VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HARDWAROVÁ AKCELERACE FILTRACE OBRAZU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Martin Fiala
AUTHOR

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HARDWAROVÁ AKCELERACE FILTRACE OBRAZU

HARDWARE ACCELERATION OF IMAGE FILTRATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Martin Fiala
AUTHOR

VEDOUCÍ PRÁCE Ing. Jan Kořenek
SUPERVISOR

BRNO 2007

Zadání:

1. Seznamte se s jazykem VHDL a architekturou vyukového přípravku FITkit.
2. Nastudujte algoritmy filtrace obrazu pomocí Laplaceova operátoru, Sobelova operátoru a mediánového filtru.
3. Algoritmy naimplementujte v jazyku C. Funkčnost implementace ověřte na přípravku FITkit pomocí MCU.
4. Navrhněte obvodovou realizaci časově kritických částí algoritmů a proveďte jejich implementaci v jazyku VHDL.
5. Funkčnost implementace ověřte na přípravku FITkit.
6. V závěru diskutujte oba přístupy z pohledu výkonosti a ceny implementace.

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Obsahem tohoto diplomového projektu je seznámení se s problematikou filtrace obrazu, zejména s teoretickými východisky jejichž původ náleží do teorie lineárních systémů a matematické analýzy funkcí více proměnných. Jsou zde vysvětleny některé přístupy a metody používané při vyhlazování obrazu a při detekci hran v obraze. Pozornost je zaměřena především na mediánový filtr, Sobelův operátor a Laplaceův operátor. Hlavní náplní projektu je diskuze různých přístupů k hardwarové akceleraci filtrace obrazu a návrh časově efektivně pracujících softwarových i hardwarových realizací filtrů v podobě programových funkcí a kombinačních obvodů s využitím teoretických znalostí o časové složitosti algoritmů. Také byly provedeny hardwarová i softwarová implementace jmenovaných filtrů na výukovém přípravku FITkit. Byl změřen čas filtrace všech implementovaných filtrů a výsledky porovnány a analyzovány.

Klíčová slova

Filtrace obrazu, Laplaceův operátor, Sobelův operátor, mediánový filtr, FPGA, FITkit, detekce hran, vyhlazování obrazu, akcelerace filtrace obrazu

Abstract

This master’s thesis contains introduction to image filtration problems, especially to theoretical outlets, whose origin lies in linear systems theory and mathematical function analysis. There are described some approaches and methods which are used to image smoothing and for edge detection in an image. Mainly Sobel operator, Laplace operator and median filter are covered. The main contents of this project is discussion of some approaches of hardware acceleration of image filtering and design of time effective software and hardware implementations of filters in a form of program functions and combinational circuits using theoretical knowledges about time complexity of algorithms. Hardware and software implementation of named filters was done too. For every filter, time of filtration was measured and results were compared and analyzed.

Keywords

Image filtration, Laplace operator, Sobel operator, median filter, FPGA, FITkit, edge detection, image smoothing, image filtration acceleration

Citace

Fiala Martin: Hardwarová akcelerace filtrace obrazu. Brno, diplomová práce, FIT VUT v Brně, 2007.

Hardwarová akcelerace filtrace obrazu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Kořenka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Děkuji vedoucímu diplomového projektu, panu Ing. Kořenkovi, za cenné informace, rady a podněty poskytnuté při vypracování diplomového projektu.

© Martin Fiala, 2007.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Obsah ... 1
1 Úvod ... 3
2 Teoretický rozbor ... 5
2.1 Diskrétní konvoluce ... 5
2.2 Diskrétní aproximace gradientu a Laplaciánu (Sobelův a Laplaceův operátor), detekce hran ... 9
2.3 Lokální vyhlazování obrazu ... 12
2.4 Algoritmy pro nalezení mediánu ... 14
2.5 Způsoby ošetření nelineárních vlastností obrazového signálu ... 15
2.6 Technologie FPGA ... 17
2.7 Výukový přípravek FITkit ... 22
2.7.1 Komponenty a způsoby užití FITkitu ... 22
2.7.2 Rozhraní SPI na FITkitu ... 23
2.8 Využití řetězeného zpracování ... 25
3 Architektura ... 26
3.1 Koncepce užitá při hardwarové realizaci filtrů ... 26
3.2 Základní popis programů softwarové části hardwarového řešení a programů filtrů ... 29
3.3 Architektura řetězené datové linky a SPI ... 30
3.4 Architektura kombinačních obvodů filtrů ... 35
3.4.1 Architektura kombinačního obvodu Sobelovba operátoru ... 35
3.4.2 Architektura kombinačního obvodu Laplaceova operátoru ... 36
3.4.3 Architektura kombinačního obvodu mediánového filtru ... 37
3.4.4 Zřetězení kombinačních obvodů filtrů ... 39
3.5 Návrh programové realizace výpočetních procesů filtrů ... 40
3.5.1 Návrh programové realizace filtrů pracujících na základě Sobelova a Laplaceova operátoru ... 40
3.5.2 Návrh programové realizace mediánového filtru ... 41
4 Zhodnocení implementace filtrů ... 42
4.1 Softwarové filtry ... 42
4.1.1 Sobelův a Laplaceův operátor ... 42
4.1.2 Mediánový filtr ... 43
4.2 Hardwarové filtry ... 44
4.2.1 Sobelův operátor ... 45

4.2.2 Laplaceův operátor ... 46
4.2.3 Mediánový filtr ... 47
4.2.4 Srovnání se softwarovou realizací ... 48
4.2.5 Vliv SPI rozhraní na možnosti hardwarové akcelerace filtrace obrazu na přípravku FITkit ... 50
5 Závěr ... 52
Použitá literatura a zdroje informací ... 53

1 Úvod

Digitální filtrace obrazu je způsob zpracování digitálního obrazového signálu obdobný filtraci obecných číslicových signálů. Obraz je reprezentován vícerozměrným, nejčastěji dvojrozměrným, signálem. Digitální obraz je možné chápat jako matici, jejíž prvky jsou obrazové elementy (pixely), jejichž hodnota je rovna hodnotě obrazové funkce v odpovídajícím místě obrazu. Při filtraci obrazu nastávají některé odlišnosti vzhledem k poznatkům známým z teorie signálů a nemusí být splněny všechny podmínky, které implikují platnost některých vztahů.

Filtrace obrazu nachází uplatnění v různých úlohách analýzy obrazu, zejména ve fázi jeho předzpracování prováděného za účelem jeho úpravy nebo zvýraznění některých jeho vlastností, na základě jejichž znalosti probíhá další zpracování. Často se využívají obrazové filtry, jejichž výstupem je aproximace hodnoty změny obrazové funkce a které slouží k detekci hran v obraze. Detekce hran je zpravidla první fází v technikách zpracování obrazu, které se využívají v oblasti počítačového vidění a umožňuje provedení lokalizace objektů zachycených v obraze. Tyto filtry obecně slouží ke zvýšení kvality obrazu pro lidskou nebo strojovou interpretovatelnost, kde kvalita je měřena subjektivně. Lze je využít i pro ostření obrazu – tedy pro zvětšení strmosti hran v obraze za účelem získání kontrastnějšího obrazu. V analogii ke klasickým filtrům je tento typ obdobou filtrů typu horní propust. Jinou často užívanou skupinou filtrů jsou filtry sloužící k vyhlazování obrazu, které usilují o potlačení šumu a osamocených fluktuací hodnot obr. funkce. Tyto jsou jakýmsi protikladem k předchozímu typu a jsou příbuzné tradičním filtrům typu dolní propust. Obvyklým problémem vyžadujícím jejich nasazení může být např. odstranění tzv. výstřelového (i jiného) šumu. Obou typů se využívá hojně v oblasti počítačového vidění a počítačové grafiky, např. při nefotorealistickém zobrazování objektů, speciálních grafických efektech ad.

Jako u všech výpočetních problémů, existuje i v případě filtrace obrazu snaha o dosažení nejnižší možné prostorové a časové složitosti výpočetních algoritmů. U některých filtrů je možné výrazně ovlivnit složitost výběrem vhodného algoritmu, u jiných je metoda výpočtu jednoznačná. Potenciál softwarových metod akcelerace je ale omezený a je-li vyčerpán, je nutné „sáhnout“ k hardwarovému způsobu realizace.

Všechny filtry, které jsou předmětem zájmu této práce, využívají pouze hodnot pixelů v malém okolí reprezentativního pixelu (právě zpracovávaného) ve vstupním obraze, na základě kterých určí (novou) hodnotu reprezentativního pixelu ve výstupním obraze (lokální filtrace).
To znamená, že teoretická prostorová složitost je konstantní, kde konstanta je relativně malá hodnota, což je možné považovat za optimální stav. Dostatečné množství paměti k uložení celého snímku je však v dnešní době poměrně snadno opatřitelné a pro dosažení uspokojivého času zpracování se jedná o nezbytný předpoklad.

Vlastnost na kterou je však vzhledem k nasazení v aplikacích reálného času kladen větší důraz a ve které některé obrazové filtry vykazují podstatně méně příznivé výsledky, je rychlost zpracování. Nejhorší časová složitost v závislosti na počtu pixelů vstupního obrazu při použití určité neměnné velikosti okolí pixelu dosahuje obvykle při sekvenčním zpracování lineárního řádu složitosti. Faktorem, který zásadně ovlivňuje celkovou výkonnost, je časová složitost výpočtu hodnoty jednoho pixelu obrazu v závislosti na počtu vstupních pixelů jeho okolí, která dosahuje u některých filtrů lineárního a při nepoužití nejlepších známých metod až polynomiálního řádu složitosti (např. mediánový filtr). Z důvodu neexistence či neznalosti rychlejších algoritmů spočívá jediná možnost další akcelerace v jejich paralelizaci a realizaci jednotlivých výpočetních kroků specializovanými hardwarovými komponentami. Tak mohou probíhat některé operace současně a eliminuje se i režie související se zpracováním instrukcí (dekódování apod.), která nastává při provádění úlohy na procesoru.

Tento projekt je zaměřen zejména na možnosti akcelerace Sobelova a Laplaceova operátoru a mediánového filtru, přičemž jsou nejprve připomenuty některé základní informace

týkající se filtrů a filtrace obecně, je popsán obecný postup užívaný v praxi při zpracování obrazu a nechybí ani matematické odvození vztahů, na jejichž základě výše jmenované filtry pracují. Tyto součásti byly vypracovány a nastudovány v rámci semestrálního projektu, na který tento diplomový projekt navazuje. Hlavní náplní této práce je však návrh různých variant obvodů a diskuze algoritmů, kterými lze filtry efektivně realizovat. Z návrhů byly vybrány některé perspektivní a následně byly softwarově i hardwarově implementovány na výukovém přípravku FITkit, který disponuje mikrokontrolerem a technologií FPGA a jeví se tedy jako vhodný pro srovnání výpočetní složitosti obou přístupů realizace.

2 Teoretický rozbor

Nejprve budou připomenuty některé základní matematické poznatky z teorie signálů, zejména z oblasti teorie lineárních systémů. Toto odvětví matematiky poskytuje teoretický základ pro mnohé typy filtrů používané při zpracování signálů a tedy i při digitálním zpracování obrazu. Dále bude následovat stručné shrnutí pojmů gradient a Laplacián známých z matematické analýzy, kde hrají významnou roli při analýzách průběhů funkcí více proměnných a budou odvozeny některé konvoluční masky aproximující gradient a Laplacián. Pozornost bude věnována také lineárním a některým nelineárním filtračním metodám umožňujícím provádění vyhlazování obrazu. Na závěr této kapitoly bude stručně popsána technologie FPGA a architektura výukového přípravku FITkit, na kterých byla provedena implementace.

2.1 Diskrétní konvoluce

Mějme systém S, který může být interpretován jako černá skříňka se vstupem $f(x)$ a výstupem $g(x) = S(f(x))$:

$$f(x) \longrightarrow S \longrightarrow g(x)$$

V našem případě je signál $f(x)$ původní obraz, representovaný pro jednoduchost v jedné dimenzi, $g(x)$ je filtrovaný výstupní obraz a S je filtrační operace.

Systém S je **lineární**, jestliže jsou splněny vlastnosti **aditivita a homogenita**. Mějme vstupní signály f_1, f_2 a výstupní signály g_1, g_2:

Jestliže

$$f_1(x) \longrightarrow S \longrightarrow g_1(x) \quad \text{a} \quad f_2(x) \longrightarrow S \longrightarrow g_2(x)$$

potom platí

$$f_1(x) + f_2(x) \longrightarrow S \longrightarrow g_1(x) + g_2(x),$$

když systém splňuje podmínku aditivity a

$$a\, f_1(x) \longrightarrow S \longrightarrow a\, g_1(x),$$

když systém splňuje vlastnost homogenity.

Chování systému vzhledem k nějaké veličině x je **invariantní**, jestliže platí, že pokud na vstupní signál $f(x)$ zareaguje systém výstupním signálem $g(x)$, potom na signál $f(x-x_0)$ zareaguje signálem $g(x-x_0)$.

Nebo-li při použití stejného zápisu jako výše:

Jestliže

$$f(x) \longrightarrow S \longrightarrow g(x),$$

potom platí

$$f(x-x_0) \longrightarrow S \longrightarrow g(x-x_0),$$

když systém splňuje vlastnost invariantnosti.

Veličina *x* může být spojitá nebo diskrétní. V digitálním zpracování obrazu se jedná o diskrétní veličinu, která má význam pozice obrazového elementu (pixelu) v obraze a funkce *f* a *g* jsou funkcemi dvou proměnných (pozice ve směru osy x, y v kartézském souřadném systému).
Z teorie lineárních systémů vyplývá klíčová vlastnost chování filtru, která spočívá v tom, že jestliže filtr splňuje podmínky linearity a časové (prostorové, posunové) invariantnosti, potom výstup filtru může být matematicky vyjádřen pro spojitý signál v podobě:

$$g(x) = \int f(t)h(x-t)dt\,, \qquad (2.1.1)$$

kde *h(t)* je tzv. impulsní odezva, což je funkce, která kompletně charakterizuje filtr a *x* i *t* značí časovou veličinu. Výše uvedený vztah se nazývá konvoluční integrál a zkráceně se zapisuje $g = f * \mathrm{h}$.
V případě diskrétního signálu se provede výpočet o stejném významu jako (2.1.1) součtem:

$$g(i) = \sum_{k=-\infty}^{+\infty} f(k)h(i-k) \qquad (2.1.2)$$

a ačkoli jsou meze součtu nekonečné, funkce h je obvykle nulová mimo určitý interval. Jestliže je rozsah, kde je *h* nenulová $(-w,+w)$, může být vztah 2.1.2 zapsán jako:

$$g(i) = \sum_{k=i-w}^{i+w} f(k)h(i-k) \qquad (2.1.3)$$

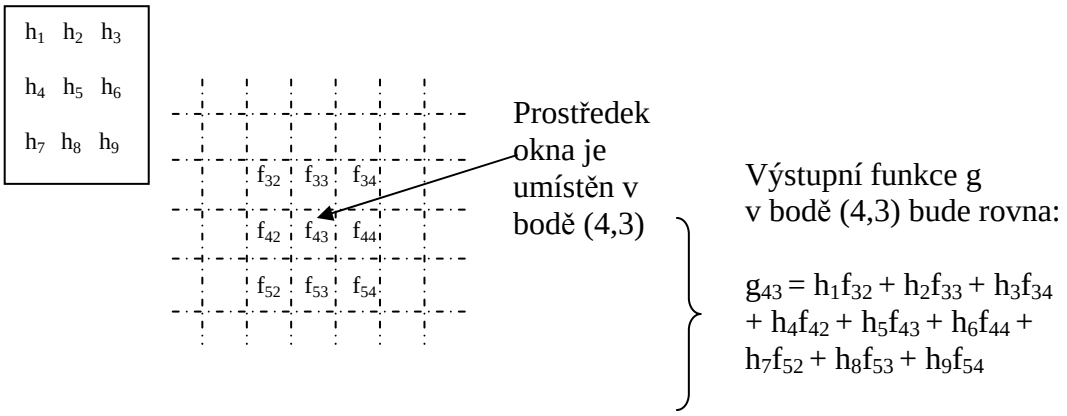
Tento vztah říká, že výstup *g(i)* v bodě i je dán váženým součtem vstupních bodů obklopující bod i, kde váhy jsou dány hodnotami *h(k)*. K výpočtu výstupní hodnoty dalšího bodu s indexem i + 1 je (jak je patrné ze vztahu 2.1.3) funkce *h* posunuta o jeden bod a vážený součet znovu spočítán. Celý výstup je vytvořen sérií těchto operací, která se nazývá **diskrétní konvolucí**.
Ve dvou rozměrech se stane impulsní odezva dvourozměrnou funkcí *h(k,l)* a výpočet (2.1.3) se změní na dvojnásobný součet:

$$g(i,j) = \sum_{k=i-w}^{i+w} \sum_{l=j-v}^{j+v} f(k,l)h(i-k, j-l) \qquad (2.1.4)$$

Funkce *h* se někdy nazývá filtrovým jádrem nebo konvoluční maskou (případně okénkovou funkcí). Z důvodu symetrie jsou téměř vždy její rozměry zvoleny v obou směrech liché a maska má často tvar čtverce.

