovolného algoritmu je jeho naprogramování na nějakém univerzálním mikroprocesoru (tím může být třeba PC a pro názornost bude v další části textu jako příklad používáno) nebo vytvoření speciálního hardwarového řešení. Oba přístupy mají své klady i zápory.

Významnou výhodou tvorby softwaru na PC je jeho dostupnost a univerzálnost stroje. Znamená to, že PC může řešit libovolnou algoritmizovatelnou úlohu. Tato výhoda je ovšem zároveň nevýhodou. Jak už to bývá u většiny univerzálních věcí, musí nutně docházet ke kompromisům. Software napsaný pro PC je tak bohužel zpravidla dost pomalý. Někdy to nevadí, protože některé operace jsou jednoduché a PC je zvládá takovou rychlostí, že to uživateli vyhovuje a lepší nepotřebuje. Jako příklad poslouží třeba grafické editory, kde se v reálném čase nechá měnit barevné podání, jas a jiné vlastnosti obrazu. Opačným příkladem může být třeba tvorba HDR obrazu, nebo detekce objektů v obraze. Takové operace mohou trvat podstatně déle.

Naproti tomu řešení algoritmů přímo v hardwaru bývá velmi rychlé. Je tomu tak proto, že hardware pro nějakou specifickou činnost, také nazývaný ASIC (Application Specific Integrated Circuit), je speciálně navržen a optimalizován. Nevýhodou je ale velmi vysoká cena speciálního HW. Ta je ovlivněna obvykle malou (v porovnání s univerzálním procesory v PC) sérií vyrobených kusů. Cena ASIC dosahuje několika milionů dolarů [3]. Problém může také nastat v případě, že se do systému zanese chyba. Narozdíl od SW se nedá prostě přepsat knihovna a poslat uživatelům update. Celý návrh se musí i s výrobou přepracovat.



Obrázek 2.6: Problém nehomogenního osvětlení, převzato z [6]

Totéž platí v případě dodatečného přidání funkčnosti.

Existuje ještě třetí možnost, a tou je využití FPGA. Co je to FPGA a jak to funguje, bude vysvětleno dále. Zde stačí poznamenat, že se jedná o programovatelný hardware, který lze upravovat pro potřeby aplikace. Návrhář systému si může zvolit zapojení logických obvodů uvnitř FPGA přesně podle konkrétní aplikace a sám si jej realizovat. Dochází tak ke spojení výhod hardwarového řešení ASIC a univerzálních mikroprocesorů.

2.5 Programovatelný hardware - FPGA

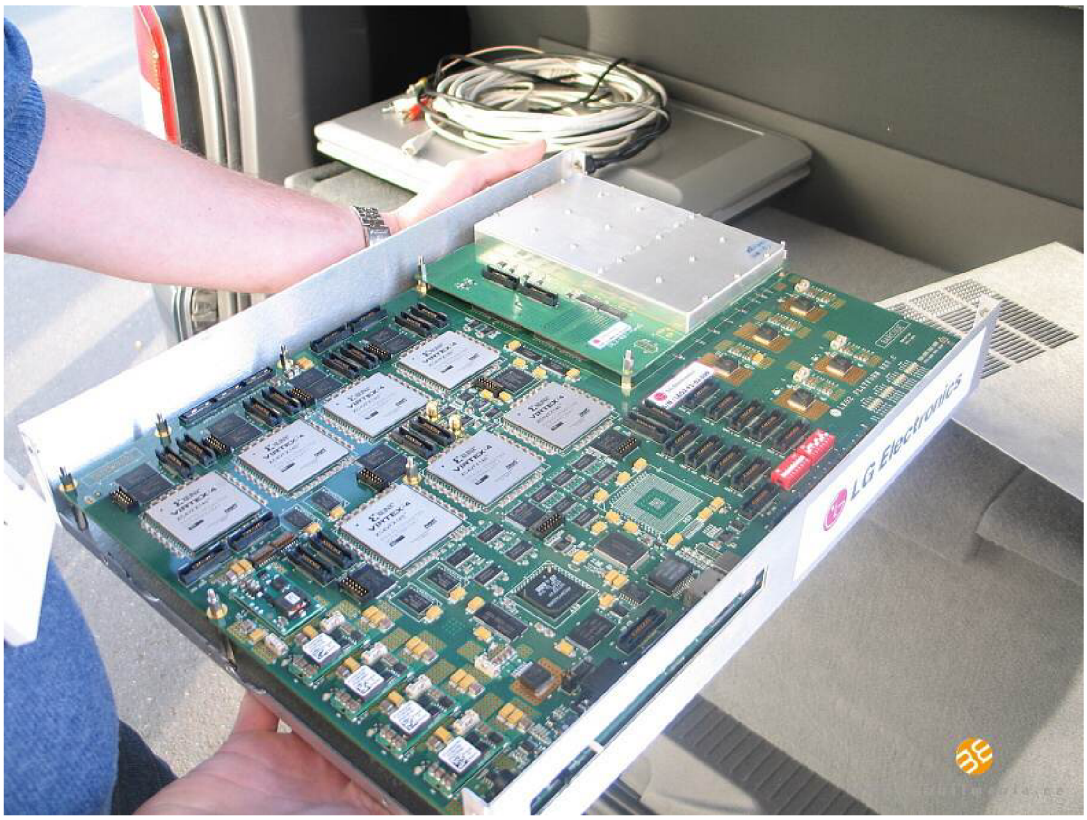
Problematika FPGA je poměrně složitá a existuje o ní celá řada inforamčních zdrojů, následující text jsem zpracoval hlavně s využitím [3], kde je vše poměrně hezky uspořádáno.

FPGA (Field-programable gate array) je programovetelné hradlové pole. FPGA kombinuje výhody softwarového řešení na univerzálním mikroprocesoru s rychlostí hardwarových řešení. V FPGA lze implementovat obvody podobně jako v ASIC, ale odpadá drahý návrh a výroba. To zajišťuje vysoký výkon aplikací v porovnání s čistě softwarovým řešením, ale zároveň snižuje jejich cenu v porovnání s ASIC. V porovnání s ASIC je rychlost nižší, ale jen přibližně $5\times \div 25\times$, zatímco cena se liší v několika řádech. Také doba vývoje je nesrovnatelně kratší [3].

FPGA se tak dobře hodí jak k tvorbě prototypů, tak k realizaci celé řady aplikací, kde výkonnost FPGA vyhovuje i ve finálním provozu. Názorný příklad prototypového užití FPGA je na obrázku 2.7. Jedná se o prototypový mobilní telefon firmy LG sloužící k testování LTE technologie [2].

Propojení FPGA

Návrh aplikace v FPGA však klade poměrně vysoké nároky na znalosti a schopnosti jeho autora. Při návrhu je potřeba vymyslet jak algoritmus, tak zapojení logické hardwarové funkce. Jinými slovy, je potřeba vymyslet nebo najít algoritmus pro řešený problém a tento algoritmus implementovat jako hardwarové zapojení.



Obrázek 2.7: Prototypový mobilní telefon firmy LG sloužící k testování LTE technologie, převzato z [2]

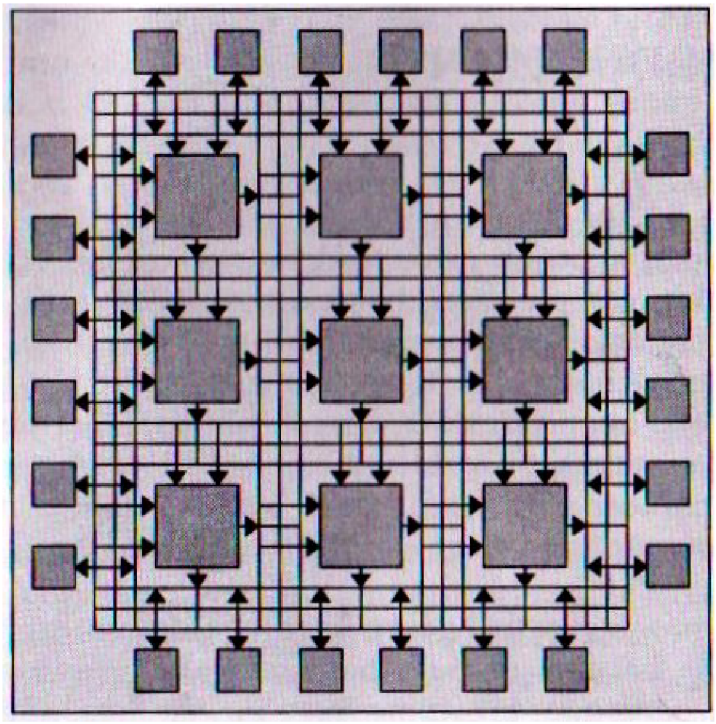
Nyní je na místě stručně popsat, jak FPGA funguje. Obrázek 2.8 ukazuje jak je FPGA uvnitř tvořeno. Jsou to logické bloky ve spojové struktuře. Nebo, chceme-li, pole (angl. array) hradel (angl. Gate), a to jsou právě písmena ze zkratky FPGA. Logické bloky se propojují pomocí spojové struktury tak, aby vytvořily požadovaný hardware. Popisu spojení se říká design.

Propojení FPGA se popisuje k tomu určenými programovacími jazyky (tzv. HDL - hardware description language), neznámější z nich jsou Verilog a VHDL. Práce na designu (obrázek 2.9) vypadá tak, že se pomocí některého z HDL jazyků popíše funkce obvodu ve formě zdrojového kódu. Ten se syntetizuje (Logic Synthesis) na síť hradel (nezávislých na FPGA). Až tento výsledek se převede na hradla dle konkrétní technologie (Technology mapping). Následuje rozmisťování v FPGA (Placement) a spojování (Routing). Na závěr je vygenerován kód, který se nahraje do FPGA.

Vygenerovaná konfigurace se může pohodlně uložit například do paměti typu flash (micro SD karta apod) a z ní se FPGA konfiguruje. Při případné změně systému stačí vyměnit flash paměť a restartovat systém. Načte se nová konfigurace a systém může pracovat jinak.

Architektura

Také zde vychází text z [3]. V FPGA jsou v podstatě nejdůležitější jen dvě věci - logické bloky, které počítají, a spoje, které umožní přesun výsledku jednoho bloku na vstup dalšího bloku.



Obrázek 2.8: Nákres struktury FPGA, převzato z [3]

Logické bloky

Je známo, že výpočty lze popsat Booleovskou funkcí. Dále je známo, že každou Booleovskou funkci lze popsat pravdivostní tabulkou. Každou složitější funkci lze složit z jednodušších. Také podmínky IF lze popsat logickou funkcí. Pak lze celý algoritmus popsat logickou funkcí. A jak už bylo uvedeno, logickou funkci lze popsat pravdivostní tabulkou.

Právě pravdivostní tabulky jsou srdcem FPGA. Pravdivostní tabulky jsou implementovány v jednoduchých prvcích, kterým se říká lookup table, nebo též LUT (dále jen LUT).