Obecná podoba konvoluční masky s hodnotami impulsní odezvy na pozicích h_1 až h_9 je ilustrována na obr. č. 1 vlevo. Hodnoty masky impulsní odezvy se budou násobit s hodnotami vstupní funkce, které leží na stejných relativních pozicích vzhledem k reprezentativnímu pixelu, jako je relativní pozice v masce impulsní odezvy vzhledem k prostřednímu prvku. První index každého prvku vstupní i výstupní obrazové funkce odpovídá pozici prvku v obraze ve vertikálním směru a roste směrem dolů, druhý odpovídá pozici v horizontálním směru a roste směrem doprava. Maska se umístí do mřížky obrazu tak, aby byl její prostřední prvek v místě právě zpracovávaného pixelu. Každá hodnota impulsní odezvy se vynásobí s hodnotou pixelu, na jehož pozici v mřížce leží a výsledky se sečtou. V dalším kroku se okno s hodnotami impulsní odezvy posune o jednu položku (pixel) a stejným způsobem se znovu provede výpočet. Diskrétní konvoluce je dosaženo tím, že se systematicky (většinou po řádcích) prochází pixel po pixelu celý obraz a provádí se výše popsané násobení a sčítání. Takovýmto

způsobem se uskutečňuje praktická realizace vztahu (2.1.4) při zpracování obrazu. Postup je ještě ilustrován na následujícím obr.



Obrázek 1 Výpočet diskrétní konvoluce obrazového signálu s impulsní odezvou reprezentovanou obecnou čtvercovou konvoluční maskou.

Upozorněme ještě na jeden fakt týkající se této problematiky. Bude-li proveden výpočet výstupní funkce g na pozici (4,3) v mřížce podle vztahu (2.1.4), dostaneme:

$$g_{43} = h_{11}f_{32} + h_{10}f_{33} + h_{1\text{-}1}f_{34} + h_{01}f_{42} + h_{00}f_{43} + h_{0\text{-}1}f_{44} + h_{\text{-}11}f_{52} + h_{\text{-}10}f_{53} + h_{\text{-}1\text{-}1}f_{54}$$

Vztah (2.1.4) platí pro obecný diskrétní signál zasazený do dvojrozměrného pravoúhlého souřadného systému. Souřadnice všech prvků funkce f jsou celočíselné, protože f je diskrétní a prvky je tedy možné indexovat jako prvky dvojrozměrného pole a přenést je do rastrové mřížky. Ve vztahu se vyskytují i hodnoty indexů impulsní odezvy, které jsou záporné, protože indexy naznačují prostorovou orientaci prvků vzhledem k pozici „prostorově prostředního“ prvku odezvy a v některých směrech (osy y a x) jsou prvky na pozici o jedna menší. V osovém souřadném systému by byla odezva zakreslena tak, že by prostřední prvek h_{00} ležel v počátku a ostatní prvky v jeho sousedství tak, aby byly ve shodě se způsobem zakreslení vstupní funkce (stejný směr pro rostoucí index, resp. rostoucí nezávislou veličinu). Kdyby byla zakreslena impulsní odezva v rastru tak, aby byl způsob indexování stejný jako v případě indexování hodnot funkce f na obr. 1, byla by vzájemná pozice prvků odezvy:

$$\begin{matrix} h_{\text{-}1\text{-}1} & h_{\text{-}10} & h_{\text{-}11} \\ h_{0\text{-}1} & h_{00} & h_{01} \\ h_{1\text{-}1} & h_{10} & h_{11} \end{matrix}$$

Obrázek 2 Rozmístění hodnot impulsní odezvy v rovině v souladu se vztahem 2.1.4

Znamená to, že pokud bychom prováděly výpočet striktně podle vztahu (2.1.4) s takto rozmístěnými hodnotami impulsní odezvy, pak by se jednotlivé hodnoty odezvy po umístění středu okna na pozici právě počítaného pixelu nenásobily s hodnotami prvků, na jejichž pozici by ležely, jako tomu bylo v případě na obr. 1. Stejná situace je známá ze zpracování jednorozměrných signálů, kde musí být při násobení s hodnotami vstupního signálu (pro výpočet konvoluce) impulsní odezva zrcadlově otočena. Tato vlastnost je dána vztahy (2.1.3), resp. (2.1.4). Aby prvky (při stejném způsobu indexace) korespondovaly s hodnotami obrazové funkce, s kterými se mají vynásobit, musely by být přemístěny tak, jak je to ilustrováno na obr. 3. Indexy by potom neurčovaly relativní pozici vzhledem k prostřednímu prvku odezvy,

pokud by byly interpretovány v souladu se způsobem indexování obrazových elementů použitém ve vstupním rastrovém obraze a neměli by žádný užitečný význam. Použití okénkové masky pro znázornění impulsní odezvy je tedy zjednodušující intuitivní záležitostí umožňující usnadnění v tom, že hodnoty odezvy jsou umístěny přímo na pro ně požadované pozice v okolí bodu, ale způsob výpočtu diskrétní konvoluce nevychází ze skutečného rozmístění hodnot funkce h v rovině, jak tomu naopak je ve vztahu 2.1.4 (kde se použije pro vynásobení s určitou hodnotou signálu prvek odezvy na pozici, která je s pozicí prvku odezvy, jenž leží na této pozici signálu, symetrická vzhledem ke středu masky).

h_{11}	h_{10}	h_{1-1}
h_{01}	h_{00}	h_{0-1}
h_{-11}	h_{--10}	h_{-1-1}

Obrázek 3 Přemístění hodnot impulsní odezvy na obr. 2 tak, aby bylo možné provádět filtraci způsobem popsáném na obr.1

V některých případech může být vícerozměrná maska rozložena na vektorový součin jednorozměrných masek. Jinak řečeno, lze rozložit vícerozměrný systém bázových ortogonálních funkcí lineární integrální transformace na součin jednorozměrných ortogonálních funkcí. Vztah lze popsat následovně:

$$h(l,m) = h_v(l)h_h(m)^T, \qquad (2.1.5)$$

kde $h_v(l)$ je maska ve vertikálním směru a $h_h(m)$ je maska v horizontálním směru. Platí-li vztah (2.1.5), potom je funkce *h* **separabilní** a odpovídající filtr je **separabilním filtrem**. Separabilita je důležitou vlastností filtru z výpočetního hlediska, protože znamená, že filtr může být realizován tak, že se obraz nejprve zkonvoluuje s impulsní odezvou odpovídající vektoru h_h a poté se zkonvoluuje výsledek získaný první konvolucí s impulsní odezvou odpovídající vektoru h_v. Tím se vymění dvourozměrná konvoluce za dvě jednorozměrné konvoluce, což se projeví jako redukce složitosti výpočtu z polynomiálního řádu na lineární.

Všechny uvedené vztahy předpokládají splnění podmínky linearity výpočetní operace, kterou filtr provádí. Lineární operace počítají hodnotu ve výstupním obraze $g(x,y)$ jako lineární kombinaci hodnot vstupního obrazu *f* v malém okolí reprezentativního pixelu *(x, y)*. Předpoklad linearity jakéhokoli obrazového filtru je ale narušen díky tomu, že hodnota obrazové funkce (jas, intenzita) je nezáporná a omezená a předpoklad prostorové invariantnosti impulsní odezvy taktéž, protože obrazy jsou vždy ohraničeny v prostoru a tak předpoklad prostorové invariantnosti platí jen pro omezené posuny konvolučních masek. Z důvodu neexistence jednoduché teorie, která by popisovala zmíněné potíže, se používá, je-li to možné, lineární přístup a dodatečně se zajišťuje, aby na okrajích obrazu nebo v mezích rozsahu hodnot obrazové funkce nevznikaly problémy. Na impulsní odezvu obrazových filtrů se proto někdy pohlíží jako na jakousi obecnou okénkovou funkci. Filtrace je potom nazývána operací pohyblivého okénka (anglicky moving window operations).

2.2 Diskrétní aproximace gradientu a Laplaciánu (Sobelův a Laplaceův operátor), detekce hran

Matematickým nástrojem pro studium změn funkce dvou proměnných (obrazová funkce) jsou parciální derivace. Změnu funkce udává její gradient - vektorová veličina ∇, určující směr největšího růstu funkce (směr vektoru gradientu) a strmost tohoto růstu (modul gradientu).
Pro spojitou funkci *f(x,y)* je gradient definován jako:

$$\nabla f = \frac{\partial f}{\partial x}\vec{i} + \frac{\partial f}{\partial y}\vec{j}\,, \qquad (2.2.1)$$

kde $\vec{i}$ a $\vec{j}$ jsou jednotkové vektory ve směru osy x a osy y. Směr (úhel) gradientu je možné vypočítat ze známých hodnot x-ové a y-ové složky vektoru gradientu. Pro modul (velikost) gradientu platí vztah:

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \qquad (2.2.2)$$

Pixely s velkým modulem gradientu $|\nabla f(x,y)|$ odpovídají hraně v obrázku. Směr hrany, která je v obrázku tvořena více hranovými pixely, je kolmý na směr gradientu v místě pixelu.
Pro odhad velikosti gradientu se používá všesměrový lineární Laplaceův operátor – Laplacián ∇^2, který vychází z druhých parciálních derivací:

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}\,. \qquad (2.2.3)$$

Jak je patrné ze vztahu, Laplacián udává pouze velikost druhé derivace funkce více proměnných, nikoli směr. V digitalizovaném obraze se aproximují parciální derivace ze vztahu (2.2.1) diferencemi:

$$\Delta_x g(x,y) = g(x,y) - g(x-n,y),$$
$$\Delta_y g(x,y) = g(x,y) - g(x,y-n),$$

kde n je malé celé číslo, obvykle 1. Diferenci lze vyjádřit i symetricky:

$$\Delta_x g(x,y) = g(x+n,y) - g(x-n,y),$$
$$\Delta_y g(x,y) = g(x,y+n) - g(x,y-n)\,.$$

Jednorozměrné konvoluční masky, kterými je možné dosáhnout výpočtu výše uvedených vztahů mají podobu:

$$mask_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, \qquad mask_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}.$$

Aproximace gradientu ve směru osy x a y:

$$g_x(x,y) = mask_x * f(x,y) \qquad g_y(x,y) = mask_y * f(x,y).$$

Konvoluční masky, které aproximují gradient obrazové funkce mají de facto funkci operátorů a z tohoto důvodu se tak i často nazývají. Operátorem, který myšlenkově vychází z výše uvedených masek se nazývá operátor Prewittové. Tento operátor odhaduje gradient v okolí 3 × 3 pro osm směrů. Pro zjištění gradientu v požadovaném směru se musí zvolit odpovídající maska:

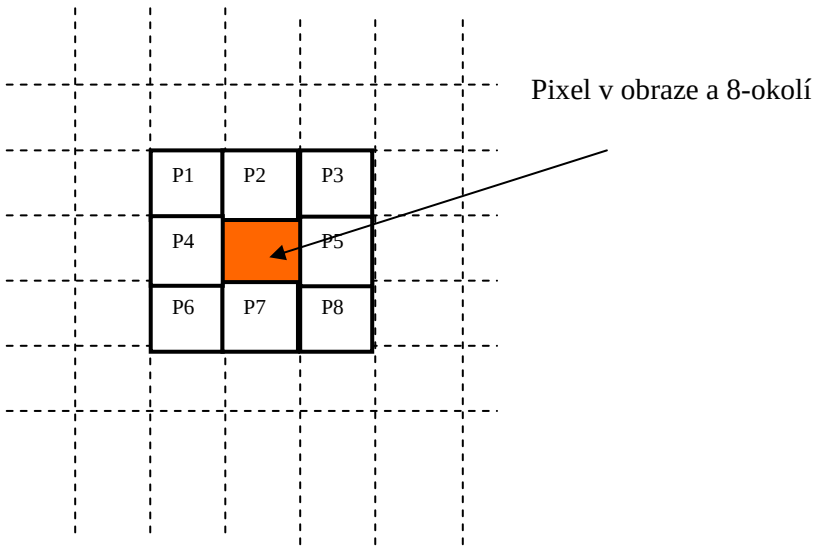
$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \qquad \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, \qquad \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}.$$

Uvedené masky odpovídají pouze třem směrům, zbývající je možné získat pootočením. Velmi podobným operátorem je Sobelův operátor, ale s tím rozdílem, že hodnotám na přímce, jejíž směr je stejný se směrem odhadovaného gradientu a která prochází středem masky, je přiřazena vyšší váha než hodnotám okolních pixelů. Masky Sobelova operátoru pro stejné směry jako masky operátoru Prewittové mají podobu:

$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \quad h_2 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}, \quad h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}.$$

Obrázek 4 Konvoluční masky Sobelova operátoru.

Pro lepší názornost následuje výpočet gradientu pomocí Sobelova operátoru pro osmiokolí ilustrované na následujícím obrázku:



Výpočet složky vektoru gradientu ve směru osy x (G_x) se provede pomocí masky h_3 a ve směru osy y (G_y) pomocí masky h_1:

G_x = P3 + 2*P5 + P8 - P1 - 2*P4 - P6

G_y = P1 + 2*P2 + P3 - P6 - 2*P7 - P8

S využitím těchto hodnot může být dále spočtena absolutní hodnota gradientu. Sobelův operátor se často v praxi používá pro detekci vodorovných a svislých hran v obraze, na což postačují právě masky h_1 a h_3. Jeho výhodou je relativně malá citlivost na šum. Typické masky Sobelova operátoru pro větší okolí:

$$\begin{bmatrix} -1/4 & -1/3 & 0 & 1/3 & 1/4 \\ -1/3 & -1/2 & 0 & 1/2 & 1/3 \\ -1/2 & -1 & 0 & 1 & 1/2 \\ -1/3 & -1/2 & 0 & 1/2 & 1/3 \\ -1/4 & -1/3 & 0 & 1/3 & 1/4 \end{bmatrix}, \qquad \begin{bmatrix} -1/4 & -1/3 & -1/2 & -1/3 & -1/4 \\ -1/3 & -1/2 & -1 & -1/2 & -1/3 \\ 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/2 & 1 & 1/2 & 1/3 \\ 1/4 & 1/3 & 1/2 & 1/3 & 1/4 \end{bmatrix}.$$

Laplaceův operátor je možné aproximovat také, podobně jako tomu bylo v případě gradientu. Níže je odvozen vztah pro výpočet Laplaciánu (druhé derivace) v nějakém bodě diskrétního signálu v závislosti na hodnotách signálu:

Spojitý	Diskrétní
$f(x)$	$v(i)$
$f'(\mathrm{x})$	$v'(\mathrm{i}) = \mathrm{v(i) - v(i-1)}$
	$v''(\mathrm{i}) = \mathrm{v'(i) - v'(i-1)}$
$f''(\mathrm{x}) = \nabla^2 f(x)$	$= [\mathrm{v(i) - v(i-1)}] - [v(i-1) - v(i-2)]$
	$= v(i-2) - 2v(i-1) + v(i)$

Ze vztahu je patrná podoba konvoluční masky v jednom rozměru - (1 -2 1). Typicky se ale používá s opačným znaménkem (-1 2 -1). Při rozšíření této masky do dvou rozměrů je na výběr ze dvou standardních masek:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Obrázek 5 Masky pro aproximaci Laplaceova operátoru pro čtyřokolí a osmiokolí.

Obě masky jsou směrově nezávislé, neboť i Laplacián neudává směr druhé derivace, ale pouze její velikost. Nevýhodou při použití osmiokolí je to, že vzdálenosti úhlopříčných pixelů jsou větší než vzdálenosti svislých a vodorovných pixelů. Nevýhodou čtyřokolí je omezení na pouze čtyři směry. Někdy se používá Laplaceův operátor s rozlišením vah pixelů, které jsou blíže nebo dále od prostředního bodu masky:

$$\begin{bmatrix} 2 & -1 & 2 \\ -1 & -4 & -1 \\ 2 & -1 & 2 \end{bmatrix}, \qquad \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}.$$

Obrázek 6 Alternativní masky Laplaceova operátoru.

V tomto případě se ztrácí invariantnost vůči pootočení masky. Hlavní nevýhodou Laplaceova operátoru je velká citlivost na šum a dvojité odezvy na hrany odpovídající tenkým liniím v obraze.

Další známé operátory aproximující gradient a sloužící k detekci hran, jejichž funkce nebude detailněji vysvětlena, jsou např. Robinsonův operátor a Kirschův operátor.
Společným problémem všech již popsaných operátorů je velká závislost jejich chování na konkrétním obrázku. Velikost masky musí odpovídat velikosti detailů v obrázku. Jejich časová složitost je v závislosti na počtu zpracovávaných prvků okolí lineární. Vzhledem k tomu, že počet zpracovávaných prvků okolí je roven relativně malé hodnotě (nejčastěji osm), nepatří tyto filtry kvůli uvedeným závislostem k těm nejproblematičtějším, co se týká doby zpracování.

Na bázi aproximace gradientu a Laplaciánu je založena i funkce řady dalších filtrů, které je možné zařadit do dvou skupin. Do jedné patří operátory založené na hledání hran v místech, kde druhá derivace obrazové funkce prochází nulou. Příkladem je Marrův - Hildrethové operátor a Cannyho hranový detektor. Do druhé náleží operátory, které „se snaží“ lokálně aproximovat obrazovou funkci parametrickým modelem, např. polynomem dvou proměnných.

2.3 Lokální vyhlazování obrazu

Řešení problému vyhlazování obrazu je relativně snadné, je-li k dispozici několik obrazů téže předlohy, které se liší pouze šumem. Potom je účinné „průměrovat“ hodnotu pixelu o stejných souřadnicích přes více obrázků. Je-li k dispozici jen jediný obraz téže předlohy, potom se musíme spolehnout na analýzu hodnot jasu ve vybraném okolí obrazového elementu. Sousední pixely mají převážně tutéž nebo blízkou hodnotu jasu. Hodnota reprezentativního pixelu je nahrazena hodnotou typického reprezentanta mezi hodnotami v okolí nebo kombinací několika hodnot.

Lineární metody vyhlazování vypočítávají novou hodnotu reprezentativního pixelu jako lineární kombinaci hodnot ve zkoumaném okolí. Pro digitální snímky lze v prostorové oblasti lineární kombinaci vyjádřit jako diskrétní konvoluci. Tyto filtry mohou být opět popsány jejich konvolučními maskami, které se liší váhami pro jednotlivé položky masky.

Základní metodou vyhlazování obrazu je obyčejné průměrování. Každému bodu je přiřazen nový jas, který je aritmetickým průměrem původních jasů ve zvoleném okolí. Odpovídající konvoluční maska pro okolí 3×3 je:

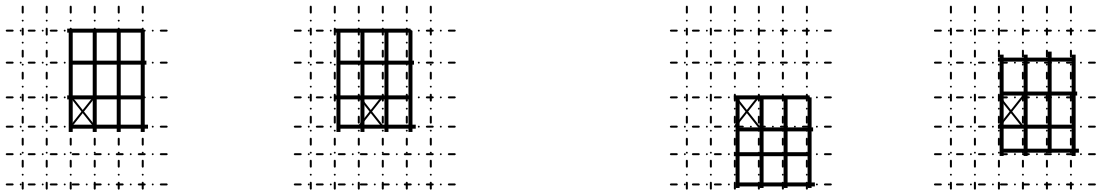
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Někdy se zvětšuje váha středového bodu nebo jeho 4-sousedů. Takové masky pro okolí 3×3 potom vypadají:

$$\frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \qquad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Maska vlevo je upravena pro průměrování s Gaussovským rozložením. Základní nevýhodou praktického použití obyčejného průměrování je rozmazávání hran v obraze. Tohoto efektu se ale někdy v počítačové grafice záměrně využívá. Tyto filtry lze jednoduše realizovat jako separabilní filtry.

Potíže s rozmazáváním hran částečně eliminují **nelineární filtrační metody**. Často jsou založeny na principu nalezení té části analyzovaného okolí (oblast o zhruba konstantním jasu), do které reprezentativní bod patří. Jen pixely této oblasti se využijí pro hledání jasové hodnoty (např. aritmetickým průměrem nebo výběrem hodnoty jasu jednoho pixelu), která bude reprezentovat celé okolí ve výstupním obrázku. To neplatí o mediánovém filtru, který pracuje na základě metod robustní statistiky a bude podrobněji vysvětlen dále. Jako příklad třídy metod vyhlazování, které používají při filtraci výběr okolí, uveďme metodu rotující masky. Kolem reprezentativního bodu rotuje malá maska, v našem případě je na obrázku 7 maskou čtverec 3×3. Možných poloh masky je 8. V každé masce se spočítá rozptyl jasů. Za homogenní okolí reprezentativního bodu se vybere maska s nejmenším rozptylem. Nová hodnota reprezentativního bodu může být dána aritmetickým průměrem hodnot ve vybrané masce. Metoda nerozmazává hrany obrazu, má dokonce mírně ostřící charakter.

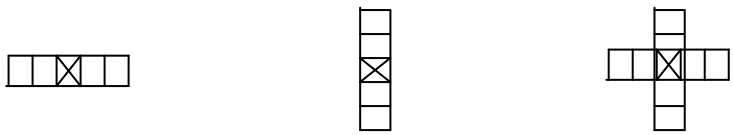


Obrázek 7 Ukázka čtvercových rotujících masek.

Nyní bude popsán princip činnosti lineárních filtrů, jejichž původ se nachází v robustní statistice. Upozorněme na hlavní potíž lineárního odhadu správné hodnoty z jejích zašuměných realizací. Pro lepší názornost si představme, že odhadujeme správný jas z instancí v okně rozměru 3×3. Nechť se osm hodnot liší jen málo od správné hodnoty a jedna hodnota je (např. díky hrubé chybě při měření) vychýlená, řekněme blížící se k ∞ (záměrně přehnané). Aritmetický průměr, tedy lineární odhad, vyjde také ∞ a bude vychýlený. Tento problém je v matematice vlastní všem lineárním odhadům, například i často používané aproximaci pomocí metody nejmenších čtverců.

Základem metod robustní statistiky je snaha najít v náhodné populaci vychýlené hodnoty, vyloučit je z uvažování a zbylé hodnoty nahradit nějakou typickou hodnotou. Ve statistice ztělesňují myšlenku robustních statistik výběrové kvantily. Jejich nejjednodušším a ve zpracování obrazů nejčastěji používaným případem je medián M. Nechť x je náhodnou veličinou. Medián M je hodnota, pro kterou je pravděpodobnost jevu $x < M$ rovna jedné polovině.

Metoda filtrace mediánem stanoví jas výsledného bodu jako medián určený z hodnot jasu bodů v lokálním okolí vstupního obrazu. I přes svou konceptuální jednoduchost je ale „nepříjemná“ pro implementaci (viz dále). Metoda redukuje stupeň rozmazání hran a relativně dobře potlačuje impulsní šum. Její hlavní nevýhodou je, že porušuje tenké čáry a ostré rohy v obraze. Tento nedostatek se dá obejít, když se použije jiné než „klasické“ čtvercové okolí:



Mezi další vyhlazující lokální nelineární filtry patří např. Sigma filtr. Z filtrů které využívají i globální informace o obraze uveďme např. filtr založený na algoritmu „Superspike“.

2.4 Algoritmy pro nalezení mediánu

Metoda řešení, která je často doporučována v odborné literatuře pojednávající o obrazových filtrech, je seřazení posloupnosti hodnot, v níž je medián hledán. Pokud je posloupnost tvořena lichým počtem prvků, mediánem je po seřazení prvek uprostřed posloupnosti. Optimální sekvenční algoritmy řazení, jako je např. Quick Sort, mají polynomiální ($O(n^2)$) nejhorší časovou asymptotickou složitost, což je řád složitosti, který určitě nepatří mezi nejpříznivější. Průměrná časová složitost algoritmu Quick Sort je linearitmická. Rychlejším řešením pro nalezení mediánu je použití algoritmu nazvaného Find, který najde v průměrném čase $O(n)$ obecně i-tý nejmenší prvek v množině o n prvcích. Nejhorší časová složitost je ale také řádu n^2. Při jeho spuštění s indexem $i = n/2$ bude nalezen medián. Důkaz časové složitost je možné nalézt v [3]. Rekurzivní algoritmus je v „pseudoprogramovacím“ jazyce uveden níže. Jelikož je známým faktem, že rekurzivní verze algoritmů nedosahuje takové efektivity jako jejich verze využívající cykly, bylo by vhodné použít při implementaci verzi nerekurzivní.

```
function Find(M:TSet; i:TIndex):TPrvek;
begin
    s :=Nějaký prvek z(M);
  M1 := {x ∈ M | x < s};
  M2 := {x ∈ M | x = s};
   ( Mohou-li být v M prvky s větší násobností )
  M3 := {x ∈ M | x > s};
  if |M1| + |M2| < i then
       return (Find(M3, i - |M1|- |M2| ))
  else if |M1| < i then
          return (s)
    else
          return (Find(M1, i))
    end
end
```

Časová složitost algoritmu závisí na pravděpodobnostním rozložení vstupních hodnot a s tím spojené úspěšnosti počátečního nastavení hodnoty pivotu (proměnná *s*). Nejlepších výsledků by měl algoritmus dosahovat při filtraci snímků, ve kterých se vyskytuje velké množství okolí obsahujících samé stejné hodnoty. Pokud ale bude vstupní snímek naopak vykazovat vysokou hustotu hran, lze naopak očekávat přiblížení se k polynomiálnímu řádu složitosti

Pro nalezení i-tého největšího prvku posloupnosti existuje ještě známý algoritmus Select, jehož **nejhorší** časová složitost spadá do řádu $O(n)$, ale tento algoritmus je výrazně komplikovanější a jeho standardní verze je dimenzovaná pro práci s posloupnostmi řádově větších délek, než jakých nabývá posloupnost tvořená prvky okolí pixelu.

V našem případě má posloupnost poměrně malou délku, na které se asi druh závislosti počtu kroků algoritmu na počtu vstupů výrazně neprojeví a znatelný bude vliv trvání běhu algoritmů pro případ posloupnosti, která obsahuje jen jeden prvek (komplikovanost, resp. délka programu, kterým je implementován algoritmus). To závisí na přesných vztazích pro časovou složitost u všech algoritmů, především na velikostech hodnot jednotlivých koeficientů v těchto vztazích. Výsledek může být ovlivněn i nerovnoměrným pravděpodobnostním rozložením na množině vstupních hodnot a lišit se v závislosti na vlastnostech obrázků. Nejlepšího kandidáta z uvedených algoritmů by bylo tedy vhodné zjistit experimentálně.

2.5 Způsoby ošetření nelineárních vlastností obrazového signálu

Nejprve bude zaměřena pozornost na ošetření komplikace způsobené nezáporností obrazové funkce. Problém lze vyřešit tak, že se bude vracet absolutní hodnota výsledku vypočteného pomocí Sobelova, resp. Laplaceova operátoru. V případě Sobelova operátoru se tím ztratí informace o směru vektoru, který vznikne pravoúhlým promítnutím vektoru (celkového) gradientu do přímky, která leží v rovině obrazu a jejíž směr je kolmý na směr hledaných hran, což ale při typické aplikaci Sobelova operátoru, kdy je zapotřebí zjistit pouze modul vektoru gradientu, nevadí. Z hlediska detekce hran přijdeme o informaci, jestli jsme nalezli hranu při přechodu z oblasti s vyšším jasem do oblasti s nižším jasem nebo naopak. Masky Sobelova operátoru je možné uspořádat do dvojic tak, že první z dvojice bude rovna druhé vynásobené číslem -1 (maska, která by byla ve dvojici s maskou h_1 by obsahovala záporné hodnoty na horním řádku masky a kladné na nejnižším). Vždy jedna dvojice může být použita pro zjištění hrany buď ve vodorovném, svislém nebo zbývajících dvou „šikmých směrech“. V případě hledání hran ve vodorovném směru určí znaménko výsledku, zda vektor gradientu směřuje Severním nebo Jižním směrem (např. pro masku h_1 pro detekci hran v horizontálním směru určuje kladná hodnota Severní směr a záporná Jižní směr a pro druhou masku pro horizontální směr, která by vznikla posunutím o čtyři místa nebo vynásobením -1, určuje kladná hodnota Jih a záporná Sever). Hodnota absolutní hodnoty výsledku při použití nějaké masky Sobelova operátoru je rovna hodnotě, která by byla získaná při výpočtu, v němž se použije vždy ta maska z dvojice, do níž maska patří, která vrátí nezáporný výsledek.

Další výpočetní potíží, která nastává vlivem omezenosti obrazové funke, je přetečení vymezeného číselného rozsahu. Výskyt tohoto případu je nutné detekovat a v případě, že je aproximovaná hodnota větší než maximální hodnota z přípustného intervalu, vrátí se místo vypočtené hodnoty tato maximální hodnota.

Posledním problémem je prostorová omezenost obrazového signálu. V okrajových oblastech snímku nelze při výpočtu diskrétní konvoluce vycházet z osmiokolí reprezentativního pixelu. Konvoluční maska se po položení jejího prostředního prvku na pozici reprezentativního pixelu dostane některou částí mimo definované hodnoty vstupního obrazu. V těchto oblastech je tedy nutné použít pomocný výpočet, který zachová způsob výpočtu daného filtru, ale využije jen dostupné hodnoty vstupního obrazu. Konvoluční masky nebo v tomto případě spíše „násobící okénka“, která je možné užít pro aproximaci Sobelova operátoru pro detekci vodorovných hran, resp. Laplaceova operátoru pro osmiokolí, jsou ilustrovány na obr. 8, resp. 9. Prvek okénka, který se bude překrývat s reprezentativním pixelem, je od ostatních odlišen oranžovou barvou pozadí. Okénka pro řádky a sloupce se aplikují na všechny body v těchto oblastech kromě rohových bodů.

-2	-1
2	1

Okénko pro levý horní rohový bod snímku.

-2	-1
2	1

Okénko pro levý dolní rohový bod snímku.

-1	-2
1	2

Okénko pro pravý horní rohový bod snímku.

-1	-2
1	2

Okénko pro pravý dolní rohový bod snímku.

-1	-2	-1
1	2	1

Okénko pro body horního řádku snímku.

-1	-2	-1
1	2	1

Okénko pro body dolního řádku snímku.

-2	-1
0	0
2	1

Okénko pro body levého sloupce snímku.

-1	-2
0	0
1	2

Okénko pro body pravého sloupce snímku.

Obrázek 8 Okénka, která lze užít pro aproximaci Sobelova operátoru v okrajových oblastech snímku.

-3	1
1	1

Okénko pro levý horní rohový bod snímku.

1	1
-3	1

Okénko pro levý dolní rohový bod snímku.

1	-3
1	1

Okénko pro pravý horní rohový bod snímku.

1	1
1	-3

Okénko pro pravý dolní rohový bod snímku.

1	-5	1
1	1	1

Okénko pro body horního řádku snímku.

1	1	1
1	-5	1

Okénko pro body dolního řádku snímku.

1	1
-5	1
1	1

Okénko pro body levého sloupce snímku.

1	1
1	-5
1	1

Okénko pro body pravého sloupce snímku.

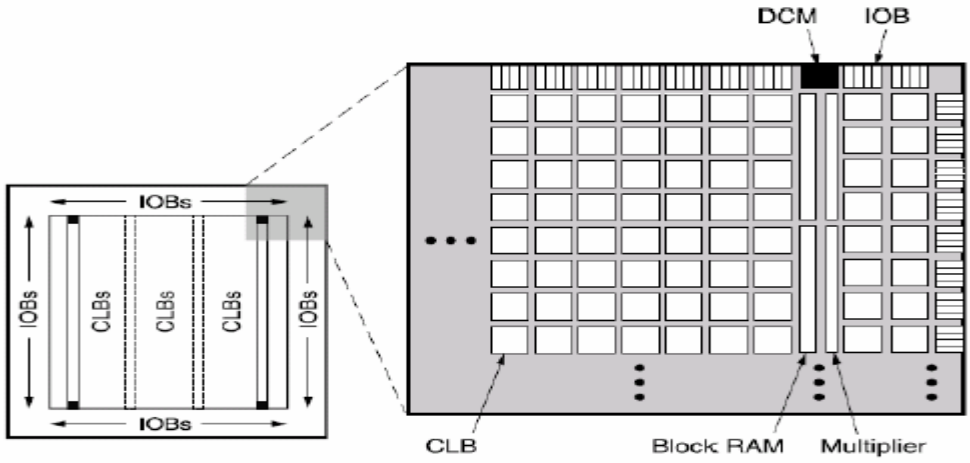
Obrázek 9 Okénka, která lze užít pro aproximaci Laplaceova operátoru v okrajových oblastech snímku.

2.6 Technologie FPGA

Obecně se jedná o pokročilý reprogramovatelný hardware fungující na bázi hradlových polí, jenž lze, podobně jako software na počítači, neomezeně modifikovat pro různé účely dle potřeby – uživatel tedy nemusí vytvářet nový hardware pro každou aplikaci znovu. Součástí výbavy přípravku FITkit je FPGA čip Spartan3 od firmy Xilinx. Jeho architektura se skládá z pěti základních mechanismů:

- CLB (Configuration logic block) – matice konfigurovatelných logických bloků
- Blokové paměti RAM (BRAM)
- Vestavěné násobičky
- DCM – správa hodinových impulsů
- IOB architektura

Jednotlivé prvky jsou ilustrovány na následujícím obrázku:

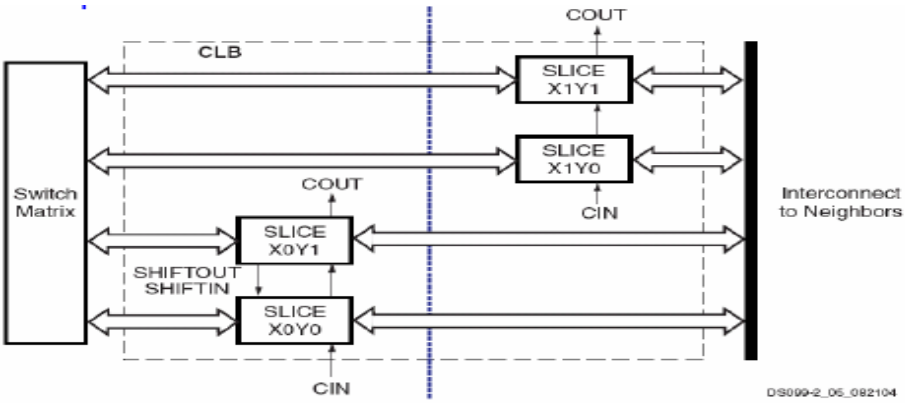


Obrázek 10 Schematický nákres architektury FPGA čipu Spartan 3.

<u>Konfigurovatelný blok logiky (CLB) obsahuje:</u>

- 4 tzv. SLICE komponenty
- 2 nezávislé „carry“ řetězce pro konstrukci rychlých sčítaček

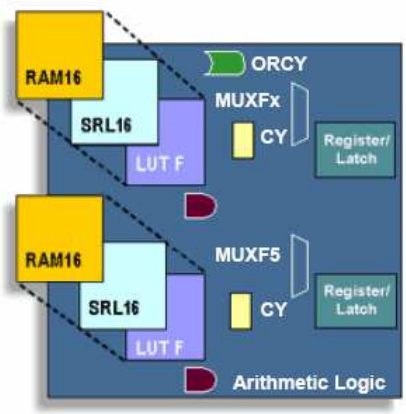
Sousední SLICE v CLB bloku mohou komunikovat prostřednictvím rychlého propojení. Dále jsou všechny připojeny ke globální propojovací matici.



Obrázek 11 Vnitřní architektura CLB bloku

Architektura SLICE obsahuje:

- 2 funkční generátory (FG)
- 2 registry (Latch)
- Multiplexory MUXF5, MUXFx
- “Carry” logika
- Pomocná logika pro aritmetiku

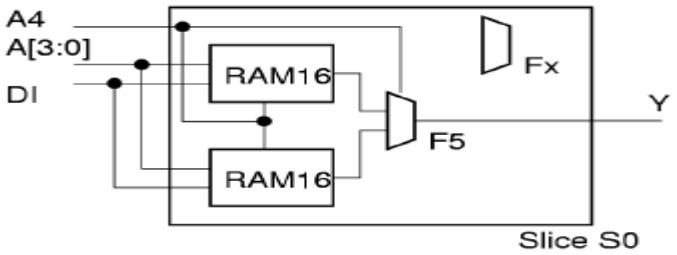


Obrázek 12 Komponenty tvořící Slice

Součástí čipu Spartan3 na FITkitu je 192 CLB, které tedy zahrnují 768 SLICE komponent, které dále zahrnují 1536 funkčních generátorů a registrů.