Z obvodového hlediska není LUT nic jiného, než výběr jedné z N, tedy multiplexor. Optimální velikost LUT se jeví v současné době jako 4-LUT, tedy 4 řídící vstupy.

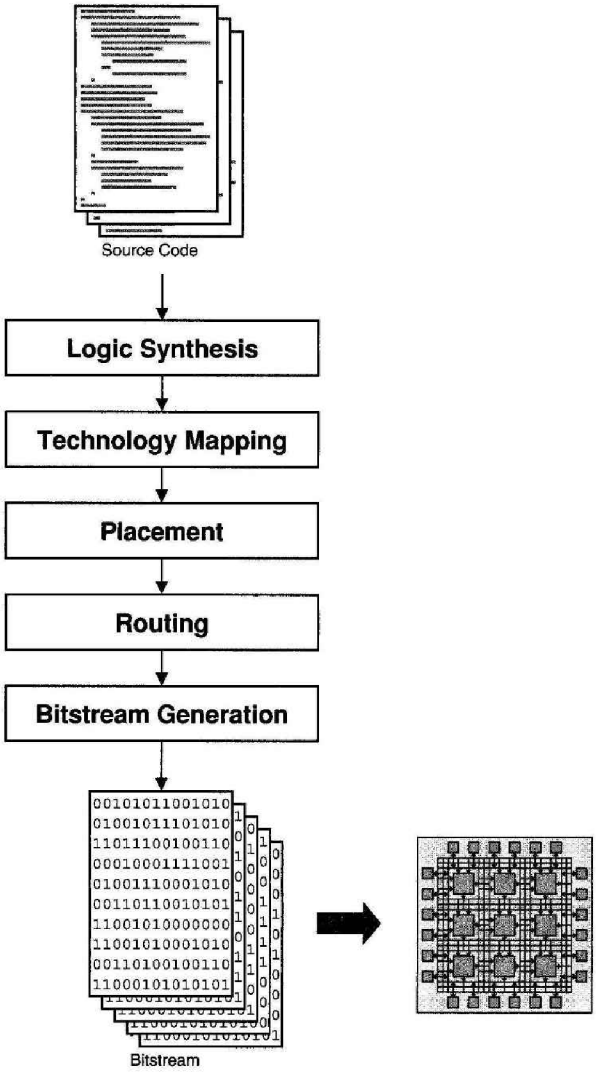
Další důležitý prvek je klopný obvod typu D, což je jednoduchá jednobitová pamět. Klopný obvod D a LUT tvoří logický blok.

Shrnutí

FPGA tak umožňuje návrháři sestavit libovolnou funkci vhodným propojením logických bloků Současně ale může být takových oddělených funkcí v FPGA více (až do plného obsazení čipu) a tyto funkce mohou běžet paralelně. To je zcela odlišné oproti klasickému sekvenčnímu postupu v mikroprocesoru, při správném návrhu tak bude systém postavený v FPGA velmi výkonný.

2.6 DSP

Zkratka DSP má v oblasti práce s digitálním obrazem dva významy. Prvním je Digital Signal Processing, což je souhrný název pro zpracování digitálních signálů. Druhá vysvětlení



Obrázek 2.9: Proces tvorby designu FPGA , převzato z [3]

zkratky je Digital Signal Processor. Zatímco "processing"označuje nějaké dění s digitálními signály, pak "processor"je nástroj, který toto dění umožní.

V této části jsem velké množství informací čerpal z publikace Stevena W. Smitha "The Scientist and Engineer's Guide to Digital Signal Processing"[8], samozřejmě i zde jako v případě FPGA existuje řada dalších zdrojů, zejména pak různé články online. Nejlépe uspořádaný přehled o problematice jsem ale našel ve zmiňované publikaci [8] a podle ní jsem zpracoval dále uvedený souhrn.

Při pohledu na DSP je třeba nejprve pohlédnou na rozdíly, které jsou mezi běžnými aplikacemi, které se zabývají prací s textovými daty a mezi aplikacemi, které řeší zpracování digitálního obrazu.

Běžné apliakce na počítačích pracují s nějakými daty, někam je přesouvají, něco v nich hledají. Oproti tomu zpracování obrazu je zcela jiné, jsou to v první řadě matemetické operace. Obě tyto věci běžné PC zvládá. Ale není dost dobře možné, aby byl procesor v PC optimalizován tak, aby byl ve všech oblastech výpočtu nejlepší, tedy nejrychlejší. Proto

vznikly speciálně optimalizované mikroprocesory určené pro co nejrychlejší práci s digitálním signály, tedy DSP.

Nabízí se otázka, k čemu je dobré, aby DSP pracovaly rychle. Pro tyto účely si půjčím příklad z [8]. Mějmě nějaký rozsáhlý textový dokument a proveďme s ním něco složitějšího, třeba konverzi formátu. Operace se spustí, a za chvíli je k dispozici výsledek. Je přitom obvykle celkem nepodstné, jestli to trvalo sekundu, nebo její zlomek. Maximálně si uživatel chvíli počká. A jak je to s digitálním signálem? Ten, podobně jako signál analogový, má v čase nějaký průběh a mění se. Představme si třeba videokameru, tam je obraz proměnnlivý v důsledku pohybu objektů v záběru kamery i vlivem pohybů kameramana. Každým okamžikem jsou tak k dispozici nová data, jiný signál, který musí být zpracován. Čekat proto nelze a čím rychleji data lze zpracovat, tím více se s nimi stihne provést operací nebo také tím více jich systém zpracuje a může tak narůst rozlišení obrazu nebo počet snímků za sekundu.

FIR filtr

Jako příklad typické funkce řešené pomocí DSP uvádí [8] FIR filtr. FIR (Finite input response, tedy filtr s konečnou odezvou na vstupní signál) pracuje tak, jak je naznačeno na obrázku 2.10, rovnicový zápis uvádí rovnice 2.6.

$$y[n] = a_0x[n] + a_1x[n-1] + a_2x[n-2] + a_3x[n-3] + a_4x[n-4] + a_5x[n-5] + ... \quad (2.6)$$

Jde vlastně o konvoluci několika vzorků vstupního signálu $x[n], x[n-1], x[n-2], ...$ s koeficienty jádra filtru $a_0, a_1, a_2, ...$ Důležité přitom je, že při výpočtu je v rovnici 2.6 pouze násobení a sčítání, tedy přesně ty operace, o kterých jsem psal výše. Pro výpočet násobení a sčítání jsou DSP optimalizované.

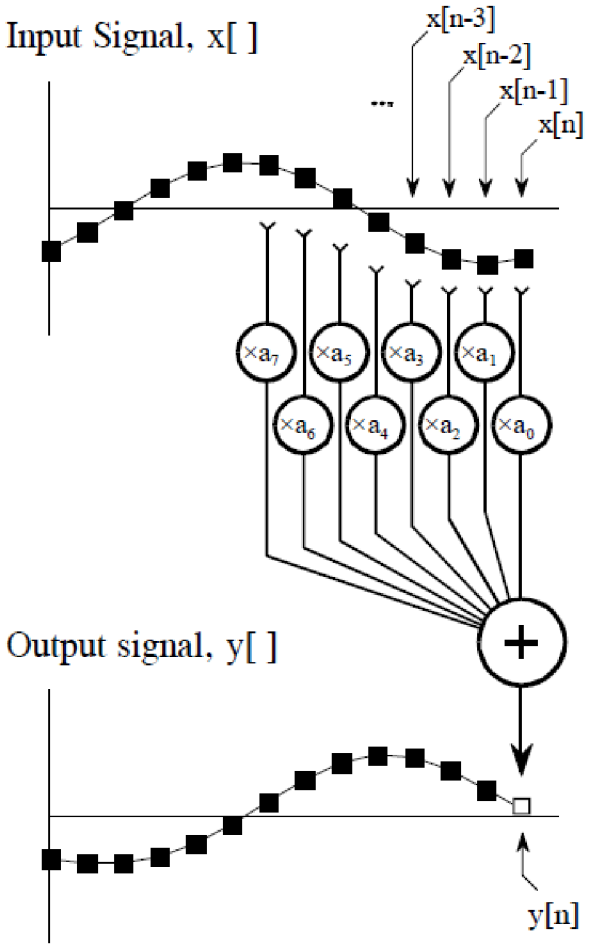
Krom samotného rychlého výpočtu je zapotřebí také optimalizovat přístup k datům. Nebylo by nic platné, kdyby byl procesor seberychlejší, když by musel čekat na data. Steven W. Smith[8] vysvětluje princip optimalizace přístupu k datům na úrovni architektury mikroprocesorů.

Architektura DSP

Architektura mikroprocesoru může být Harvardská, nebo podle Von Neumanna. Von Neumannovská architektura obsahuje jednu paměť, která je společná pro program i pro data. Harvardská architektura má paměti dvě, jednu paměť pro data a druhou paměť pro program.

Von Neumannova architektura je rozšířena v řadě mikroprocesorů, zejména pak v osobních počítačích. Jednu paměť s jedním procesorem spojuje jen jedna datová sběrnice. V jednom okamžiku tak lze přenášet data jen z jednoho umístění. Takže když bude mikroprocesor Von Neumannova typu řešit instrukci sčítání, musí číst data z paměti celkem třikrát. Nejprve instrukci a poté dva operandy. Jestliže každé čtení probíhá v jednom cyklu procesoru, pak na tuto instukci je zapotřebí celkem třech cyklů.

Harvardská architektura má paměťi dvě, pro program a data zvlášť. Při zpracování instrukce se dvěma operandy je ovšem sběrnice paměti programu vytížena méně, než pamět sběrnice dat, na jednu instrukci se musí přečíst dva operandy. Tento stav se dá řešit přesunutím části dat do programové paměti. Například jádro již dříve zmiňovaného FIR flitru, to jsou konstanty, mohou být uloženy v programu.