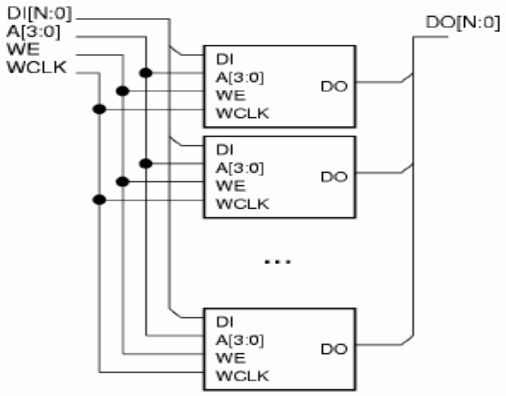
Funkční generátor

Funkční generátor je realizovaný jako vyhledávací tabulka (anglicky look up table - LUT). Jedná se o logické hradlo se 4 vstupy a jedním výstupem, které umožňuje realizovat libovolnou binární funkci čtyř proměnných. S pomocí multiplexorů lze z LUT jednoduše vytvářet složitější funkce. Vzhledem ke konfigurovatelnosti LUT a vlastnosti, že každé kombinaci hodnot 0 a 1 na vstupu může být přiřazena hodnota 0 nebo 1 na výstupu, je možné LUT využít i jako paměť o velikosti 16×1bit. Pomocí multiplexorů lze opět vytvořit i větší paměť, např. 32×1bit.



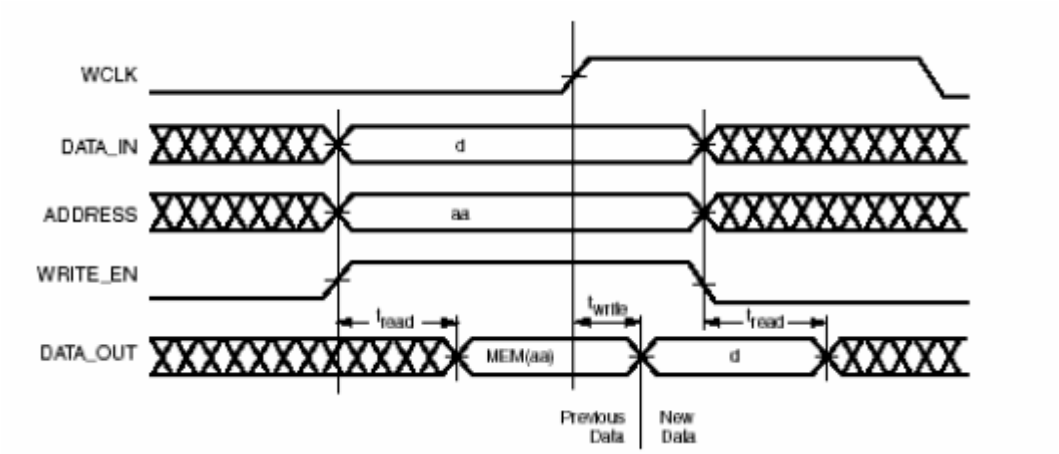
Obrázek 13 Vytvoření paměti 32×1bit ve Slice.

Paměti lze paralelně řadit vedle sebe a tvořit paměti o libovolné šířce dat:



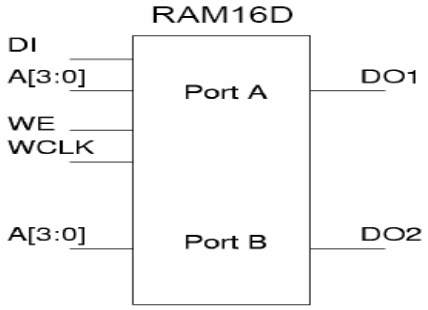
Obrázek 14 Způsob vytvoření paměti o větší bitové šířce.

Čtení paměti je asynchronní. To znamená, že po vystavení adresy jsou na výstup vystavena data a není potřeba čekat na hodinový signál. Zápis je synchronní a data jsou do paměti zapsána na vzestupnou hranu hodinového signálu WCLK.



Obrázek 15 Synchronizace paměťových signálů při zápisu.

Pro konstrukci FIFO (first in first out - fronta) pamětí, které se u hardwarových filtrů užívají, je vhodná dvou - portová paměť Ram16D, která je vytvořena ze dvou funkčních generátorů. Port A umožňuje provést operace asynchronního čtení a synchronního zápisu. Pomocí portu B lze provést asynchronní čtení. S využitím funkčních generátorů je dále možné vytvořit i posuvný registr, zpožďovací obvody, generátory náhodných čísel (LFSR), čítače libovolných sekvencí, apod.

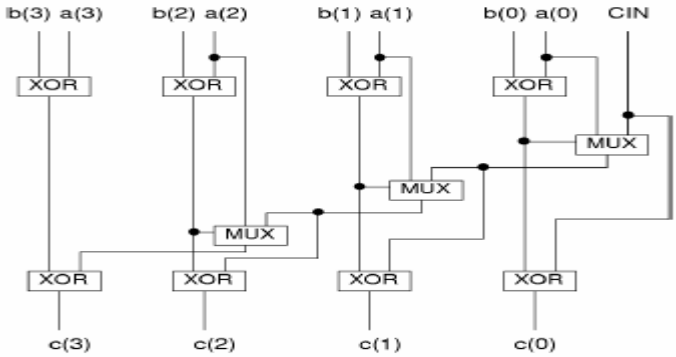


Carry logika

Je v každém SLICE tvořena:

- 2 multiplexory (MUXCY)
- 2 hradly xor (XORCY)
- 1 hradlem or (ORY)

Používá se pro konstrukci rychlých sčítaček, čítačů, komparátorů apod. Nyní bude uveden příklad použití pro sčítačku, která by pro implementaci některých filtrů jistě mohla přijít vhod:

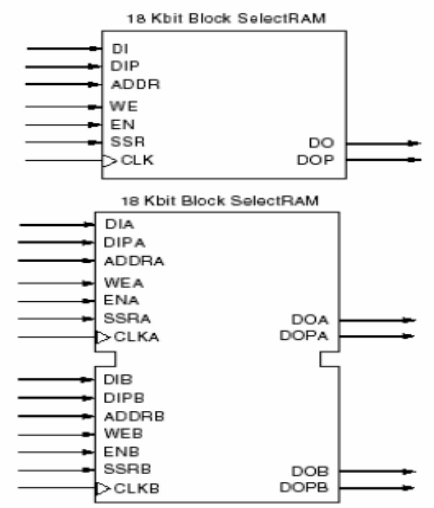


Obrázek 16 4-bitová sčítačka vytvořená s využitím „carry logiky“ a LUT.

Sčítačka je pouze 4-bitová, ale pro ilustraci postačuje. První XOR je implementován v LUT. Přenos přenosového příznaku je implementován v MUXCY. Výpočet koncového výsledku na základě předběžného výsledku a přenosu je implementován v XORCY. Implementace vyžaduje dvě SLICE komponenty.

Block Select RAM

Jedná se o paměťový blok o velikosti 18Kb. Přístup k datům při čtení i při zápisu je synchronní, data jsou dostupná v následujícím hodinovém taktu. Využitelná je jedno - portová i dvou - portová varianta. Pro každý port lze nastavit jednu z následujících datových šířek:

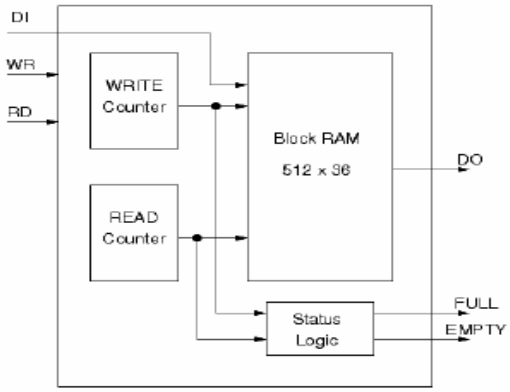


Obrázek 17 Jedno - portové a dvou - portové rozhraní BRAM

- 16k x 1 bit 2k x 9 bits
- 8k x 2 bits 1k x 18 bits
- 4k x 4 bits 516 x 36 bits

Paměť je vhodná pro uložení většího objemu dat, např. jako paměť instrukcí pro procesor, dále při konstrukci FIFO pamětí, FSM obvodů, apod. Varianta čipu Spartan3 použitá na FITkitu obsahuje 4 BRAM.

Obrázek 18 zachycuje schéma naznačující využití BRAM jako FIFO paměti. Nyní bude následovat podrobnější popis vysvětlující realizaci FIFO. Je zřejmé, že je třeba mít uložený aktuální začátek a konec fronty, která bude uložena v BRAM. K BRAM se tedy musí přidat jednak registr pro uložení adresy prvního prvku fronty (sloužící jako čtecí ukazovátko) a také registr (zápisové ukazovátko) pro uložení paměťového místa, jehož adresová hodnota je o jedna menší než adresa posledního prvku fronty (adresa pro zápis nového prvku), je-li zvolen směr růstu fronty v paměťovém prostoru směrem od vyšších k nižším hodnotám adres. Aby bylo možné uskutečňovat zároveň zápis i čtení, je nutné, aby byly výstupy registrů připojeny každý k jinému portu BRAM. Po přečtení hodnoty prvního prvku ve frontě se musí stát prvním prvkem bezprostředně následující prvek. Toho je docíleno aktualizací hodnoty čtecího registru dekrementací o jedna v rozsahu paměti. V případě, že je současnou hodnotou adresy uložené v registru 0, bude následující adresou nejvyšší hodnota adresy z paměťového rozsahu. Při zápisu nastává obdobná situace, která je vyřešena analogicky. Dále je třeba detekovat stavy plné (hodnota čtecího ukazovátka +1 se rovná hodnotě zápisového ukazovátka) a prázdné fronty (hodnoty čtecího i zápisového ukazovátka jsou stejné). Samozřejmě je možné paměť implementovat i tak, aby fronta rostla směrem nahoru (použití inkrementace místo dekrementace, modifikace vztahu hodnot čtecího a zápisového registru určujícího plnou frontu). Komponenta BRAM může být instanciována ve VHDL.



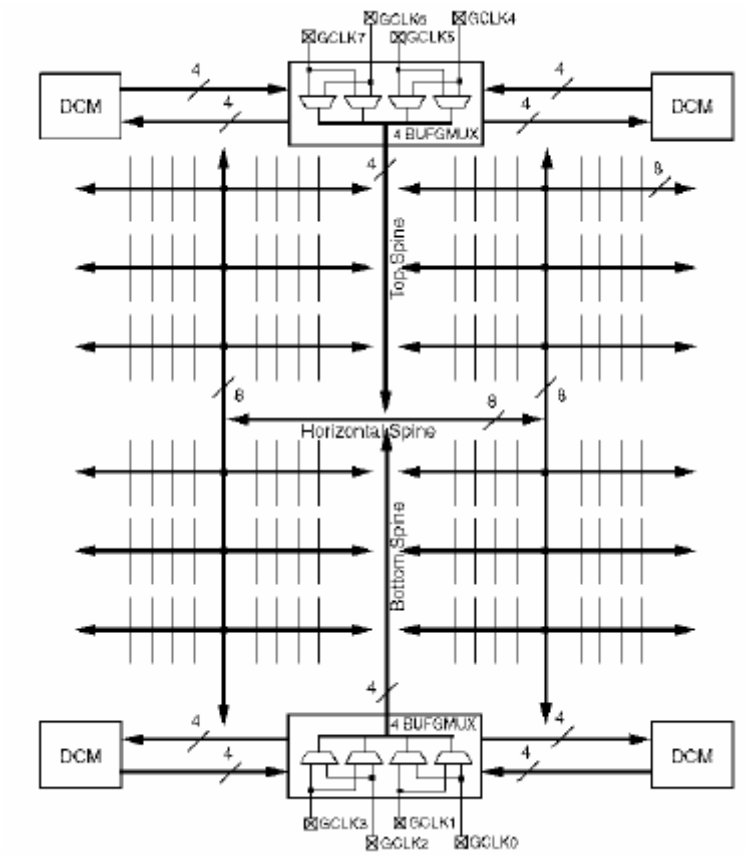
Obrázek 18 Nástin způsobu využití BRAM při konstrukci FIFO

Vestavěné násobičky

Vestavěná násobička ušetří velké množství logiky a dosahuje vysoké rychlosti výpočtu. Délka vstupních operandů je18 x 18 bitů a 36 bitů pro výstupní operand. Podporuje znaménkové i neznaménkové operandy. Pro vytvoření větších násobiček lze kaskádně propojit více kusů. Vhodná je pro DSP aplikace (operace konvoluce, multiple and acumulate). Čip na FITkitu obsahuje 4 násobičky.

Globální rozvod hodinového signálu

Hodinový signál je přiváděn do FPGA skrze GCLK piny a je rozváděn rovnoměrně po celém čipu pomocí speciálních hodinových rozvodů (rychlejší než běžné vodiče). Signál vstupuje přes BUFG/BUFGMUX komponenty, které odstraní případné zkreslení a zákmity. Hodiny lze upravovat pomocí speciálních DCM obvodů (korekce zpoždění, násobení, dělení, fázový posun).



Obrázek 19 Rozvod hodinového signálu

Input/Output Block (IOB)

Každý pin FPGA může být konfigurován jako vstup, výstup nebo obojí. Jsou podporovány jednotlivé vodiče i diferenciální páry (pouze u dvou sousedních pinů). Je podporováno velké množství standardů (LVTTL, LVCMOS, PCI33, PCI66, PCI-X, GTL, HSTL, STTL, AGP, LVDS, LDT, LVPECL).

2.7 Výukový přípravek FITkit

Platforma FITkit je výukovým přípravkem Fakulty informačních technologií vhodným zejména pro implementaci vestavěných systémů. Následuje popis architektury přípravku.

2.7.1 Komponenty a způsoby užití FITkitu

Součásti kitu:

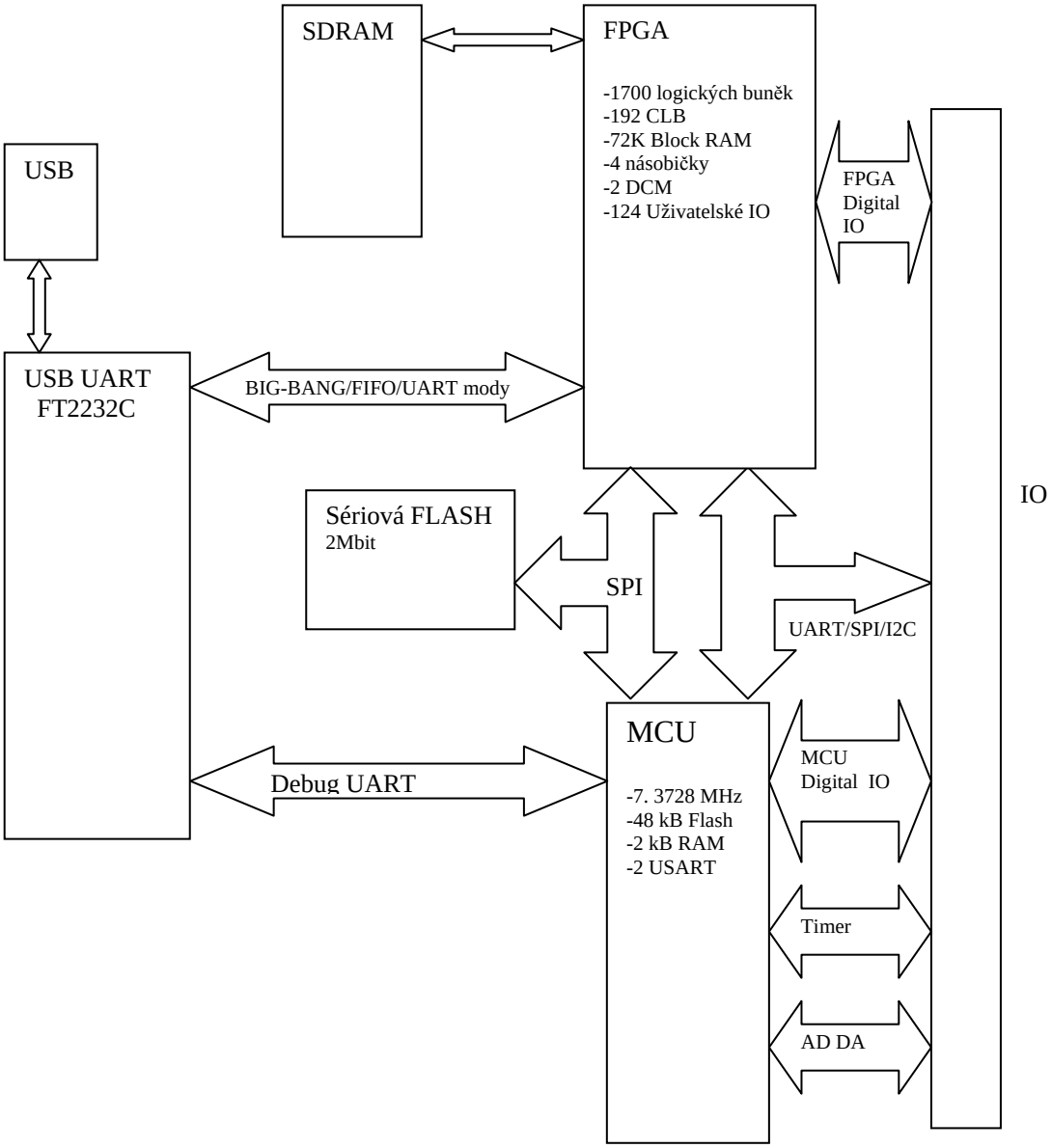
- čip FPGA : Spartan3 XC3S50-4PQ208C (Xilinx)
- MCU : MSP430F168 (Texas Instruments)
- USB-UART převodník FT2232C
- Audio IN / OUT
- konektory PS2
- konektor VGA
- konektor RS232
- DRAM 8x8mbit
- FLASH paměť
- Klávesnice
- LCD displej
- Pinheaders

Obrázek 20 Fotografie FITkitu, pohled shora.

Obě programovatelné jednotky FITkitu (MCU, FPGA) je možné programovat pomocí kabelu s rozhraním JTAG. Komunikace s kitem, resp. s MCU, je možná i přes rozhraní USB a pro nahrání konfigurace FPGA lze použít naprogramovaný MCU. Vzhledem ke skutečnosti, že USB rozhraním v této době PC běžně disponují a USB kabel je relativně lacinou záležitostí, je komunikace prostřednictvím USB dostupnějším způsobem. Pro *MCU* je k dispozici program, který umí po restartu nahrát konfiguraci uloženou ve *FLASH* paměti do *FPGA*. Dále dokáže nahrát konfiguraci pro *FPGA* do paměti *FLASH* nebo přímo do *FPGA*. Konfigurační soubor se v obou případech přenese z PC přes USB rozhraní za pomocí terminálu pro sériovou komunikaci, který podporuje protokol *XModem* v režimu *1KCRC*, jehož implementace je součástí programové výbavy FITkitu. Pomocí terminálu je umožněna obecně komunikace mezi programem běžícím na MCU a PC a tím i ovládání programu. V uživatelských aplikacích napsaných pro kit je možné využít knihovnu libmsp, která obsahuje funkce umožňující realizovat komunikaci a výměnu dat mezi MCU, FPGA, FLASH pamětí a PC, konverzi datových typů a další při programování často potřebné operace.

Při návrhu FPGA aplikací se využívá skutečnosti, že vlastnosti hardwaru se v dnešní době převážně popisují vhodným programovacím jazykem (VHDL). Popis musí splňovat určité podmínky, aby mohla být provedena jeho syntéza. Poté je možné vygenerovat binární konfigurační soubor pro danou platformu. Veškerý potřebný návrhový software je k dispozici zdarma (ModelSim, Xilinx Ise WebPack). Software pro mikrokontroler se tvoří v jazyce C a do spustitelné formy se překládá pomocí GNU překladače (mspgcc). Generování programovacích dat pro FPGA z popisu v jazyce VHDL probíhá zcela automaticky pomocí profesionálních návrhových systémů, které využívají standardní GNU Makefile soubory.

Blokové schéma znázorňující klíčové komponenty i s jejich parametry je uvedeno na následující straně. Kompletní schéma a podrobnější informace o architektuře FITkitu a práci s ním je možné nalézt na adrese http://merlin.fit.vutbr.cz/FITkit/.



Obrázek 21 Zjednodušené blokové schéma architektury FITkitu, které zachycuje převážně komponenty, kterou budou využity při implementaci filtrů. Některé komponenty FITkitu jsou vynechány.

2.7.2 Rozhraní SPI na FITkitu

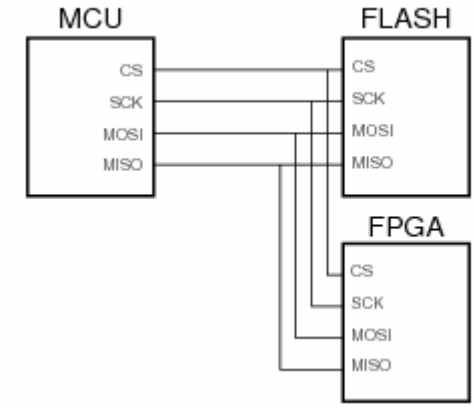
Z důvodu silného vlivu rychlosti komunikace mezi MCU a FPGA, zprostředkované tímto rozhraním, na celkovou dobu zpracování hardwarových filtrů implementovaných na FITkitu se ukázalo jako vhodné popsat zde blíže způsob, jakým funguje.

Rozhraní SPI (*Synchronous Peripheral interface*) je sériové vysokorychlostní rozhraní typu master-slave. Rozhraní umožňuje obousměrnou komunikaci řízenou tzv. master (nadřazeným) zařízením s libovolným počtem slave zařízení, které jsou k této sběrnici připojeny.

Rozhraní tvoří následující signály:

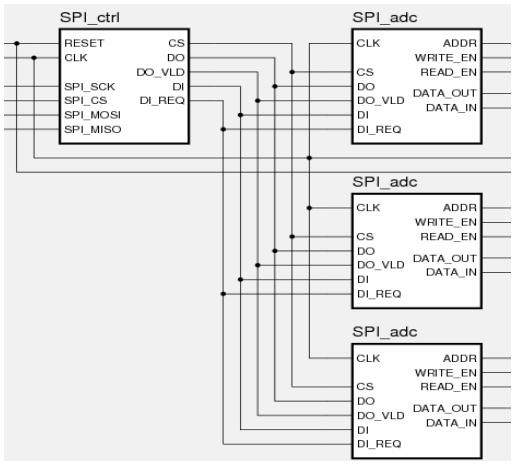
- CS (*chip select*) - signál vymezující datový rámec, aktivní v logické 0 (při přenosu). Signál generuje master.
- SCK (*clock*) - hodinový signál, který v závislosti na zvoleném režimu určuje okamžik, kdy dochází k vzorkování dat. Generuje jej master.
- MOSI (*master out slave in*) - data vystupující z master zařízení (data out).
- MISO (*master in slave out*) - data vstupující do master zařízení (data in), která generuje slave.

SPI na fitkitu umožňuje procesoru, který je nakonfigurován v master režimu, kromě komunikace s pamětí FLASH komunikovat i s periferiemi realizovanými uvnitř FPGA, např. s paměťmi BRAM, apod. Slave zařízení FLASH a FPGA sdílejí veškeré signály (obr. 22) a rozlišení adresovaného zařízení se provádí na úrovní přenášených bitů. Aby byla zajištěna bezchybná komunikace, musí mít FPGA i FLASH vzájemně disjunktní množinu operačních kódů. Rozhraní SPI je nakonfigurováno tak, že k vzorkování signálů MOSI a MISO dochází při vzestupné hraně hodinového signálu SCK, přičemž neaktivní úroveň SCK je logická 1. Frekvence hodinového signálu SCK je určena hodnotou uloženou v registrovém páru U1BR1a U1BR0. Tyto registry specifikují dělitel hodinového signálu SMCLK, který má kmitočet 7.3728MHz. Protože nejmenší možný dělitel je roven 2, je maximální možná frekvence SCK rovna 3.6864MHz, což odpovídá přenosové rychlosti 460.8 kB/s. Aby bylo možné uvnitř FPGA využívat snadno sběrnici SPI a připojit libovolné množství zařízení, byla vytvořena interní sériová sběrnice a protokol, který umožňuje komunikovat s konkrétním zařízením uvnitř FPGA. Komunikaci můžeme rozdělit na tři části:



Obrázek 22 SPI na FITkitu.

- **Odeslání operačního kódu**. Operačního kód je osmice bitů, která určuje, že komunikace se týká FPGA a nikoliv FLASH. Kromě rozlišení cíle obsahuje operační kód informace pro SPI řadič uvnitř FPGA.
- **Adresace cílového zařízení**. Datové bity určující adresu zařízení, kterého se komunikace týká. Počet bitů adresní části může být libovolný.
- **Přenosu dat**. Počet přenášených bitů je opět jako v případě adresy libovolný. Data se mohou číst i zapisovat současně. Počet datových bloků je opět neomezen.



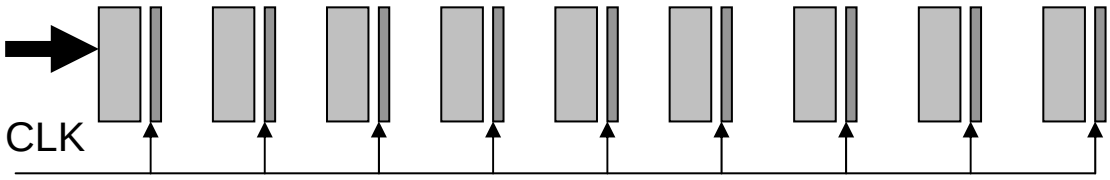
Obrázek 23 Komponenty tvořící SPI sběrnici – SPI řadič (vlevo) a SPI dekodéry (vpravo).

Sběrbice je vytvořena pomocí komponent SPI řadiče a SPI dekodéru. Základ tvoří SPI řadič, který provádí jednoduchou konverzi SPI protokolu na vnitřní sériový protokol. Vnitřní sběrnice může být obsazena libovolným počtem SPI dekodérů, jejichž úkolem je dekódovat adresu cílového zařízení a poskytnout jednoduché paralelní rozhraní umožňující obousměrnou komunikaci.

Frekvence CLK, kterou jsou synchronizovány komponenty SPI, musí být pro bezchybnou funkci alespoň 4x vyšší, než je maximální frekvence hodinového signálu SPI_SCK.

2.8 Využití řetězeného zpracování

Činnost obecných kombinačních obvodů může být rozdělena do několika na sebe navazujících fází provádějících pouze část jejich funkce nebo-li může být využito řetězené zpracování. Tento způsob implementace umožní zvýšení hodinového kmitočtu vzhledem ke kmitočtu, jehož perioda by nemohla být menší než zpoždění „nejpomalejší cesty“ celého kombinačního obvodu.



Obrázek 24 Obecné schéma řetězeného zpracování.

Vytvoří se k stupňů oddělených registry (obr. 24). Označí-li se počet datových jednotek N, zpoždění stupně t a zpoždění registru d, bude řetězením dosaženo oproti neřetězenému způsobu zrychlení dané výrazem: $N.k.t/((k+N-1)(t+d))$. Řetězení má smysl pouze když je N dostatečně velké. Pro všechny cesty vedoucí od vstupu na výstup musí být zajištěno stejné zpoždění.

3 Architektura

V této kapitole bude popsán přístup, kterým byli softwarově a hardwarově realizovány tři výše popsané typy filtrů (Sobelův, Laplaceův, mediánový) v prostředí přípravku FITkit. Bude rozkreslena architektura hardwarové části řešení zahrnující obvod řetězené linky přivádějící data a kombinační obvody jednotlivých filtrů a popsána struktura programů implementující softwarová řešení jednotlivých filtrů a softwarovou část hardwarového řešení.

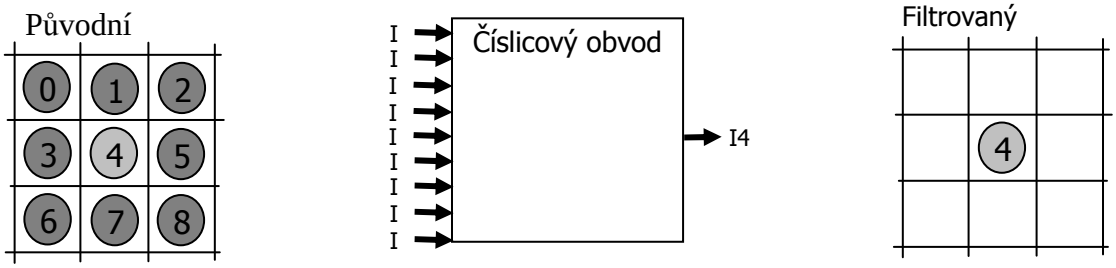
3.1 Koncepce užitá při hardwarové realizaci filtrů

V případě všech tří filtrů byl implementován hardwarový způsob výpočtu filtrace pouze pro neokrajové oblasti snímku a okrajové oblasti jsou filtrovány softwarově. Obvod, který by realizoval filtraci celého obrázku, by byl výrazně komplikovanější. Na celkovou dobu filtrace na FITkitu nemá uvedený přístup vzhledem k existenci jiných vlivů omezujících maximální dosažitelné zrychlení (SPI) a vlivem relativně malé percentuální hodnoty, určující počet okrajových prvků vzhledem k celkovému počtu prvků obrázku (2%, u malých testovacích obr. přibližně 6%), žádný dramatický účinek. Zpomalující vliv by se projevil v případě, že by hardwarové řešení filtrující téměř celou oblast obrázku a přenos vypočtených dat byly přibližně stejně rychlé jako softwarové řešení filtrující pouze okraje obrázku, což by odpovídalo zrychlení „velmi vzdálenému“ naměřenému zrychlení.

V rámci návrhu hardwarové akcelerace bude nejvíce prostoru a úsilí věnováno fázi filtrace, která se týká výpočtu výsledné hodnoty pixelu ze vstupních hodnot okolí pixelu. Bude tedy proveden návrh obvodů umožňující dosažení efektivního způsobu uskutečnění výpočtů, na základě kterých jednotlivé filtry fungují. Nastíněny budou i další možnosti hardwarové akcelerace, které ale nebudou implementovány.

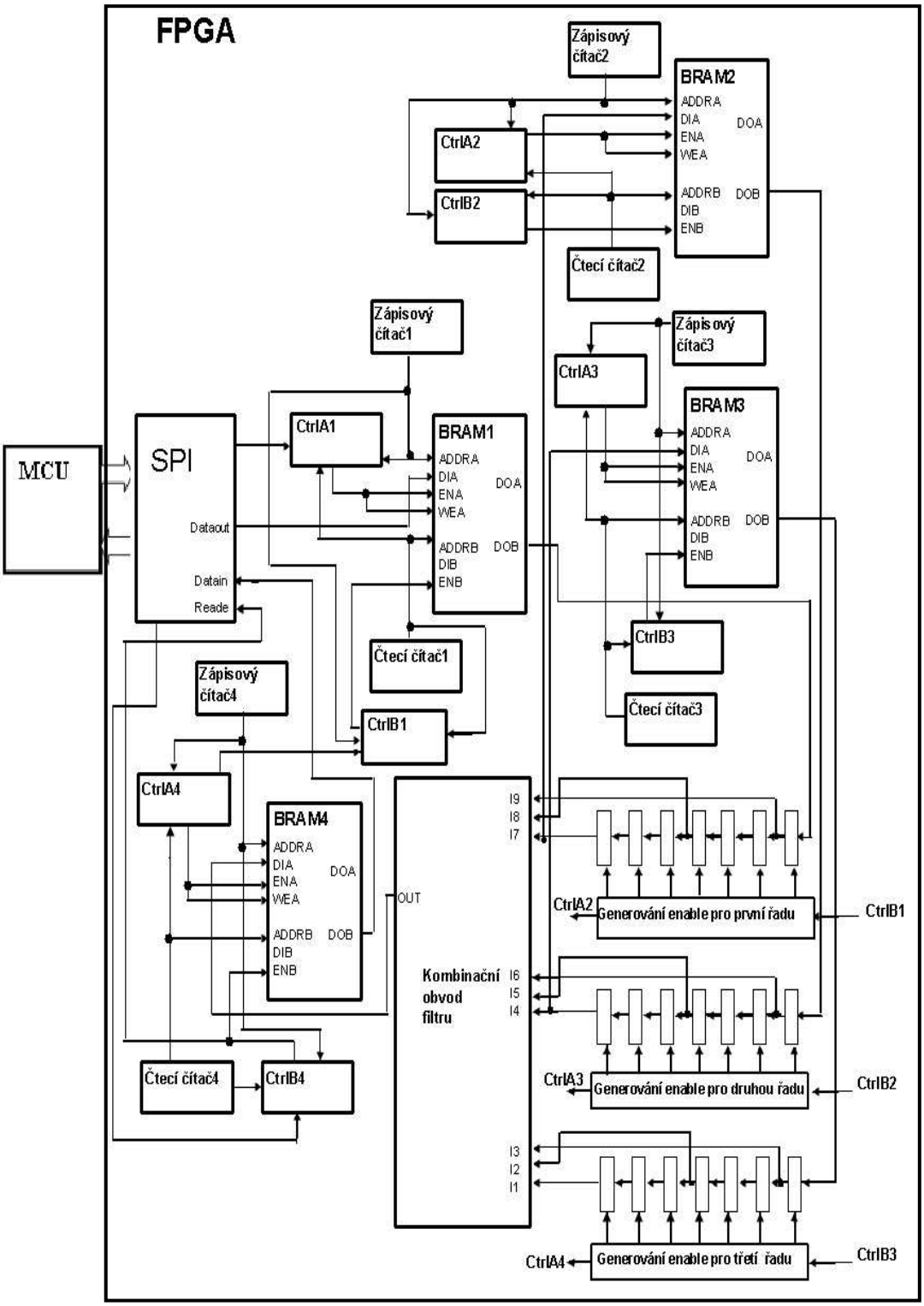
Urychlení výpočtů obvodovou realizací je umožněno zejména paralelizací některých výpočetních kroků, kterého lze při tomto způsobu řešení dosáhnout, jako např. současné provádění operací aritmetických operací, které na sobě nezávisejí apod. Další výpočetní ztráty, které zde nenastávají jsou režie způsobená zpracováním instrukcí, např. jejich dekódování. Některé instrukce navíc provádí akce, které při hardwarovém způsobu řešení neexistují, jako jsou přesuny operandů mezi registry a pamětí, programové skoky, volání funkcí a další, jimž se při provádění na procesorové jednotce nedá vždy vyhnout. Hardwarový obvod je na rozdíl od procesoru, který je univerzální jednotkou, specializovaný a „ušitý na míru“ tak, aby vyhovoval kritériím konkrétního problému. To umožňuje dosáhnout daleko vyšší efektivity implementace některých potřebných základních výpočetních operací. Např. bitový posun, jehož režie bude při provádění na procesoru rovna minimálně jednomu taktu, lze v obvodovém řešení realizovat bez jakéhokoli zpoždění.