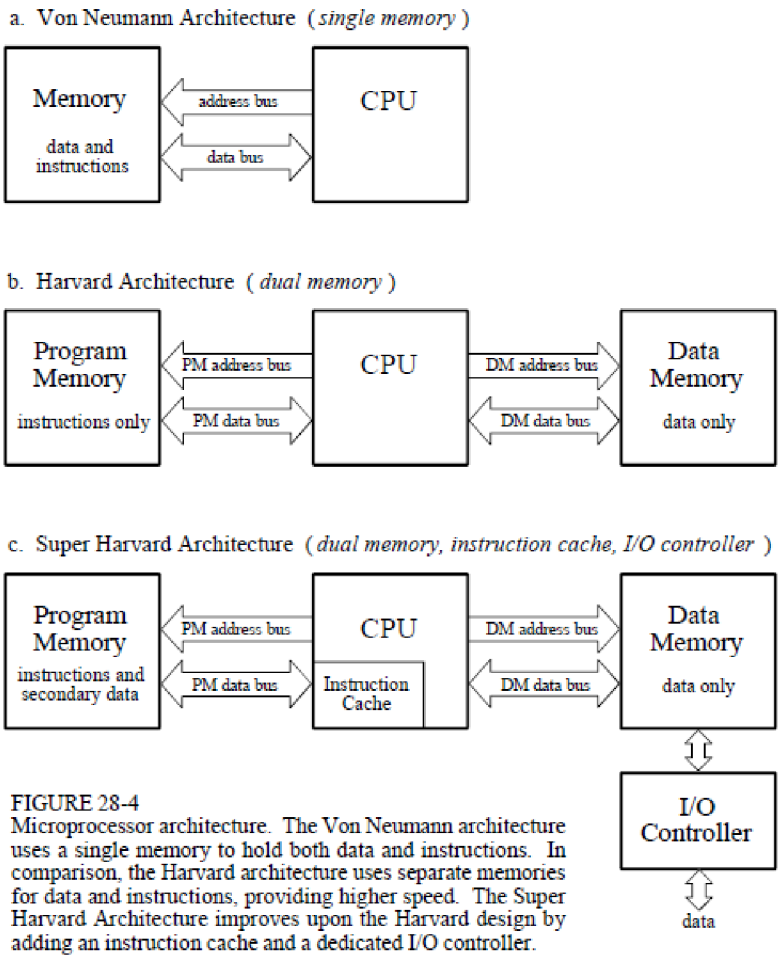
Obrázek 2.10: FIR filtr, převzato z [8]

To už se ale pomalu dostávám k třetímu typu architektury, který je vylepšním architektury Harvardské. Pod názvem Super Harvardská Architektura se skrývá Harvardská architektura rozšířená o vyrovnávací paměť instrukcí (instruction cache) a také o řadič I/O zařízení.

Všechny tři varianty ukazují bloková schémata na obrázku 2.11. Obrázek je tak jako tento příklad převzat z [8].

Přínos vyrovnávací paměti instrukcí se projeví zejména v případě programu se smyčkami, což je ve zpracování signálů poměrně častý jev. Tedy bude se opakovaně vykonávat nějaká posloupnost instrukcí. Budou-li tyto instrukce čteny z vyrovnávací paměti, není potřeba číst je z paměti programu. Tedy v paměti programu jsou koeficienty filtru, jsou tam také instrukce, a to vše je ve vyrovnávací paměti CPU. Po datové sběrnici tak stačí jen číst data (v případě FIR jsou to vzorky) a násobit je daty , která už jsou v CPU. Výhoda je zřejmá, každou takovou instrukci stačí přečíst jedinou hodnotu, což je proti třem čtením u Von Neumanna značný rozdíl. Pochopitelně, první iterace smyčky bude pomalejší, neboť vyrovnávací pamět se musí nejprve naplnit.

Řadič operací I/O neboli Input/Output tedy vstup/výstup je navržen tak, aby dokázal vysokou rychlostí dostat data z vnějšího vstupu do paměti dat a také obráceně z paměti na



Obrázek 2.11: Architektury, převzato z [8]

výstup. Za povšimnutí stojí vynechání CPU z tohoto přenosu, uplatňuje se DMA. Obsaženy jsou paralelní i sériové linky, obě velmi rychlé. [8] uvádí rychlosti kolem 160MB/s. Tyto vlastnosti jsou klíčové pro DSP. Cílem je dostat data do procesoru, provést výpočet a dostat data ven dříve, než přijde další vzorek.

2.7 OpenCV

Pro tvorbu aplikací zpracování rastrového obrazu existuje knihovna OpenCV. Jedná se rozsáhlou volně šiřitelnou knihovnu, která obsahuje přes 500 optimalizovaných algoritmů pro práci s obrazem [5].

S výhodou se dá tato knihovna použít při práci s obrazem, protože řeší načítání i úpravu, tvorbu i ukládání bitmap. Krom toho je v knihovně implementována řada operací z různých kategorií zpracování obrazu - detekce příznaků, objektů, hran, transformace, histogramy a další. Je možné využít i pár funkcí pro GUI.

Do aplikace načtený rastr je potom představen prostým jednorozměrným polem, kde jsou jednotlivé pixely řádek po řádku za sebou. V případě monochromatického obrazu každému pixelu odpovídá jedna položka v poli, představuje jasovou hodnotu, v případě

obrazu barevného je jeden pixel uložen ve třech po sobě jdoucích buňkách v poli, tj. pro každou barevnou složku (červená, zelená, modrá) jeden záznam v poli.

Nejen pro načítání obrazových dat je v OpeCV nadefinována řada užitečných datových struktur a řada funkcí pro práci s nimi, dále funkce pro vykreslování grafických prvků do bitmapy, filtrační algoritmy a řada dalšího. Také jsou obsaženy některé základní funkce pro tvorbu jednoduchého GUI. Vše je samozřejmě zdokumentováno na webu OpenCV [5].

Kapitola 3

Návrh

Tato část se zabývá popisem návrhu a tedy vlastně cíle této práce. Nejprve popisuji myšlenku návrhu a zabývám se některými jejími podstatnými detaily. Následuje popis dostupných systémů, v nichž lze implementaci navrženého vylepšení vyzkoušet. Je zde také popsáno, která z uvedenách platforem se využije a proč, včetně kompletního popisu jejího rozhraní. Nechybí ani popis množiny vhodných operací.

3.1 Hlavní myšlenka projektu

Cílem práce je nalezení a implementace vhodného způsobu předzpracování obrazu v hardwaru. K tomu účelu poslouží FPGA.

Integrální obraz

Myšlenku práce lze formulovat takto:

Uvažujme komplexní systém s FPGA, který zpracovává obraz od jeho zisku kamerou až po nějaké více či méně složité operace nad získaným obrazem v počítači. Nechme FPGA počítat nějaké vhodné operace pro menší či větší bloky obrazu. Výsledky těchto operací přidejme k obrazu ve formě několika málo sloupců vpravo či řádků dole v přenášené bitmapě.

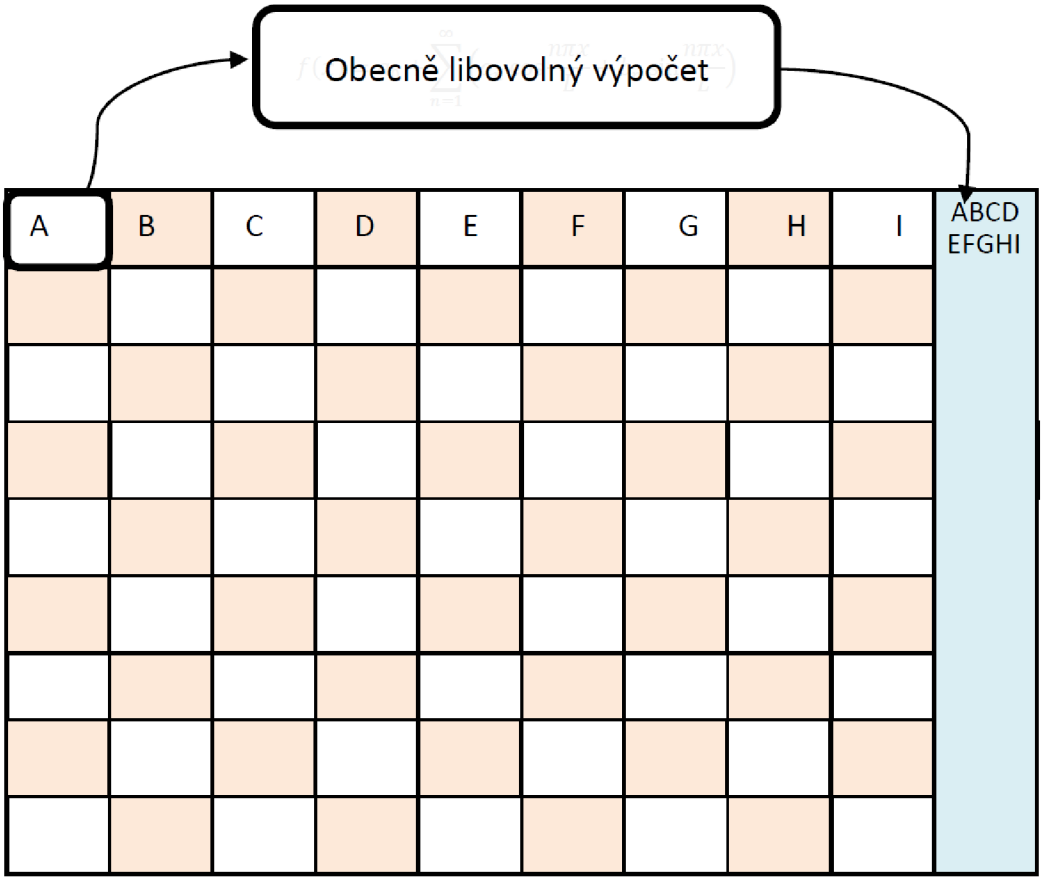
Princip myšlenky ilustruje nákres na obrázku 3.1.

Pro takový obraz se samozřejmě musí přizpůsobit algoritmus v SW části provádějící s obrazem operace tak, aby využíval informace vypočtené při předzpracování v HW a umístěné na stanoveném místě v rastrovém obrazu.

Metoda ovšem přináší i problém, a tím je nestandardní rozměr zvětšeného obrazu, ať už by se jednalo o řádky navíc dole, nebo o sloupce na kraji. Tím by se pochopitelně změnily rozměry obrazu v systému a tak by se muselo přizpůsobit pracně mnoho věcí v již existujícím systému. Pro zkušební provoz a testování v rámci této práce se proto použije úprava, kdy hodnoty na okraji obrazu budou přepsány hodnotami získanými výpočtem, takže rozměr obrazu zůstane zachovaný.

Takto bude dosaženo stále stejných rozměrů přenášeného obrazu a přesto přeneseme předzpracované dodané informace. Bude to ovšem za cenu ztráty malého množství obrazových dat. Z hlediska kompozice obrazu je ovšem třeba uvážit, že okraje záběru stejně obvykle žádnou důležitou informaci nenesou a objekt zájmu je někde blíže ke středu snímku. Pak lze považovat ztracenou informaci za zanedbatelnou.

Takový systém může být postaven více způsoby, v dalším textu popisuji některé možnosti na konkrétních příkladech, včetně platformy, která je pro projekt k dispozici. Tou také ve



Obrázek 3.1: Princip integrálního obrazu

výčtu začínám, následuje čistě DSP varianta a varianta FPGA optimalizovaného vybaveného pro DSP operace

Dostupná platforma

K dispozici pro tento projekt je kamera CAMEA UnicamD2 [1]. Jedná se o zařízení vytvořené pro použití v dopravní telematice (např. při kontrole dodržování nejvyšší povolené rychlosti motorových vozidel nebo při kontrolách jízdy na červenou apod.). Její provedení je vyobrazeno na obrázku 3.2.