Při realizaci filtrů jako kombinačních číslicových obvodů se obvykle uplatňuje přístup, kdy jsou na vstup obvodu přivedeny pouze potřebné hodnoty pixelů v okolí reprezentativního pixelu a výstupem je výsledná hodnota pro pixel:



Obrázek 25 Ilustrace výpočtu prováděného kombinačním obvodem jednotlivých filtrů

Obraz může být zpracováván sekvenčně pixel po pixelu a vždy aktuální hodnoty okolí pixelu se předávají ke zpracování jedné komponentě ilustrovaného obvodu. Může se ale také použít více komponent, na kterých můžou výpočetní operace probíhat současně. V extrémním případě může být použit stejný počet hardwarových komponent jako je pixelů v obraze. Zpravidla je ale filtrů méně a hodnoty pixelů jsou jim předávány pomocí řetězené linky složené z registrů a FIFO pamětí. Tento přístup umožní dosažení velkého zrychlení, používá se při zpracování obrazu v reálném čase a je implementován i v rámci tohoto projektu. Pohled na celé zapojení obvodu použitého při realizaci všech hardwarových filtrů je vyobrazen na obr. 26.



Obrázek 26 Realizace hardwarového filtru na FITkitu

3.2 Základní popis programů softwarové části hardwarového řešení a programových filtrů

Činnost filtrů je koncipována tak, že se do MCU nejprve nahraje patřičný program provádějící požadovaný typ filtrace, který musí být dále schopný uložit do flash paměti MCU obrázek, který se přenese z PC prostřednictvím USB rozhraní a který se má filtrovat. Součástí všech programů tedy musí být funkce, která zajistí správné zpracování dat souboru s obrázkem zasílaným přes USB do MCU a také funkce dodávající odpovídající data při odesílání obrázku získaného filtrací zpět do PC. Program musí dále umožňovat komunikaci s FPGA pomocí SPI, protože bude prostřednictvím tohoto rozhraní zasílat a číst obrazová data do/z příslušné BRAM. Použitým obrázkovým formátem je windows bitmap (přípona bmp), konkrétně varianta s 24-bitovou hloubkou barev. Důvodem je jeho relativně snadno zpracovatelná a přehledná vnitřní struktura. Na jeden pixel obrazu připadají tři byty, z nichž každý nese informaci o jasu jedné ze tří barev (RGB). Obrázek je uložen jako posloupnost bytů odpovídajících barevným složkám posloupnosti pixelů začínající levým dolním bodem obrázku a končící pravým horním bodem obrázku.

Kapacita paměti flash je 48kB. V této paměti jsou ale uložena také programová data a celou kapacitu paměti tudíž nelze k uchování obrázku využít. Programy byly navrženy tak, aby dokázaly zpracovávat obrázky maximálních možných velikostí, což vyžadovalo z důvodu relativně malého volného místa paměti flash hospodárné nakládání s ní. To je ale komplikováno vlastnostmi flash. Ta je rozdělena na bloky o velikosti 512KB. Před zápisem do bloku musí být tento nejprve celý vymazán. Kdyby bylo možné využít při implementaci paměti RAM, bylo by řešení relativně jednoduché. To znamená, obrázek by se při přenosu z PC ukládal do paměti, kde by nakonec tvořil souvislý blok začínající na určité adrese. Odtud by se postupně po bytech zasílal do BRAM na začátku datového řetězce realizovaného v FPGA. Při čtení z FPGA by se data ukládala opět do RAM počínaje stejnou adresou, na které začínal vstupní obrázek. V případě paměti flash by se ale před zapsáním bytu na tuto adresu musel nejprve smazat celý blok a tím by mohlo dojít i k výmazu hodnot, které ještě nebyly odeslány ke zpracování. Filtrovaný obrázek se tedy musí do paměti ukládat na adresu, která je nižší než adresa prvního ještě neposlaného bytu vstupního obrázku a mezi oběma adresami musí být dostatečný rozestup k tomu, aby nedošlo k vymazání bloku flash, ve kterém jsou uloženy dosud neodeslané hodnoty. Problém se více komplikuje při softwarovém způsobu filtrace při ukládání vypočtených hodnot. Zde musí být „offset“ mezi hodnotami v některých případech větší, protože data trojic pixelů připadajících na jednotlivé řádky osmiokolí jsou od sebe oddělena bloky paměti o velikosti počtu bytů, který zabírá v paměti řádek obrázku zmenšený o tři pixely. Při filtračním výpočtu potom nastává situace, kdy je nutné znát trojice hodnot barevných složek pixelů, které se mohou u obrázků dostatečně velkých šířek nacházet v různých třech blocích flash. Zápis musí být proveden na takovou adresu, aby nedošlo k výmazu poslední potřebné trojice hodnot. Ve všech programech je tedy nutné detekovat při ukládání vypočtených hodnot začátek dalšího bloku flash a v případě jeho nálezu musí být blok vymazán. V každém programu je proto definováno makro, jehož funkcí je vymazání bloku, určeného adresou, která náleží do intervalu adresového prostoru připadajícího na blok. Při filtraci je dále nutné rozlišovat mezi pixely okrajových a vnitřních oblastí snímku a pro výpočet bodů okrajových oblastí je nutné volat jim odpovídající rutiny, které provádějí příslušné výpočty. Současná nutnost detekce začátku bloků flash a konců a začátků řádků obrazu vede ke komplikovanějším programům, než by se možná dalo očekávat. Projeví se i mírným zvýšením režie programů. Filtrace prvního, resp. posledního řádku obrazu vyžaduje rovněž volání speciálních funkcí a je ve všech programech jakýmsi prologem, resp. epilogem hlavní filtrační funkce.

Programová část hardwarového řešení se od ostatních programů odlišuje tím, že místo volání funkce vracející novou hodnotu pro barevnou složku neokrajového pixelu obrazu se zavolá funkce, která přečte tuto hodnotu z FPGA pomocí SPI. Tyto programy musí také

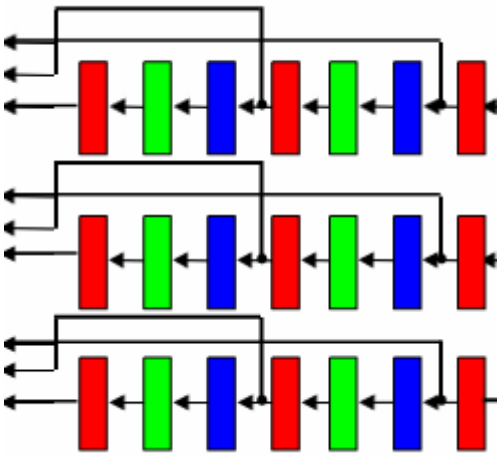
v hlavní smyčce odesílat vstupní obrazová data do FPGA. V případě detekce barevné složky pixelu, který se nachází v nejpravějším nebo nejlevějším sloupci obrázku se opět musí zavolat odpovídající výpočetní funkce, ale navíc také přečíst byte z FPGA – ten se nebude ukládat, neboť se nejedná o relevantní hodnotu, pouze je nutné ji odebrat z posloupnosti hodnot přicházejících z FPGA. Na každý pixel nacházející se v nejlevějším nebo nejpravějším sloupci připadají tři takovéto byty.

Filtrace probíhá vlivem použitého obrázkového formátu, směrem od spodního řádku k hornímu. Řádky se prochází směrem zleva doprava. Oblast okolí reprezentativního pixelu obsahující hodnoty, na základě kterých je možné provést diskrétní konvoluci v bodě obrazu odpovídajícímu reprezentativnímu pixelu nebo popřípadě provést jiný filtrační výpočet, jsou předávány příslušné výpočetní funkci pomocí adresy pixelu, který se nachází v dolní levém rohu okolí, z důvodu eliminace režie, která by nastávala při číselném předávání souřadnic a vypočítávání adresy pixelu v paměti. Funkce vždy vrátí hodnotu pro jednu barevnou složku a pro výpočet celého pixelu musí být volána vždy třikrát.

Detailní popis hlavních filtračních funkcí je možné nalézt v přiložené programové dokumentaci.

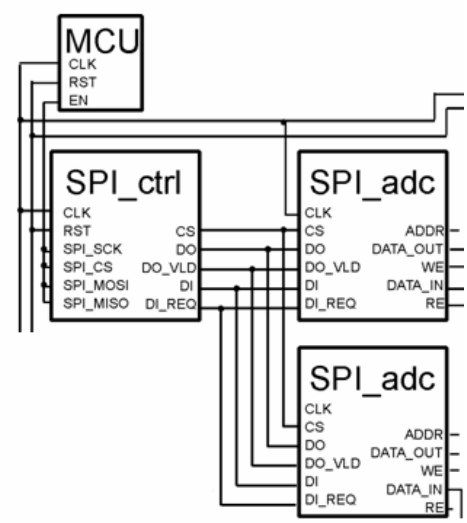
3.3 Architektura řetězené datové linky a SPI

Na obr. 26 je zachyceno schéma obvodu, který je společný pro všechny typy filtrů. Je zde zřetelně viditelná cesta dat, základní použité komponenty a jejich propojení. Detailní schéma je možné shlédnout v příloze. Z téhož obr. je patrná také komunikace mezi MCU a obvodovými komponentami. Paměti FIFO jsou realizovány pomocí dvou - portových variant BRAM. Pro datovou délku 8 bitů (9 bitů, viz kap. 2.6) umožňuje paměť adresaci 2k paměťových míst, což pro uložení počtu hodnot pixelů rovnajícímu se přibližně počtu pixelů v řádku obrazu bude jistě postačovat. Struktura obvodu je ovlivněna i formátem vstupních obrázků (windows bitmap, viz předchozí kapitola). Kombinační obvod filtru zpracovává v jednom taktu vždy výpočet jedné barevné složky pixelu. Funkci obvodu musí být přizpůsobena i situace na jeho vstupu. Pro každý řádek osmiokolí je zde sedmice registrů a na vstup obvodu jsou přivedeny pouze hodnoty z prvního, prostředního a posledního registru z každé řady. Pomocí zbývajících registrů je pouze docíleno uložení bytů nesoucích informaci o jasu zbývajících dvou barev, které zrovna v aktuálním taktu nejsou potřeba při výpočtu a zachování jejich správného pořadí. Problém je osvětlen na obr. 27 znázorňujícím příslušnost bytů uložených ve vstupních registrech ke stejným barevným složkám v určitém okamžiku činnosti obvodu. Vnitřní architektura kombinačního obvodu se liší v závislosti na typu filtru. Filtr pracující na základě Sobelova operátoru nepotřebuje pro funkci hodnoty všech devíti vstupních hodnot. Celá prostřední řada registrů by zde mohla být odstraněna a namísto dvou BRAM použita pouze jedna pro uložení dvou řádků. Pro jednoduchost je zde ale všech 21 registrů ponecháno i při realizaci tohoto filtru. Tímto opatřením nebude změněn počet hodnot, kterými musí být celá linka naplněna, aby kombinační

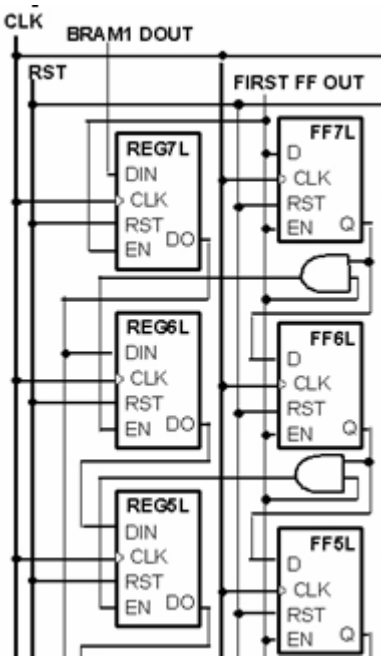


Obrázek 27 Vizualizace příslušnosti bytů uložených v registrech na vstupu kombinačního obvodu k barevným složkám RGB

obvod počítal s relevantními vstupními hodnotami, čili jím nebude způsobeno žádné zpoždění. Registry, do kterých jsou ukládána data z BRAM, by mohly být také odstraněny a hodnoty odpovídající posledním pixelům každého řádku okolí by mohly být na vstup kombinačního obvodu přiváděny přímo z BRAM. Čtení hodnoty z BRAM ale probíhá jinak, než čtení z registru a při zařazení BRAM do linky s registry je nutné jej ošetřit jistým speciálním způsobem (viz dále). Za účelem sjednocení synchronizace na vstupu kombinačního obvodu a zjednodušení celé situace tedy byly na začátek všech řad registrů (vzhledem ke směru pohybu dat) přidán jeden registr. Frekvence hodinového signálu, kterým jsou synchronizovány všechny komponenty je prozatím stejná jako kmitočet MCU (signál SMCLK). Registry vyžadují být před zahájením činnosti obvodu resetovány. To je zajištěno signálem RST mikrokontroleru. Jeho aktivace je společně s dalšími inicializačními akcemi, jako je inicializace SPI, již implementována ve funkci, která je součástí programové knihovny FITkitu. Tato inicializační funkce se provede jako první v každém programu, který se na FITkitu spustí. Detail vnitřní struktury použitého SPI rozhraní je zobrazen na obr. 28. Rozhraní je tvořeno jedním řadičem (SPI_ctrl) a dvěma dekodéry (SPI_adc). První slouží k zasílání nových dat do BRAM1 a ke čtení dat získaných filtrací z BRAM4. Šířka jeho datového vstupu a výstupu je 8 bitů. Na datový vstup druhého dekodéru je přiveden signál, jehož aktivní hladina indikuje neprázdný stav BRAM4. S pomocí dekodéru lze tedy v programu otestovat stav paměti a zabránit čtení, jestliže v paměti ještě není uložena žádná vypočtená hodnota. Datová šířka tohoto dekodéru je nastavena na jeden bit. Připojení pamětí BRAM k datovému vstupu SPI rozhraní umožňuje zasílat a číst data po dávkách a tím snížit počet iterací hlavní programové smyčky, pomocí které je uskutečňována filtrace, vzhledem k počtu iterací, který by se musel vykonat, kdyby se BRAM1 a BRAM4 nahradily registry. Ukládání hodnot je realizováno způsobem FIFO. Port A je užíván vždy u každé BRAM k zápisu nové hodnoty do paměti a port B slouží ke čtení hodnot z paměti. V BRAM2 jsou v průběhu výpočtu po naplnění celé linky uloženy hodnoty pixelů, které leží mezi prvním pixelem spodního řádku a posledním pixelem prostředního řádku aktuálně zpracovávaného osmiokolí a jsou zde uloženy i hodnoty dvou barevných složek, které náleží v různých okamžicích činnosti obvodu k prvnímu pixelu na spodním řádku osmiokolí a poslednímu pixelu prostředního řádku okolí. V BRAM3 jsou po naplnění linky uloženy hodnoty pixelů, které leží mezi prvním pixelem prostředního řádku aktuálně zpracovávaného osmiokolí a posledním pixelem horního řádku osmiokolí a jsou zde uloženy i hodnoty dvou barevných složek, které náleží v různých okamžicích činnosti obvodu k prvnímu pixelu na prostředním řádku osmiokolí a poslednímu pixelu horního řádku okolí. Činnost všech čítačů je povolena ve stejné okamžiky, kdy je povolen zápis, resp. čtení portu paměti, na který je z výstupu čítače přiváděna adresa.

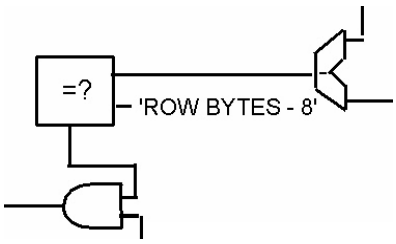


Obrázek 28 SPI rozhraní filtrů



Obrázek 29 Úsek první řady registrů na vstupu kombinačního obvodu včetně řízení

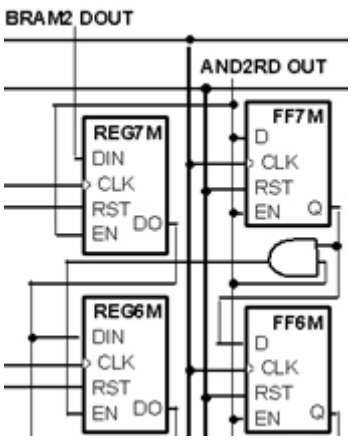
Adresová hodnota, na kterou je možné zapsat do BRAM1 nový prvek a z které lze přečíst hodnotu prvku v čele fronty, je přivedena z čtecího a zápisového čítače, které jsou přidružené k BRAM1. Vnitřní strukturu komponenty CtrlA1, která řídí zápis do BRAM1 je možné nalézt v detailním schématu v příloze. Sestává ze sčítačky, komparátoru a jednoho hradla AND. Zápis do BRAM1 je povolen v případě, že paměť není zaplněna a zároveň je signál WE (write enable) SPI dekodéru v aktivní hladině. Paměť není plná, je-li na výstupu komparátoru, na jehož vstup je přivedena hodnota adresy ze zápisového čítače zvětšená o jedna a hodnota adresy z čtecího čítače, aktivní hladina. Výstup komparátoru je v aktivní hladině v případě nerovnosti obou vstupních hodnot. Čtení z BRAM1 je řízeno komponentou CtrlB1. Tato se skládá ze sčítačky, komparátoru, hradla AND a klopného obvodu typu D a pro její činost je nutný i výstupní signál komparátoru komponenty CtrlA4. Čtení je povoleno v případě, že je BRAM1 neprázdná a zároveň není plná paměť BRAM4. Je-li v paměti BRAM1 uložena alespoň jedna hodnota, je výstup komparátoru, na jehož vstup je přivedena hodnota adresy z komponenty zápisový čítač1 a z komponenty čtecí čítač1, v aktivní hladině. Výstup komparátoru je v aktivní hladině v případě nerovnosti obou adres. Detekce stavu plné paměti BRAM4 je realizována stejně jako v případě BRAM1 a aktivní hladina na výstupu komparátoru opět signalizuje, že v paměti je ještě volné místo. Výstupy obou komparátorů jsou přivedeny na vstupy dvouvstupového hradla AND, jehož výstupem je signál povolující čtení z BRAM1. Vždy, když je tento signál v aktivní hladině, je možné posunout hodnoty v celém řetězci přivádějícím obrazová data.



Obrázek 30 Vnitřní struktura komponenty CtrlB2

Vzhledem k vlastnostem BRAM, která vystaví při čtení požadovanou hodnotu uloženou na nějaké adrese na výstup až v následujícím taktu, je ale nutné tento (enable) signál o jeden takt zpozdit. K dosažení tohoto cíle slouží klopný obvod typu D (označený na schématu v příloze First FF). Jeho výstupem je signál povolující zápis do prvního registru z řetězce registrů, pomocí kterých jsou přiváděny data přímo na vstup kombinačního obvodu filtru.

Na obr. 29 jsou zachyceny pouze první tři z těchto registrů z hlediska směru toku dat společně s jejich řízením. Registr, do kterého se ukládají data z BRAM1, je označen REG7L. V této řadě registrů jsou vždy (po naplnění celé linky) uloženy hodnoty barevných složek pixelů odpovídající spodnímu řádku okolí reprezentativního pixelu. Na vstup kombinačního obvodu jsou z této řady přivedeny pouze hodnoty z registrů REG7L, REG4L a REG1L. Ve zbývajících dvou paralelních řadách to jsou hodnoty z registrů stejného pořadí. Napravo je řada klopných obvodů typu D, které slouží k propagaci signálu enable. Tím je zajištěno zahájení činnosti dalších komponent řetězce ve správný čas, tedy až se k nim dostanou relevantní data. Na vstup EN klopných obvodů je přiveden výstup ze zmíněného klopného obvodu, který měl zpozdit povolující signál a propagace enable signálu je tedy možná jen v případě, že je možné provést čtení z BRAM1. Posun dat do následujícího registru řetězce je pro každý článek umožněn, jestliže je obsahem odpovídajícího klopného obvodu hodnota 1, která je aktivní hladinou signálu enable a zároveň je povoleno čtení z BRAM1. Obvod by pracoval správně i kdyby byl na vstup EN registrů přiveden stejný signál jako na vstupy EN klopných obvodů a příslušné součástky FPGA čipu realizující odpovídající AND hradla by mohly být využity pro jiné účely. Registry, ke kterým se ještě nedostala obrazová data, by pak předávaly dál pouze hodnoty, na které byly resetovány a pracovaly by zbytečně. Poslední (sedmé) hradlo AND,



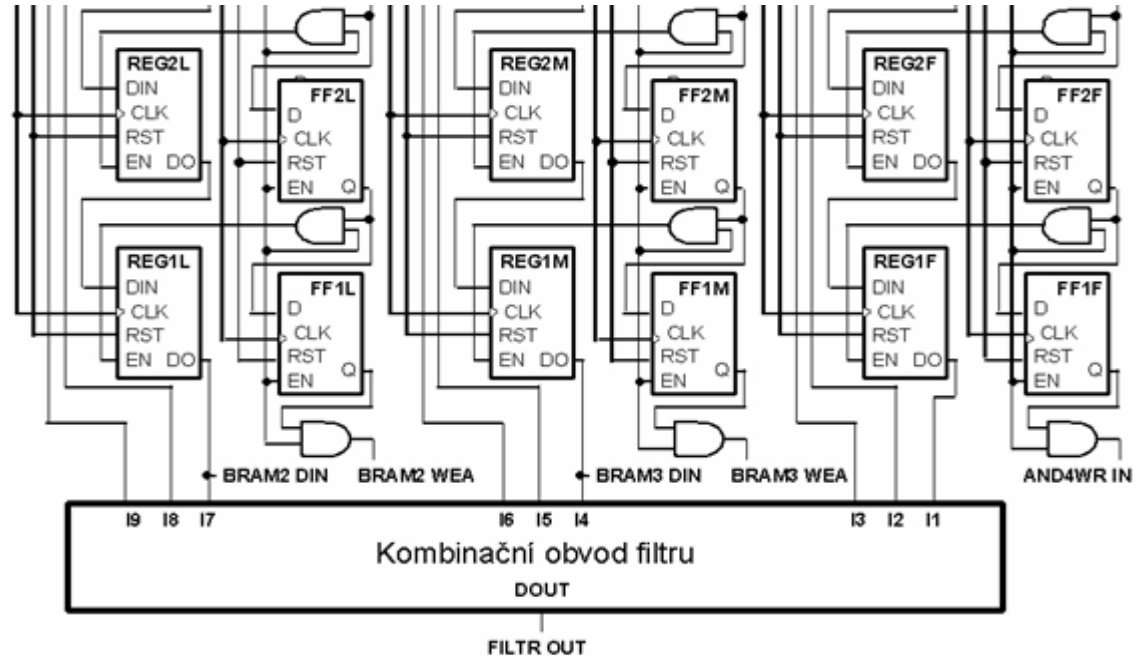
Obrázek 31 Úsek prostřední řady registrů na vstupu kombinačního obvodu

jehož výstup povoluje, je-li v aktivní hladině, zápis hodnoty uložené v posledním registru prvního řetězce (REG1L) do paměti BRAM2, je však pro správnou činnost obvodu nezbytně nutné (obr. 32 vlevo dole). Tímto hradlem je de facto realizována činnost komponenty CtrlA2 z obr. 26, která řídí zápis do BRAM2.

Adresy čelního prvku fronty uložené v BRAM2 a místa pro uložení nového prvku jsou přiváděny na adresové vstupy paměti z přidružených čítačů. Čtení z BRAM2 je umožněno až po naplnění fronty, jejíž délka je rovna počtu bytů připadajících na řádek obrazu zmenšený o hodnotu 8. V této hodnotě je zahrnuto 7 bytů, které jsou uloženy ve výše zmíněném řetězci registrů na vstupu kombinačního obvodu. Odčítanec je ještě zvětšen o jedna, z důvodu kompenzace zpoždění způsobeného při čtení z BRAM2. Smysl tohoto opatření by byl zřejmý při pohledu na průběhy signálů při simulaci činnosti obvodu. Indikaci plné fronty v BRAM2 zajišťuje komparátor, jehož vstupy jsou požadovaná hodnota délky a aktuální hodnota délky fronty vypočtená pomocí odčítačky z adres, které jsou vystaveny na výstupech čítačů náležejících k BRAM2 (obr. 30). Rozdíl hodnot obou adres bude vždy roven počtu prvků ve frontě uložené v BRAM2, přestože se hodnoty adres pohybují v adresovém prostoru cyklicky a nastane tedy v jistý moment situace, kdy se hodnota jedné z adres vynuluje. Výsledkem odečtení potom bude záporná hodnota rozdílu obou aktuálních hodnot v doplňkovém kódu, která je vlastně rovna binárnímu číslu vzniklému odečtením této hodnoty, zapsané jako kladné, od maximální hodnoty kterou lze zobrazit na daném bitovém rozsahu a následným přičtením jedničky. Výsledkem je tedy opět hodnota udávající délku fronty. Řídící komponenta CtrlB2 tedy sestává z odčítačky, komparátoru a hradla AND. Její vnitřní struktura je totožná s vnitřní strukturou komponenty CtrlB3, kde jsou však přivedeny na vstupy odčítačky adresy z čítačů náležejících k BRAM3. Signál přivedený na vstup ENB paměti BRAM2 z výstupu hradla AND řadiče CtrlB2 je v aktivní hladině, je-li povolen zápis do BRAM2 a současně je na výstupu komparátoru indikujícího stav plné fronty v BRAM2 hodnota logická 1. Tento stav nastane, jestliže je BRAM1 neprázdná, BRAM4 není plná a v BRAM2 je již uložen správný počet prvků.

Stejný signál je přiveden i na EN vstup prvního registru prostřední řady registrů na vstupu kombinačního obvodu (REG7M) a na EN vstupy klopných obvodů sloužících k propagaci signálu enable pro prostřední řadu registrů (obr. 31). V této řadě jsou vždy (po naplnění celého řetězce) uloženy hodnoty barevných složek pixelů odpovídající prostřednímu řádku okolí reprezentativního pixelu. Vztahy mezi jednotlivými registry a klopnými obvody jsou stejné jako v případě první řady registrů, které byly popsány výše. Datový výstup posledního registru je přiveden na datový vstup paměti BRAM3. Signálem povolujícím zápis do BRAM3 je výstup posledního hradla AND, kterým je realizována funkce řadiče CtrlA3. Signál je v aktivní hladině, když je povoleno čtení z BRAM2 a zároveň je na výstupu D posledního klopného obvodu prostřední řady (FF1M) hodnota logická 1. Paměť BRAM3 je řízena na základě hodnot adres čítačů zápisový čítač3 a čtecí čítač3 analogicky jako paměť BRAM2. Její datový výstup je přiveden na datový vstup prvního registru poslední posloupnosti registrů na vstupu kombinačního obvodu filtru (REG7F). Zápis do tohoto registru je povolen, je-li povoleno čtení z BRAM3 a stejný signál je opět přiveden i na EN vstupy klopných obvodů příslušejících poslední řadě registrů (obr. 32 vpravo). V těchto registrech jsou uloženy obrazová data odpovídající hornímu řádku okolí právě zpracovávaného pixelu. Je-li výstup posledního hradla AND v aktivní hladině, znamená to, že je linka naplněna a z výstupu kombinační obvodu je možné číst správné výsledky.

Zápis do BRAM4 je tedy povolen, je-li tato paměť nezaplněná a právě zmíněný signál v aktivní hladině. Stav plné paměti BRAM4 je detekován obdobným způsobem, jako tomu je v případě BRAM1, s využitím hodnot čítačů náležejících k BRAM4. Čtení z FIFO v paměti je povoleno, právě když je paměť neprázdná a signál RE SPI dekodéru je v aktivní hladině. Neprázdnost paměti je opět indikována analogicky jako u BRAM1. Řadič CtrlA4 tedy sestává ze stejných komponent jako řadič CtrlA1 a řadič CtrlB4 se skládá ze stejných komponent jako řadič CtrlB1.

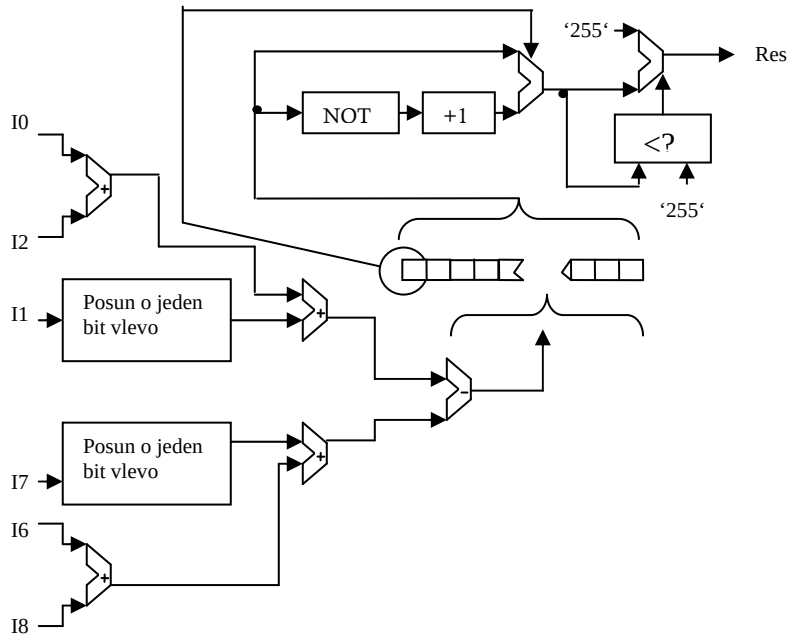


Obrázek 32 Koncový úsek tří řad registrů přivádějících data na vstup kombinačního obvodu filtru včetně jejich řízení a propagace signálu enable.

3.4 Architektura kombinačních obvodů filtrů

3.4.1 Architektura kombinačního obvodu Sobelova operátoru

Na obr. níže je nakresleno schéma zachycující vnitřní strukturu komponenty řešící výpočet Sobelova operátoru s maskou h_1*(-1) pro jednu barevnou složku, jejíž hodnota musí ležet v intervalu od 0 do 255:



Obrázek 33 Schéma kombinačního obvodu pro realizaci filtru pracujícího na základě Sobelova operátoru.

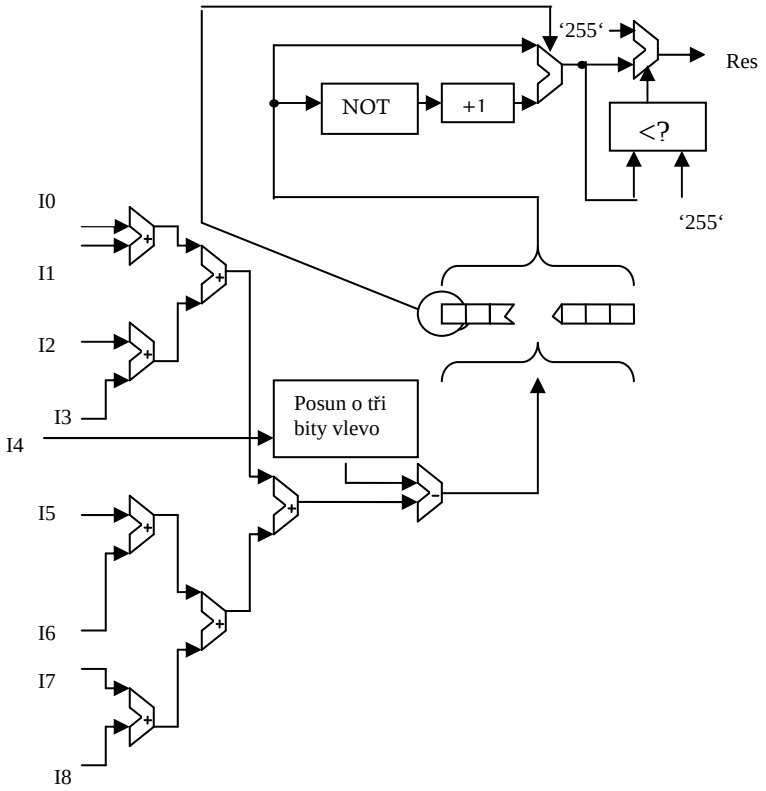
Nejprve současně proběhne součet hodnot barevných složek pixelů 0 a 2, 6 a 8 a bitový posun hodnot barevných složek pixelů 1, 7 (čísla označují pozice pixelů v osmiokolí, jak jsou vyobrazena na obr. 25 vlevo). V další fázi se sečtou hodnoty získané v první fázi a získá se tak výsledek součtu hodnot prvního řádku masky h_1*(-1) vynásobených s odpovídajícími hodnotami vstupního obrazu a součtu hodnot třetího řádku masky h_1*(-1) vynásobených s odpovídajícími hodnotami. Součet odpovídající hodnotám horního řádku masky je nyní nutné odečíst od součtu, který odpovídá hodnotám spodního řádku. Sčítačky v první úrovni musejí být 9-bitové, protože součtem dvou 8-bitových čísel může vzniknout 9-bitové číslo. Ze stejného důvodu musí být ve druhé úrovní sčítačky 10-bitové. Odčítačka ve třetí úrovni bude 11-bitová, neboť zde musí přibýt znaménkový bit.

Výsledkem odčítačky je vlastně hodnota v doplňkovém kódu. Znaménko hodnoty rozdílu je možné zjistit podle znaménkového bitu, který leží tradičně o jedno místo nalevo od pozice bitu s nejvyšším významem (řádem). Hodnotou znaménkového bitu je řízen multiplexor, který umístí na výstup buď nezměněný výstup odčítačky v případě, že na jejím výstupu byla

kladná hodnota nebo je na výstup umístěna hodnota, která byla na výstupu odčítačky záporná a musela být převedena na kladné číslo stejné velikosti. Toho je dosaženo negací všech bitů hodnoty a přičtením jedničky, což je typický postup při změně znaménka čísla reprezentovaného v doplňkovém kódu. Nakonec je nutné provést kontrolu velikosti vypočtené hodnoty a v případě, že došlo k přetečení povoleného rozsahu, zajistit, aby se na výstupu objevila maximální přípustná hodnota (č. 255). Detekce je docíleno pomocí komparátoru (jehož funkce může být založena pouze na horních třech bitech, tudíž jednoduchá a zpoždění relativně malé). Tím je ovládán multiplexor, jehož výstupem bude vypočtená hodnota nebo maximální povolená hodnota. Zpoždění kombinačního obvodu je tedy přibližně rovno $T = 4t_s + 2t_m + t_n + t_c$ (3.4.1), kde t_s je zpoždění sčítačky, které je považováno za přibližně stejné jako zpoždění odčítačky a zpoždění operace přičtení hodnoty 1 k nějakému číslu, t_m je zpoždění multiplexoru, t_n je zpoždění hradla invertoru a t_c je zpoždění komparátoru. Zpoždění připadající na komparátor by mohlo být ještě také eliminováno, kdyby se signál řídící poslední multiplexor (na výstupu celého obvodu) určil již z hodnoty na výstupu odčítačky a byl vyhodnocován paralelně s korekcí znaménka.