Kamera snímá monochromatický obraz o rozlišení 1392×1040 bodů prostřednictvím CCD snímače. Pomocí dvanáctibitového A/D převodníku se obraz digitalizuje. Následuje zpracování uvnitř kamery v FPGA, do okolního světa je obraz vysílán po gigabitovém ethernetu.

Celé zařízení popisuje blokové schéma na obrázku 3.3. Veškeré zpracování obrazu v kameře obstarává systém v FPGA Xilinx Spartan-3. Řízení FPGA, zavádění designu FPGA z flash paměti (micro SD karta), konfiguraci převodníku, komunikaci s periferiemi (např. motor pro zoom) obstarává MCU z řady MSP 430. Design celého systému nebude v této práci popsán, neboť není jejím předmětem.

Pro účely této práce je důležitá zejména přítomnost FPGA v kameře a to, že přes FPGA proudí veškerá snímaná a následně odesílaná data. Zřejmě tedy může FPGA při práci s daty

Obrázek 3.2: Kamera CAMEA UnicamD2, obrázek pochází z dokumentace ke kameře [1]

tato data nějakým způsobem ovlivnit - provést s nimi vybrané operace.

Protože je ale FPGA již v kameře používáno, není možné pro programování využít všechny prostředky FPGA. K dispozici jsou tři BRAM a design se musí vejít pod 40% slices.

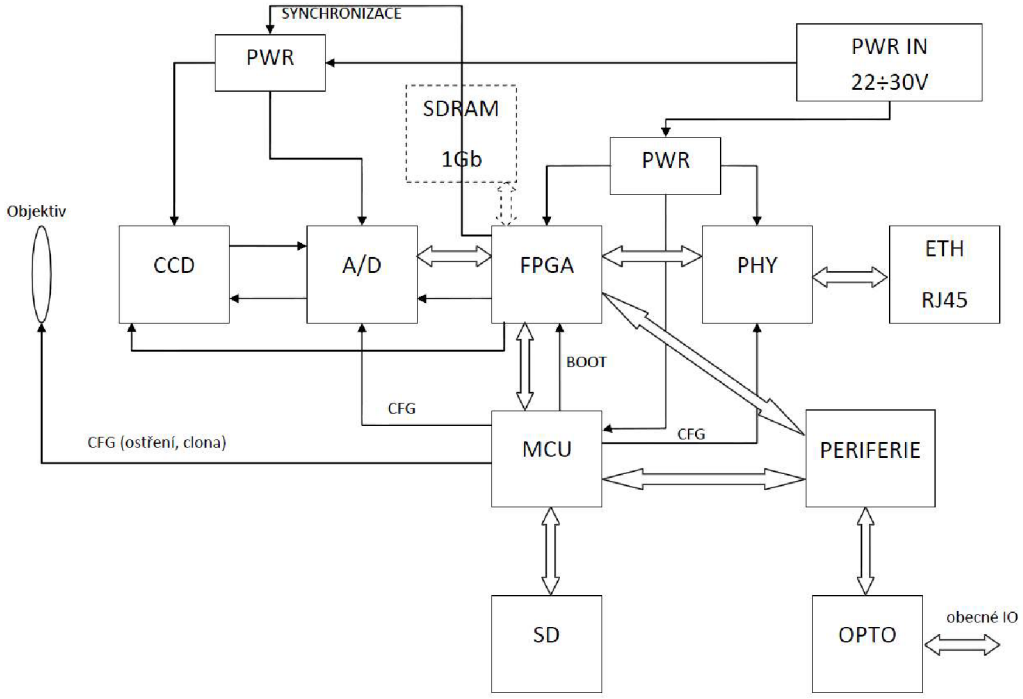
Jiné platformy - DSP

Lze najít i jiná řešení, nejčastěji s DSP. Informace k tomuto řešení jsem čerpal z webu společnosti Texas Instruments, konkrétně z článku Programmable DSP Platform for Digital Still Cameras [11] a také z popisu řešení apliakce Digital Still Camera [10]. Důvodem, proč jsem zvolil zrovna fotoaparát, je názornost - je jednodušší než kamera, ale zároveň ukazuje dobře princip.

Takové zařízení se skládá z několika základních prvků. Musí obsahovat senzor, nebo chceme-li snímač, na který se s využitím vhodné optiky usměrní světlo. Je přitom v kontextu této práce celkem jedno, jestli se jedná o snímač CMOS nebo CCD, pouze v případě CCD je potřeba ještě A/D převodník. Nasnímaná digitální data se musí nějak zpracovat a někam je uložit, provést nějaké úpravy, zkomprimovat, zobrazit, případně odeslat.

Blokové schéma řešení společnosti Texas Instruments 3.4 ukazuje možnou architekturu. Pro účely programování datových operací je nejdůležitější modrý blok Embeded Processor. V tomto případě může jít o DSP z řady TMS320DM3x (Texas Instruments).

DSP má na starosti celý proces zpracování obrazu. Proces má řadu kroků, od samotného nastavení optiky, přes doostřování a podobné úpravy až po kompresi a ukládání. Je tedy nutné veškerou funkčnost naprogramovat přímo v DSP, což může být nesnadný úkol. Jak napoví datasheet [12], architektura DSP bývá dost složitá, z čehož plyne i náročné programování systému. Zejména potom přidání nějaké funkčnosti do již hotového systému bude dost komplikovaný úkol.



Obrázek 3.3: Blokové schéma CAMEA UnicamD2 [1]

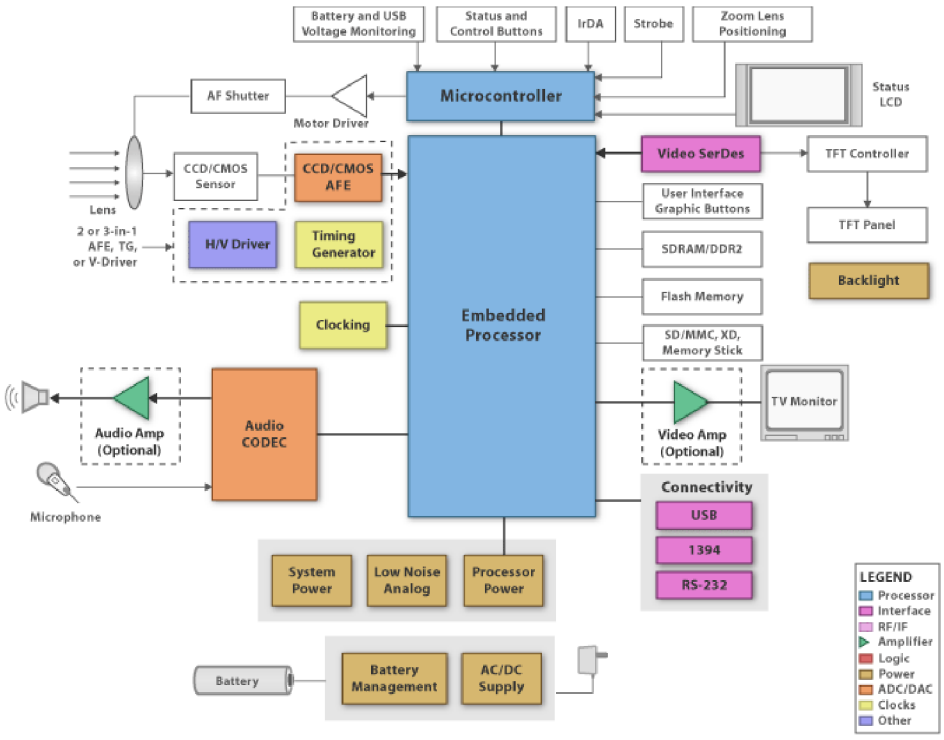
Jiné platformy - DSP FPGA

Další zajímavé řešení představuje Xilinx XtremeDSP Video Starter Kit - Spartan-3A DSP Edition [13]. Jedná se o vývojový kit obsahující osazenou desku, kameru, potřebný software a kabely. Jádrem řešení je čip z řady Spartan-3A DSP, což je FPGA čip vyvinutý pro vysoce výkonné aplikace z oblasti zpracování digitálních signálů. Proti standardním FPGA Spartan-3A má DSP verze vylepšenou architekturu, Xilinx uvádí, že verze DSP je jejím vylepšením a rozšířením. Obsahuje speciální XtremeDSP slices a více BRAM optimalizovaných pro rychlejší práci. Z mého pohledu se po seznámení se specifikací jedná o velmi dobrý produkt, s cenou 2695 USD jej však považuji vzhledem k dostupnosti jiné platformy za nevhodný pro účely této práce.

Způsob řešení

V designu kamery CAMEA UnicamD2 existuje rozhraní znázorněné na obrázku 3.5. Osmibitová data se čtou v okamžiku, kdy je zároveň platný obraz a řádek. Signály jsou synchronizovány.

V tomto rozhraní se vždy přečtou data a provede se nad nimi stanovená výpočetní operace. Část dat se přenese v nezměněné podobě na výstup, část se přepíše daty vypočtenými v souladu s principem popsaným obrázkem 3.1. Ilustrace výpočtu pro kus obrazu je na obrázku 3.6. Nad bloky dat o velikost $k \times k$ z obrazu o šíři n se provedou výpočty. Bloky dat zůstanou neměnné, s výjimkou bloků od pozice $n-1-n/k$, kam se budou zapisovat výsledky blokových výpočtů. Výsledků bude zřejmě n/k, po optimalizacích se počet pravděpodobně změní. K diskuzi zatím zůstává otevřena otázka, co s daty, která se nebudou přepisovat výsledky (na obrázku 3.6 jsou to data na řádcích nula až šest).



Obrázek 3.4: Blokové schéma podle Texas Instruments, převzato z [10]

3.2 Vhodné operace

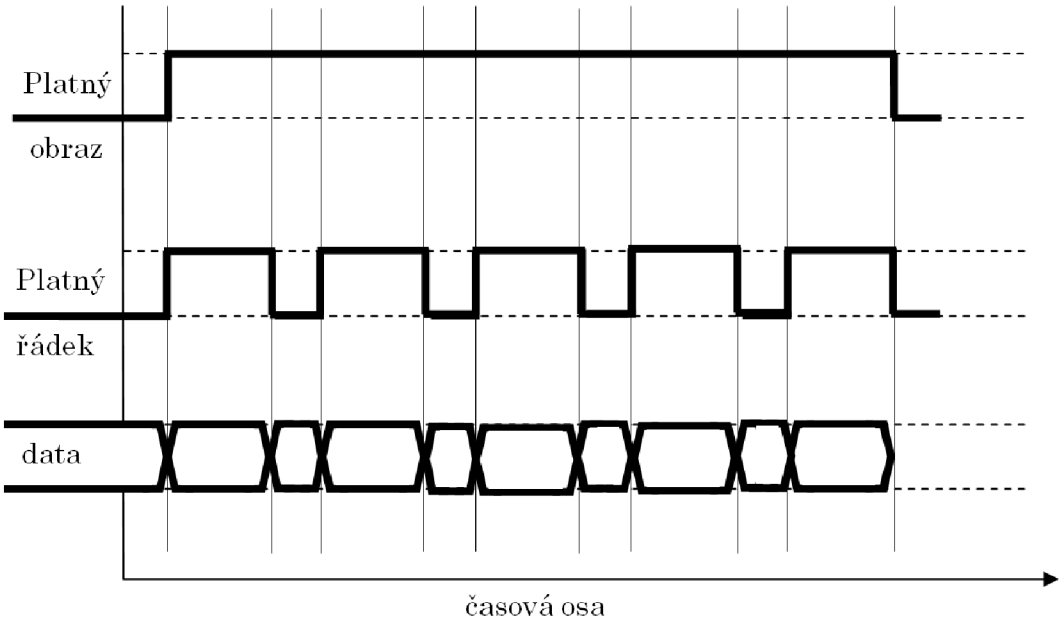
Vzhledem k předchozímu textu v této kapitole je nyní možné specifikovat množinu vhodných operací pro implementaci v FPGA. Bude to množina všech takových operací, pro které současně platí, že jejich vykonání můžeme rozdělit a nechat část předzpracovat mimo software a zároveň nebudou vyžadovat při zpracování v FPGA mnoho paměti a také jejich implementací v FPGA získáme časovou úsporu takovou, že se vyplatí implementace v FPGA provádět.

Pokud by byla vybraná operace v rozporu s nejméně jedním bodem specifikace, bude implementace v FPGA buď úplně vyloučená, nebo nevýhodná.

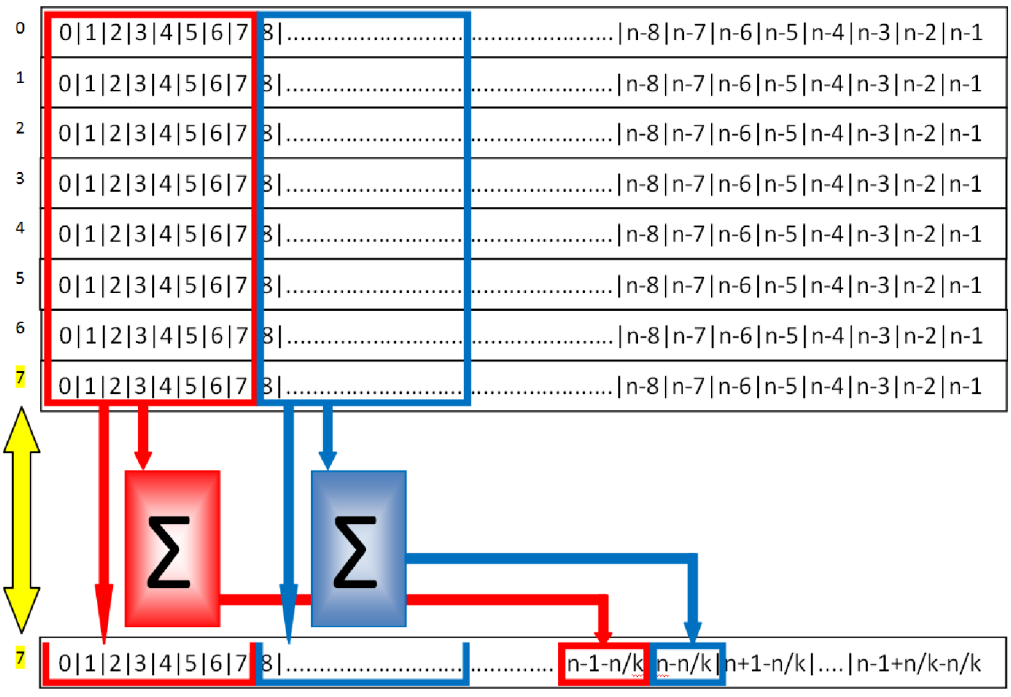
Segmentace prahováním jako testovací příklad

Po dlouhém zvažování jsem vybral jako demonstrační operaci metodu pro segmentaci obrazu popsanou v kapitole 2.

V tomto příkladě nelze očekávat žádné významné zrychlení, účelem je v tomto případě vyzkoušet implementaci operace v FPGA a možnosti zanesení upraveného obrazu do výstupních dat.



Obrázek 3.5: Rozhraní v kameře



Obrázek 3.6: Princip předzpracování obrazu

Kapitola 4

Řešení

Tato část se zabývá popisem vlastního řešení. Obsaženy jsou všechny etapy vývoje od úvodních pokusů.

Řešení probíhalo postupně v několika fázích. Nejprve jsem provedl test manipulace s daty v kameře. Zde bylo účelem ověřit správnost předpokladu, že lze bez větších komplikacá zasahovat do proudu dat. Potom jsem implementoval demonstrační software, jehož cílem a účelem bylo prokázat zrychlení výpočtu rastrové operace v případě, že na vstupu bude obraz modifikovaný v duchu myšlenky této práce. Následovala implementace funkce pro kameru. Samozřejmostí je řádné otestování řádné funkčnosti veškerých implementovaných prvků. Výsledky dílčích testů jsou zde obsaženy také.

4.1 Demonstrační software

Při tvorbě demonstračního softwaru se jako nejlepší jeví použití knihovny OpenCV. Výhod je celá řada: bezvadně vyřešené otevírání rastrových obrázků (včetně komprimovaných jako JPEG aj.), vyřešené ukládání obrázků, pro mě příjemná datová struktura představující bitmapu a zejména již dříve jsem s OpenCV pracoval.

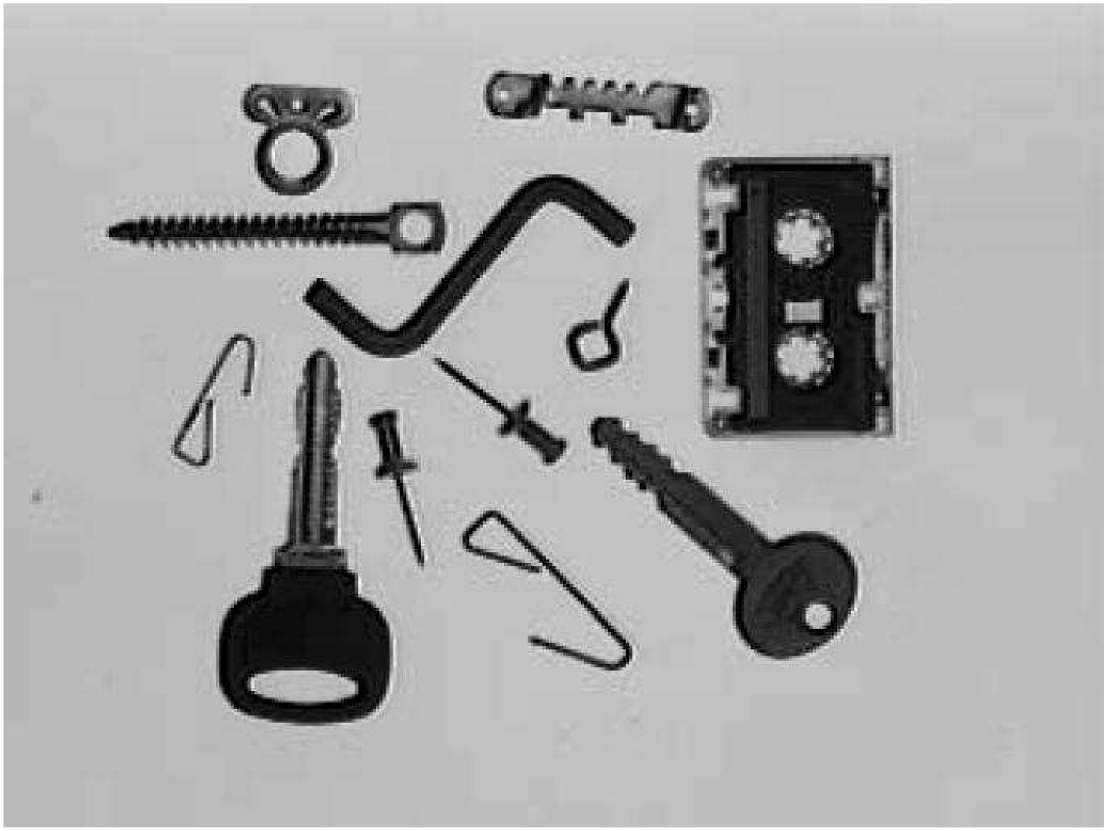
Aplikací jsem vytvořil více. První implementuje segmentaci prahováním. V souladu s teorií jsem při tvorbě programu ověřil, že je problém s nastavením prahu. V základní verzi tuto informaci musí dodat zkušený uživatel, což je značně nepohodlné a při automatizovaném řešení by to bylo i vyloučené. Proto jsem zvolil cestu, kdy se práh stanoví automaticky na základě obsahu obrazu.

Druhá aplikace také segmentuje prahováním, ale navíc řeší problém, který vznikne při nehomogenním osvětlení snímku. V takovém případě první aplikace pro větší či menší části obrazu zcela selhává. Zde je principem výpočet hodnot optimálního prahu v různých částech snímku a aplikace prahování na tyto různé části s různými prahy.

Prahování globální

Jedná se o konzolovou aplikaci naprogramovanou v jazyce C s využitím knihovny OpenCV. Program načte obrázek zadaný jako vstupní parametr v konzoli, provede nad ním požadovanou operaci a poté jej uloží jako bitmapu pod původním názvem rozšířeným o sufix ".prah.bmp".

Jak již bylo uvedeno v úvodu této části, aplikace provádí prahování s automatickou volbou prahu. Při tvorbě této funkce jsem vyšel z algoritmu 6.2 popisovaného v [4] na straně 181. Algoritmus jsem ale modifikoval pro práci s dvěma prahy namísto jednoho.



Obrázek 4.1: Testovací obrázek - předměty, převzato z přednášky [6]

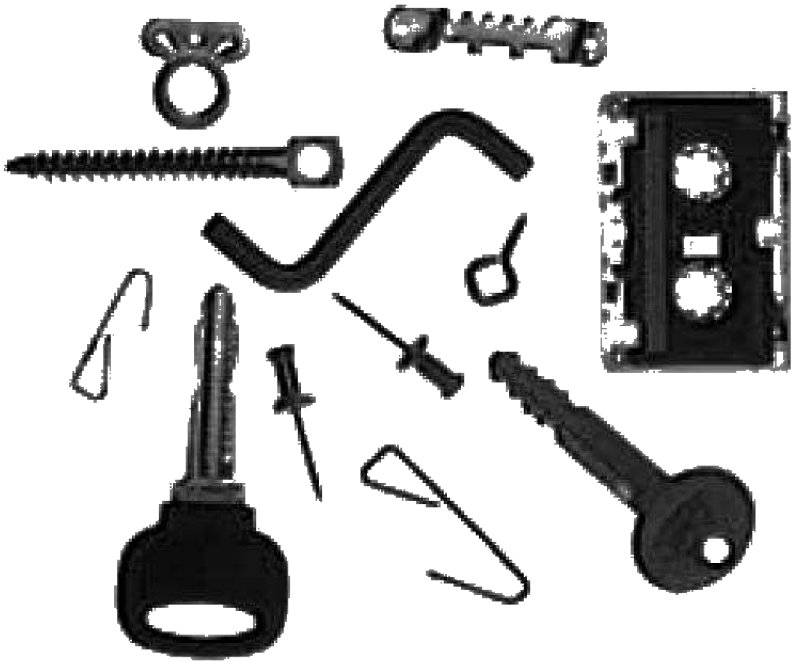
Algoritmus je iterativní, nejprve se zvolí náhodně nejaký práh, v mém případě dva prahy. Potom se provede prahování a zároveň se vytvoří tři histogramy: pod prahem jedna, nad prahem dva a mezi prahy. Ze středních hodnot sousedních histogramů se potom stanoví nový práh jako vážený průměr. S vypočtenými prahy se provede nový výpočet středních hodnot a stanoví se nové prahy. Experimántálně jsem zjistil, že pro stanovení prahů postačí pět iterací. Toto plyne i z přednášky Michala Španěla o segmentaci obrazu [6].

V souladu s literaturou jsem aplikaci otestoval na stejných obrazech - vytáhl jsem obrázky z přednášky Michala Španěla o segmentaci obrazu [6] a vyzkoušel segmentaci přímo s nimi. Zde uvedu příklady dva, první obrázek 4.1 patří k obrázkům, kde je výsledek uspokojivý, jak je vidět na obrázku 4.2.Je vidět, že předměty jsou celé, pozadí přitom zcela zmizelo.

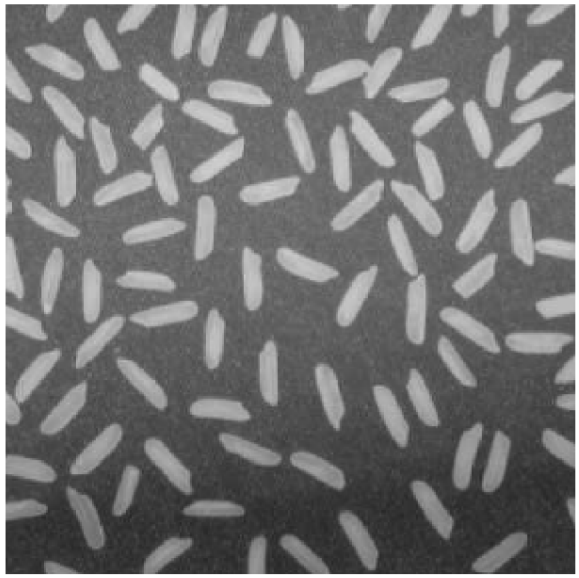
Další příklad ukazuje obrázek 4.3, který je nerovnoměně nasvícen. To je velký problém. Spočítám-li práh pro tmavou část obrázku, nefunguje to na světlé části a naopak. Při testování jsem si vypisoval hodnoty počítaných prahů. V případě takto nehomogenního osvětlení se jednotlivé iterace o nové prahy doslova praly. Jedna iterace určila práh, následující určila jiný, další zase ten první a takto se to opakovalo. Jak taková segmentace nefunguje, ukazuje obrázek 4.4. Je zde dob5e vidět, jak ve spodní polovině obrazu nachází program korektně zrna rýže, avšak jinde je výsledek naprosto špatný.

Adaptivní prahování

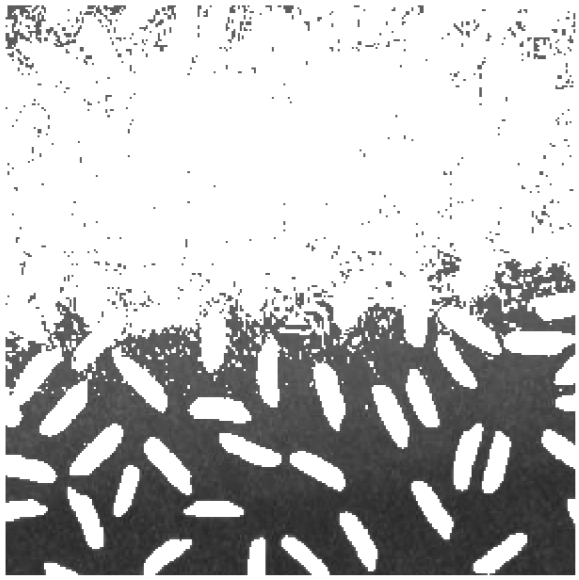
Jak potvrdil test s globálně počítaným prahem, zlobí tento způsob segmentace nerovnoměrné osvětlení. Řešením může být výpočet prahu ne pro celý obraz, ale jen pro jeho menší části



Obrázek 4.2: Testovací obrázek - předměty, po provedení segmentace



Obrázek 4.3: Testovací obrázek - rýže, převzato z přednášky [6]



Obrázek 4.4: Testovací obrázek - rýže, po provedení segmentace

a každou část segmentovat zvlášť.

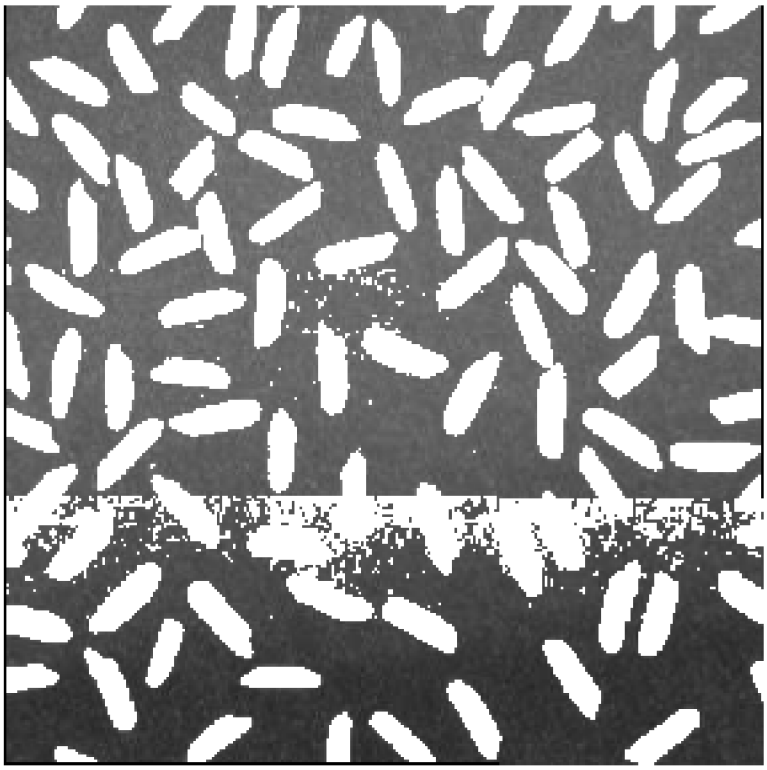
Segmentace po částech znamená rozdělit obraz na bloky vhodné velikosti a v nich spočítat prahy. Potom tyto části prahovat s odpovídajícími lokálními prahy. Problém může představovat zvolení správné velikosti bloku. Experimentálně jsem ověřil, že je-li blok příliš velký, pak přestává plnit svou funkci, jelikož nedokáže eliminovat nehomogenní osvětlení právě pro svou velikost. Problematický je i příliš malý blok. Zde se až tak moc sice neprojeví problém nehomogenního osvětlení, avšak vzniká zde problém, že celý blok je v hledaném objektu. Pak je vypočtený práh nekorektní. Jeden z možných výstupů aplikace je na obrázku 4.5. Je zde ještě vidět řada problémů, ale objketů nalezl algoritmus mnohem více, než předchozí verze.

Zhodnocení čistě softwareového řešení

Výsledky aplikací nejsou příliš zázračné a také nejsou překvapivé. Mě posloužili k poznání a ukázce, jak náročné co do množství výpočtů mohou být i tak jednoduché úlohy bez nějakých pokusů o optimalizace. Navíc je výpočet díky velkému množství iterací zdlouhavý, když aplikaci předložím fotku, je čekání na výsledek rozpoznatelné. To není rozhodně žádoucí stav, zvláště s přihlédnutím k ne zrovna nejlepším výsledkům metody.

Simulace zpracování obrazu s HW přezpracovanými daty

V této části se krátce věnuji poslednímu pokusu se softwarem v této práci. Vytvořil jsem aplikaci, která připraví data pro prahování do obrazu a obraz uloží, druhá aplikace modifikovaný obraz zpracuje. Výsledek není překvapivý, zatímco příprava prahů trvá dlouho, samotné zpracování je rychlé, což ukazuje na správnost tohoto přístupu.



Obrázek 4.5: Testovací obrázek - rýže, po provedení segmentace s adaptivním prahem

4.2 Implementace pro kameru

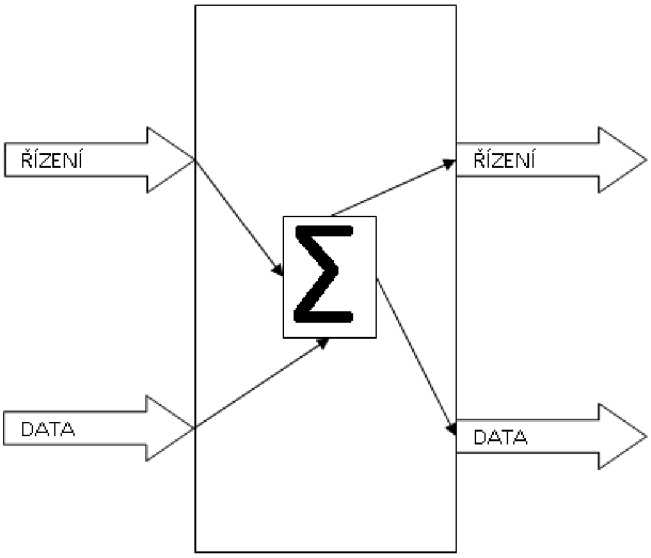
Implementace pro kameru probíhala v jazyce VHDL. Využíval jsem přitom simulaci v programu ModelSim, kde jsem sestavil testbench se signály odpovídajícími rozhraní v kameře (obrázek 3.5). Odsimulované funkce poté nahrával do kamery Jirka Šustek ze společnosti Camea, která mi poskytla platformu na vyzkoušení. Takový způsob práce byl pro mě něčím zcela novým a jiným od mého zaběhlého zvyku, kdy si program přeložím a vyzkouším ihned v počítači, na kterém pracuji.

Důležitou součástí práce byla simulace. K simulaci je zapotřebí vytvořit jak simulovanou komponentu, tak správné simulační prostředí. K vytvoření simulačního prostředí slouží Test Bench, což je VHDL kód, který je vytvořen k tomu, aby otestoval funkcionalitu a správnost modelované komponenty ve VHDL. [9].

V principu je implementace provedena jako čená skříňka nasazená do interface kamery, který je popsaný v návrhu 3, a ukazuje ho obrázek 3.5. Je nutné zachovat tok dat i řídící signály, jen data je možné modifikovat, ne však jejic uspořádání a časování. Pro znázornění principu lze použít obrázek 4.6.

Test manipulace s daty

Prvotní akce spočívala ve vyzkoušení manipulace s daty. Jak popisuje dřívější text a zejména obrázek 3.5, je zde synchronizovaný tok obrazových dat na výstup, který je potřeba vhodným způsobem upravit tak, aby nesl nějakou přidanou informaci. K tomuto účelu je nezbytně nutné umět data vyjmout z toku, provést s nimi nějaké modifikační operace a zase je do toku vložit. Pořadí dat musí zůstat zachováno a také musí být dodržena synchronizace signály



Obrázek 4.6: Princip zásahu do systému

hodiny, platný řádek a platný obraz.

Pro testovací funkci jsem zvolil aplikaci, kdy se na určité místo obrazu vloží nějaká konstantní data. Pro zvýšení názornosti a odstranění vlivu náhody jsem použil dvě svislé linky konstatní barvy, nic takového se v obraze určitě náhodně nevyskytne a bude tak na první pohled zřejmé, zda pokus funguje, či nikoli.

Jak vypadá výsledek testu, ukazuje obrázek 4.7. Červené elipsy nejsou součástí obrázku, ty jsou dodány pro zvýraznění oblasti manipulovaných dat dodatečně v grafickém editoru. Měněná data jsou bílý a šedý pruh konstatní barvy.

Testovací funkce je poměrně jednoduchá. Jak je vidět z grafu 3.5, jsou data synchronizována hodinovým signálem. Proto stačí počítat hodinové signály, které odpovídají pořadovému číslu aktuálního pixelu na řádku. V tomto případě se počítalo do sta - bílý pruh, a do dvou set - šedý pruh. Při napočítání daného počtu se provede s pixelem potřebná operace. Počítání hodin je jeden proces, samotná úprava dat je proces druhý.

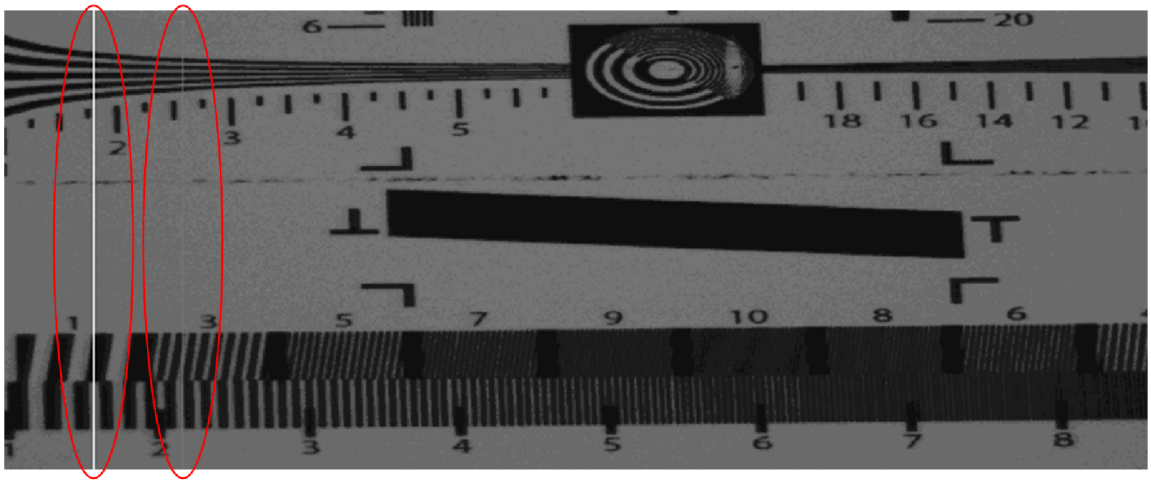
Test ukázal, že je možné do datového toku zasáhnout a data modifikovat.

Test práce s větším množstvím dat

Zatímco předchozí test jen zkoušel vložit konstantu, tento test si klade za cíl vkládat do toku dat hodnotu vypočtenou z dat obrazových tak, aby výsledek spadal do množiny operací popsaných v návrhu řešení.

Zde jsem provedl blokové zpracování. Kamera snímá monochromatický obraz o rozlišení 1392×1040 bodů. Tento obraz jsem rozdělil na bloky. Jejich velikost jsem zvolil dle filozofie pana docenta Kunovského libovolně, tedy vhodně. Vhodně v tomto případě znamená, že se mi s tím bude dobře pracovat a bude to dostatečně demonstrativní, nemusí to ale znamenat, že to je nejlepší z hlediska výpočtu, to může být předmětem dalšího zkoumání.

Bloky jsem zvolil o šíři 116 pixelů, což je přesně jedna dvanáctina šířky obrazu. V souladu s návrhem se dvanáctý sloupec obrazu zahodí a použije se pro přenos spočítancých dat.



Obrázek 4.7: Výstup z testovací funkce, červené elipsy jsou dodány pro zvýraznění sloupců dat, s kterými bylo manipulováno (bílý a šedý konstatní pruh)

Výška bloku není pevně dána a může se pohybovat v rozmezí od jednoho řádku do výšky celého sloupce, tj. až 1040 řádků. Proč zrovna takto? Znamená to pouhé přičítání hodnot. Kód pracuje tak, že počítá data na řádku, s každými hodinami jde jedna hodnota. A v určitých rozmezích přičítá aktulní data do odpovíající proměnné-čítače. Postupně se tak vždy projde celý řádek a nasčítají se hodnoty z jednotlivých úseků. Ty se poté zapíšou do datového toku od pozice 1300. Další řádek se ovšem číatče nenulují ale přičítají dále a zase zapíšou hodnoty na konec. Tak se postupně hodnoty zvyšují a každá obsahuje v sobě i hodnoty předchozí. Tento postup umožňuje při zpracování obrazu vzít sumu hodnot z bloku o téměř libovolné velikosti. Blok může být navíc plovoucí ve sloupci.

Princip činnosti ukazují náseldující ukázky kódu.

Listing 4.1: Plnění čítače

```
--zkusime soucet v radcich
if (counter < 116 ) then
  block_1_counter <= std_logic_vector(unsigned(block_1_counter)
                     + unsigned(data_in));   --prvni blok
end if;
```

Čítače bloku `block_X_counter` kde `X` je hodnota od jedné do jedenácti přičítající aktuální data z datového toku v kameře `data_in`. Toto se děje ve všech sloupcích/blocích, `counter` přitom říká, v kterém sloupci dat se práve systém nachází.

Po pruchodu celým řádkem jsou vypočtené sumy, je potřeba je uložit, jak ukazuje 4.2.

Listing 4.2: Ukládání dat do bitmapy

```
--ted ta spocitana data nahazim na konec obrazu
    --protoze jsou 32 bit tak je rozdelim,

    if (counter = 1300) then
         data_out <= block_1_counter(31 downto 24);
    end if;

    if (counter = 1301) then
```

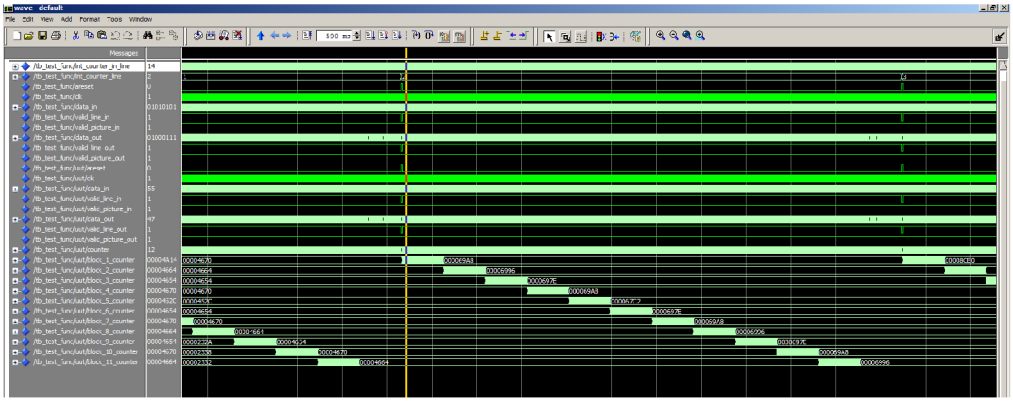
```
        data_out <= block_1_counter(23 downto 16);
    end if;

    if (counter = 1302) then
        data_out <= block_1_counter(15 downto 8);
    end if;

    if (counter = 1303) then
        data_out <= block_1_counter(7 downto 0 );
    end if;
```

Pro všechny čítače bloku se provede kód poodobný tomu v ukázce. Čítače jsou kvůli velké velikosti bloku a postupnému sumovaní velké 32b, data v kameře jsou osmibitová, proto se rozdují.

Tanto systém jsem simuloval v ModelSimu, následují ukázky výstupů ze simulace. Obrázek 4.8 slouží pouze jako ukázka postupné práce čítačů bloků, to je ta schodovitá oblast dole. Dat je mnoho, proto nejsou v tomto případě vidět konkrétní hodnoty, avšak je vidět globální pohled na zpracování jednoho celého řádku obrazu.

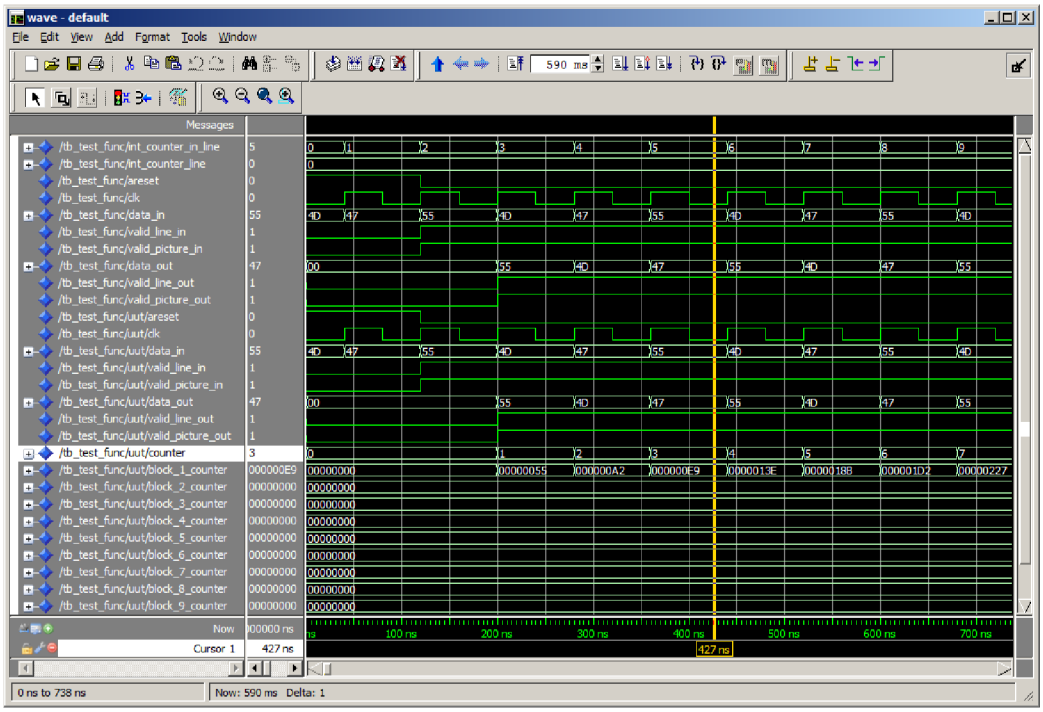


Obrázek 4.8: Simulace chování obvodu 1

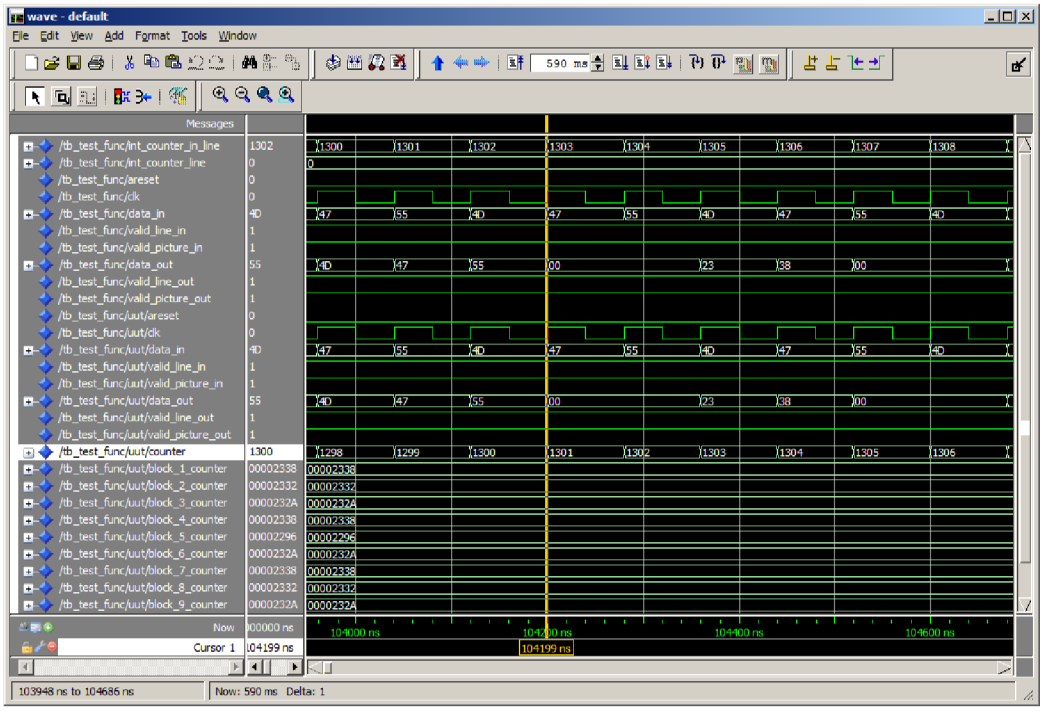
Podrobnější už je obrázek 4.9. Zde je dobře vidět počáteční reset, nastavení signálů rozhraní, generovaná data i postupné plění čítače pro první blok. Generování dat a řídících signálů probíhá podle rozhraní v kameře. Vše je synchronizováno hodinovým signálem, s daty se pracuje až po naskočení `valid_line`.Za povšimnutí stojí zpoždění signálů `x_out` za signály `x_in` o jeden takt hodin. To je dáno tím, že data na jednu hodinovou hranu systém přečte na svém vstupu, a až s další nábežnou hranou je může umístit na výstup.

Následuje obrázek 4.10. Zde je dobře vidět stav, kdy se dosáhne pixelu číslo 1300 a začne kopírování dat z čítačů do výstupního proudu po osmibitových blocích.

Tento test ukázal, že je možné s daty manipulovat i v rámci vetších celků a modifikovat určitá obrazová data na základě obrazových dat z předhocích částí obrazu.



Obrázek 4.9: Simulace chování obvodu 2



Obrázek 4.10: Simulace chování obvodu 3

Kapitola 5

Závěr

Cílem projektu bylo navrhnout a implementovat přesun části výpočetně náročného zpracování obrazu z PC do FPGA. Předpokladem bylo, že je takový přesun možný a tento předpoklad se ukázal jako správný.

Pro splnění stanoveného cíle jsem nastudoval problematiku reprezentace rastrového obrazu a také principy práce s rastrovým obrazem. Zaměřil jsem se přitom zejména na operace užitečné v oblasti počítačového vidění a také počítačové grafiky. Neméně důležitou část studia jsem věnoval problematice FPGA a s tím souvisí i studium jazyk VHDL. V souvislosti se zpracování obrazu v HW jsem se také seznámil s DSP a způsoby programování vestavěných systémů, neboť to je neopomonutelná varianta ve světě digitálního obrazu nebo obecně digitálního signálu.

Po nastudování zmiňované problematiky jsem rámcově popsal množinu operací vhodných pro zpracování v FPGA a souhlasil s výběrem nabídnuté zvláště vhodné existující platformy pro vyzkoušení, že je řešení podle návrhu této práce možné, práce díky tomu nemusela být zcela abstraktní a mohl jsem simulovat proti existujícímu rozhraní. To přináší zejména tu výhodu, že je zřejmé, že se pracuja na něčem, co existuje a co se dá realizovat. To je alespoň pro mě velmi důležité.

Poté jsem provedl první experimentální manipualci s daty v kameře a zabýval se pokusy s reálným zrychlením v rámci simulace v PC, kdy jsem vybranou úlohu nad obrazem zpracoval běžným způsobem a také způsobem, kdy jsem měl k dispozici obraz z předzpracovanými daty podle myšlenky této práce. Přitom jsem vyzkoušel i segmentace obrazu jednoduchou, avšak výpočetně dosti náročnou metodou prahování.

Poté jsem simuloval o něco složitější proces ve VHDL. Tato simulace ukazuje, že lze pracovat s bloky dat, což je velmi důležité. Podařilo se tak nasimulovat vlastně černou skříňku, která se zapojí do toku dat a na vstupu dostává obrazová data a řídící signály, na výstup potom odesílá neměnné řídící signály a data modifikovaná.

Ač možná vypadá práce snadná, nebylo to celé až tak lehké. Kvalitní podklady jsou zpravidla zahraniční a často dosti útržkovité, hlavně na internetu, jejich vyhledání dalo nemálo práce. Největším trápením této práce bylo programování ve VHDL. Samotný jazyk by asi nebyl tak složitý, ale jako celek to bylo nové, je to trochu jiný způsob práce než třeba Java nebo C. Vytvořit vlastní simulaci, vytvořit v ní simulovanou komponentu, to nebylo lehké. V této oblasti musím ještě hodně pokročit, budu-li mít možnost se FPGA dále věnovat. Naopak docela dobře šlo sepisování zprávy potom, co jsem nastudoval nalezené podklady.

Experimentální výsledky tedy ukázaly, že lze vhodným způsobem přímo v hardware předzpracovat obrazová data ještě před jejich odesláním do softwaru a současně předzpraco-

vané výsledky přenést v rámci odesílaného obrazu. Toto je velmi důležitý výsledek a zároveň přínos této práce. Implementaci, otestovaní a v ideálním případě i nasazené tohoto principu v konkrétní průmyslové aplikaci považuji za nejlepší způsob pokračování práce na tomto projektu. Je zde přitom velký prostor pro správnou volbu aplikace a také pro rozdělení mezi hardware a software.

Kapitola 6

Přílohy

Součástí této práce je přiložené CD, které obsahuje kompletní text této práce, obrázky v ní použité a veškeré náležitosti potřebné pro znovuvytvoření tohoto dokumentu v systému LaTeX. Dále jsou obsaženy veškeré zdrojové kódy, které vznikly při vypracování této práce. Organizace dat na CD je i přes svou zřetelnost na CD popsána v souboru readme.txt umístěným v kořenovém adresáři CD. Každá aplikace pak obsahuje textový soubor s návodem na použití a kompilaci.

Literatura

[1] CAMEA, s. s.: UnicamD2, Inteligetní kamera pro použití v dopravní technice. 19.2.2007.
URL www.camea.cz

[2] Doseděl, T.: Zkusili jsme LTE, aneb 50 megabitů vzduchem. 26. 10. 2008.
URL http://www.mobilmania.cz/clanky/zkusili-jsme-lte-aneb-50-megabitu-vzduchem/sc-3-a-1120620/default.aspx

[3] Hauck, S.; Dehon, A.: *Reconfigurable Computing: The theory and practice of FPGA-BASED Computation*. Morgan Kaufmann Publishers, 2008, iSBN 978-0-12-370522-8.

[4] Šonka, M.; Hlaváč, V.; Boyle, R.: *Image Processing, Analysis, and Machine Vision*. Thomson, třetí vydání, 2008, iSBN 0-495-24438-4.

[5] OpenCVWiki: Welcome - OpenCV Wiki.
URL http://opencv.willowgarage.com/wiki/

[6] Španěl, M.: Segmentace obrazu, analýza histogramu, analýza barev, shlukování, přednáška.
URL https://www.fit.vutbr.cz/study/courses/POV/private/lectures/pov_04_segmentace_obrazu.pdf

[7] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer press, 2004, iSBN 80-251-0454-0.

[8] Steven W. Smith: *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, San Diego, CA 92150-2407, P.O.BOX 502407, 1997, iSBN 0-9660176-3-3.

[9] Teemu Pitkänen: VHDL Test Bench.
URL http://www.tkt.cs.tut.fi/kurssit/1216/K06/Luennot/lecture5.pdf

[10] Texas Instruments: Digital Still Camera.
URL http://focus.ti.com/docs/solution/folders/print/80.html

[11] Texas Instruments: Programmable DSP Platform for Digital Still Cameras. 2000.
URL http://focus.ti.com/lit/an/spra651/spra651.pdf

[12] Texas Instruments: TMS320DM355 Digital Media System-on-Chip. 2009.
URL http://focus.ti.com/lit/ds/symlink/tms320dm355.pdf

[13] Xilinx: XtremeDSP Video Starter Kit - Spartan-3A DSP Edition.
URL http://www.xilinx.com/products/devkits/DO-S3ADSP-VIDEO-SK-UNI-G.htm