3.4.2 Architektura kombinačního obvodu Laplaceova operátoru

Na obr. 34 je nakresleno podobné schéma realizace Laplaceova operátoru využívající masku pro osmiokolí (obr. 5 vpravo):



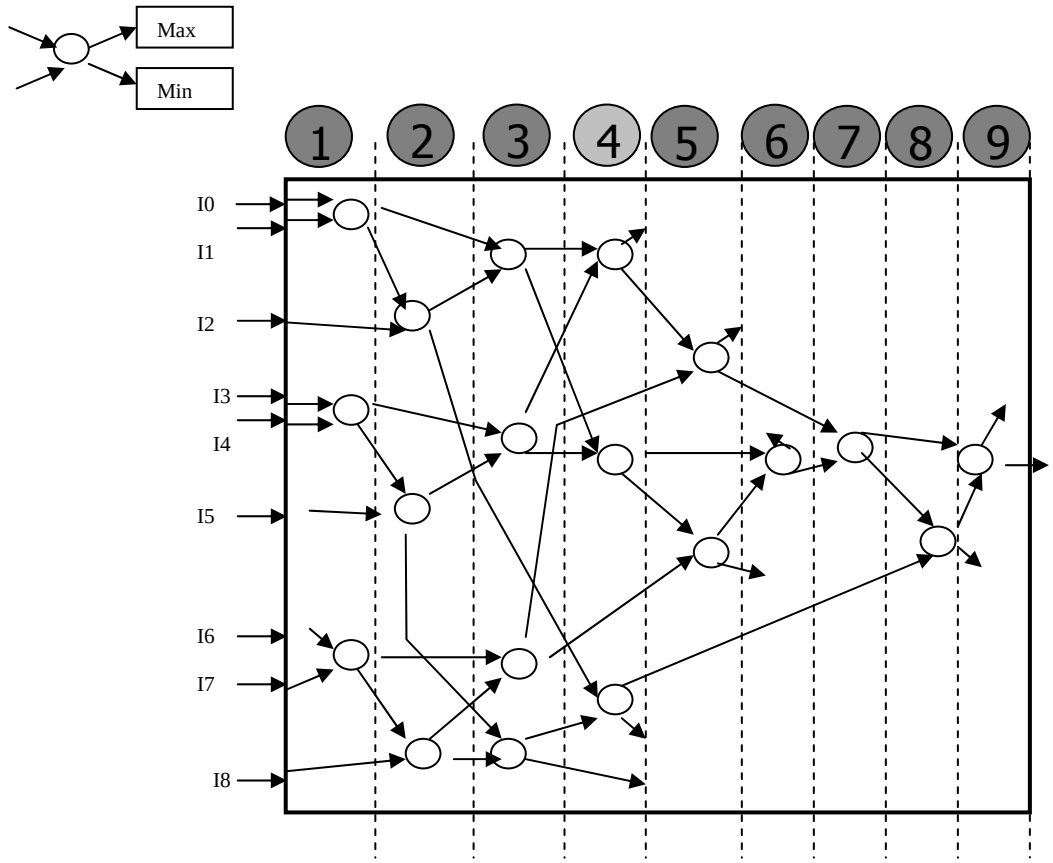
Obrázek 34 Schéma kombinačního obvodu pro realizaci filtru pracujícího na základě Laplaceova operátoru.

Pomocí sčítaček se vypočítá součet všech hodnot pixelů kromě pixelu na pozici 4 (obr. 25). Hodnota tohoto pixelu se mezitím bitově posune o tři řády vlevo. Výsledek součtů se potom odečte od hodnoty získané posunem, provede se korekce v případě záporné hodnoty nebo při přetečení stanoveného číselného rozsahu stejným způsobem jako u Sobelova operátoru a tím se získá výsledek. Sčítačky v první úrovni musejí být opět 9-bitové, ve druhé úrovní 10-bitové, sčítačka ve třetí úrovni 11-bitová a odčítačka ve čtvrté úrovni 12-bitová, protože zde musí přibýt znaménkový bit.

Celkové zpoždění kombinačního obvodu je při použití stejných značek pro zpoždění vnitřních komponent, jaké byly uvedeny v případě Sobelova operátoru, rovno: $T = 5t_s + 2t_m + t_n + t_c$ (3.4.2).

3.4.3 Architektura kombinačního obvodu mediánového filtru

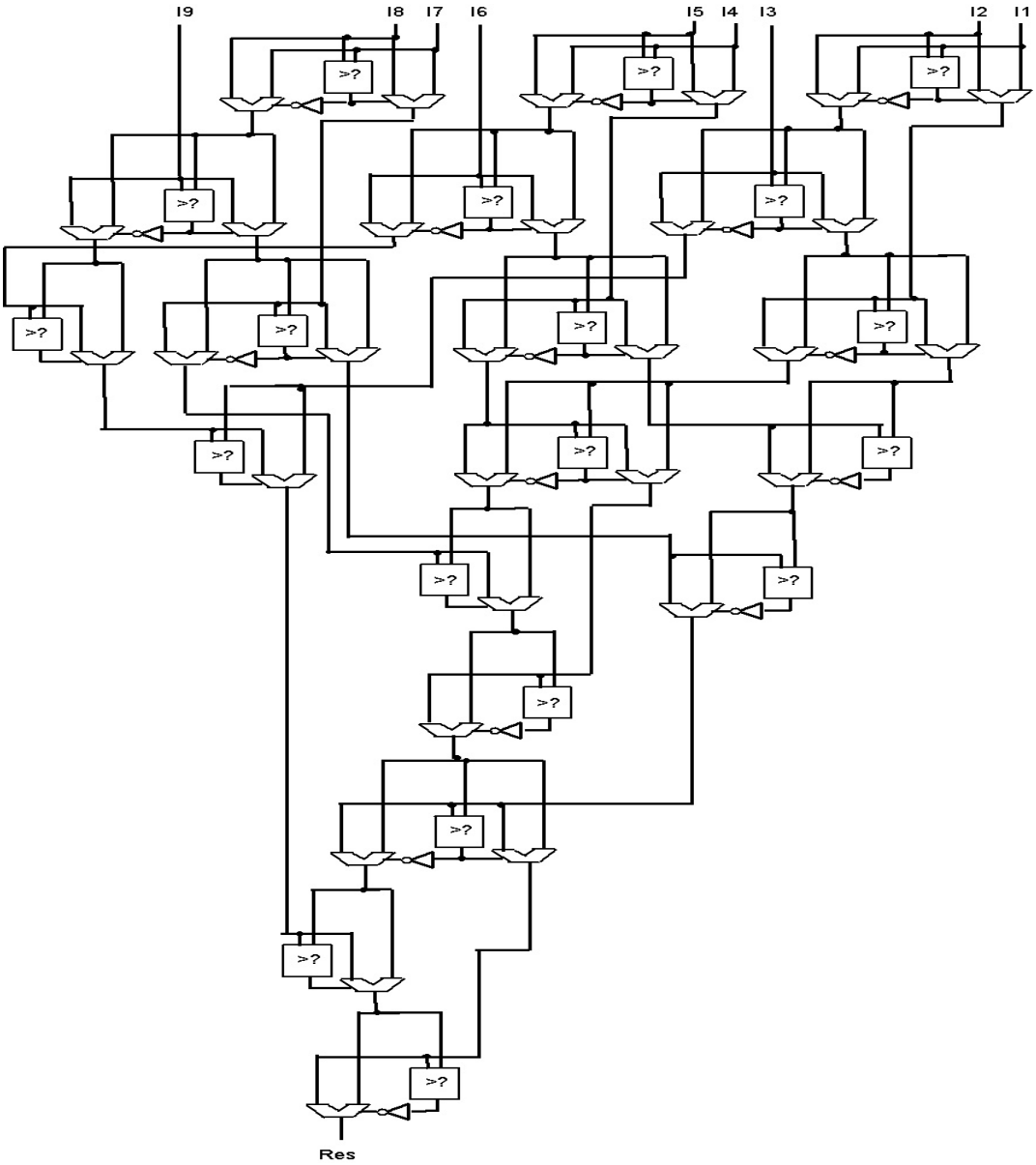
Implementovaný způsob hardwarové realizace nalezení mediánu je vyobrazen na následujícím obrázku:



Obrázek 35 Principiální schéma obvodu realizujícího mediánový filtr. Funkce elementu znázorněného kružnicí s šipkami je zřejmá z obr. vlevo nahoře.

Místa se stejným zpožděním jsou označena svislými přerušovanými čarami. Celkové zpoždění kombinačního obvodu pro devět vstupních hodnot je rovno $T = 9t_e$, kde t_e je zpoždění porovnávacího elementu.

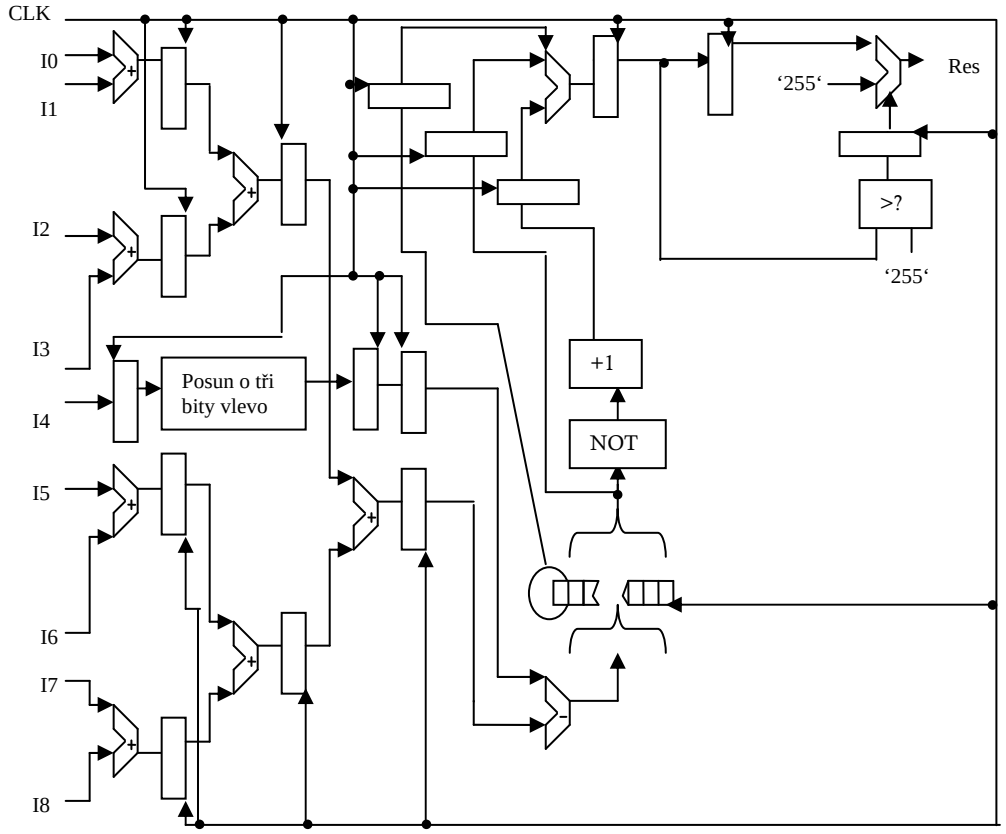
Schéma zobrazující vnitřní architekturu kombinačního obvodu pomocí známých logických komponent je nakresleno na obr. 36. Obvod se skládá z devatenácti porovnávacích elementů. Element je realizován pomocí jednoho komparátoru, dvojice multiplexorů a invertoru. Na vstupy multiplexorů a komparátoru musí být přivedeny ve stejném pořadí hodnoty, které se mají porovnávat, tzn. na první vstup všech komponent stejná první hodnota a na druhý vstup všech stejná druhá hodnota. U některých elementů je podle obr. 35 jeden výstup nevyužitý, a tyto je možné realizovat jen se spotřebou jednoho multiplexoru namísto dvou nebo případně bez invertoru. Nejdelší zpoždění obvodu je rovno $T = 9t_c + 9t_m + 7t_n$.



Obrázek 36 Schéma vnitřní struktury obvodu realizujícího výpočet mediánu z devíti vstupních hodnot.

3.4.4 Zřetězení kombinačních obvodů filtrů

Neuspokojuje-li zcela zrychlení dosažené nasazením výše uvedených obvodů požadavky kladené na dobu zpracování, je možné ještě provést jejich zřetězení. Zřetězení pro případ Laplaceova operátoru s vloženými registry a synchronizací by mohlo vypadat:



Obrázek 37 Obvod pro výpočet Laplaceova operátoru se zřetězením.

Největší zpoždění bude pravděpodobně vykazovat kombinační obvod sestávající ze sériově zapojené sčítačky (operace + 1) a invertoru. Je to obvod, jehož vstupní hodnota je čtena z registru, do kterého se uloží výstup z odčítačky. Perioda hodinového signálu musí být přizpůsobena kombinačnímu obvodu s největším zpožděním a musí tedy být dostatečně dlouhá na to, aby se hodnota invertovala, přičetla se k ní jednička a uložila se do registru, resp. pokud by déle trvala operace obecného součtu, musela by být perioda dost dlouhá na to, aby byl proveden součet a uložení do registru. Zřetězení u ostatních obvodů je možné uskutečnit analogicky. Kdyby požadavkům na rychlost zpracování nepostačovalo ani takto řetězené řešení, bylo by možné rozdělit výpočet na ještě menší úseky zřetězením sčítaček a komparátoru. Potom a nebo už i v případě prvního zřetězení by se možná staly nejpomalejším článkem celého obvodu FIFO paměti, ale ty by bylo možné eventuelně snadno nahradit posuvnými registry.

3.5 Návrh programové realizace výpočetních procesů filtrů

Obsahem této podkapitoly je návrh efektivní implementace úlohy filtrace obrazu v její sekvenční, resp. programové formě pro vybrané filtry. Je zde podrobně popsán programový způsob implementace stejných výpočtů, pro které byla v kapitole 3.4 navržena obvodová řešení.

3.5.1 Návrh programové realizace filtrů pracujících na základě Sobelova a Laplaceova operátoru

Maska Sobelova operátoru obsahuje jen hodnoty jedna a dva, čili zde výpočet násobení dvěma lze provést bitovým posunem číselné hodnoty o jeden bit doleva. Kromě sčítání a odčítání je pro implementaci nutná i operace porovnání, protože výsledek nesmí být záporný a nesmí překročit stanovený rozsah.

Hlavní rutina provádějící výpočet Sobelova operátoru s maskou h_1*(-1) (pro jednu barevnou složku) byla implementována v jazyce C následovně:

```
unsigned char SobelOperatorFiltr(unsigned char* addr)
{
    int res;
    res=addr[0]+(addr[3] <<1)+addr[6];
    addr+=(PIC_WIDTH_BYTES<<1);
    res=res-addr[0]-(addr[3] <<1)-addr[6];
    abs_val_saturate(res);
    return (unsigned char)res;
}
```

Funkce dostává jako parametr pouze adresu, na které je uložena hodnota některé barevné složky prvního pixelu dolního řádku osmiokolí. Nejprve se provede součet hodnot barevných složek pixelů, které korespondují s nejnižším řádkem konvoluční masky, přičemž hodnota barevné složky prostředního pixelu musí být bitově posunuta o jedno místo doleva. „Offset“ mezi barevnými složkami je, jak plyne z popisu formátu windows bitmap, tři byty. Poté se posune ukazatel v paměti o počet bytů odpovídající množství paměti, které zabírají dva řádky obrazu a tím se dostane na adresu odpovídající barevné složky prvního pixelu z trojice pixelů, které odpovídají hornímu řádku konvoluční masky. Hodnoty těchto pixelů se odečtou od již získaného součtu. Nyní je nutné ověřit, zda je hodnota proměnné res nezáporná a leží v požadovaném intervalu. V případě, že není splněna některá z těchto dvou podmínek, je nutné hodnotu upravit způsobem popsaným v kapitole 2.5. To zajistí makro abs_val_saturate.
Výpočet okrajových bodů, které leží v prvním a posledním řádku obrazu a v levém a pravém sloupci je uskutečněn pro každou z těchto oblastí speciální funkcí, jejichž struktura je podobná výše uvedené základní funkci.

Násobení se dá obejít i v implementaci Laplaceova operátoru s maskou pro osmiokolí – provede se bitové posunutí o tři bity. Základní funkce realizující výpočet neokrajových bodů je konstruována podobně jako v případě Sobelova operátoru:

```
unsigned char LaPlaceOperatorFiltr(unsigned char *addr)
{
    int res;
    res=addr[0]+addr[3]+addr[6];
    addr+=PIC_WIDTH_BYTES;
    res=res+addr[0]-(addr[3]<<3)+addr[6];
    addr+=PIC_WIDTH_BYTES;
    res=res+addr[0]+addr[3]+addr[6];
    abs_val_saturate(res);
    return (unsigned char)res;
}
```

Ve srovnání se základní rutinou Sobelova operátoru se v této funkci provádí větší množství aritmetických operací. Typičtěji se takovéto výpočetní problémy implementují v cyklech, ale prosté přiřazení vztahu poběží rychleji, protože nenastane režie při zpracování instrukcí skoků a při testování podmínky cyklu.

3.5.2 Návrh programové realizace mediánového filtru

Při implementaci mediánového filtru byla použita nerekurzivní varianta algoritmu Find. Funkce použitá pro výpočet neokrajových hodnot obrázku je přílohou 2. První cyklus zařazení položek do správných množin se odlišuje, protože jsou hodnoty čteny z flash. Proměnné M1, M2, M3, M1_cnt, M2_cnt, M3_cnt a address_free jsou globální. Na začátku hlavní filtrační funkce se naalokuje pole o délce 40 prvků. Datovým typem položky pole je neznaménkový celočíselný typ o délce 8 bitů. Adresa každého desátého prvku pole ve směru od začátku se postupně přiřadí ukazatelům M1, M2, M3, address_free. Pomocí polí M1, M2, M3 jsou realizovány stejnojmenné množiny rekurzivní verze algoritmu uvedené v kap. 2.4. Délka pole pro každou množinu je potom 10 prvků. V zájmu snadnějšího přepisu algoritmu jsou pole indexována od prvku s indexem 1 (ne 0). V proměnné adress_free je vždy uložena adresa prvního prvku pole, které bude možné v příští iteraci algoritmu využít pro některou z množin M1, M2, M3. V ukazateli ptr je uložena adresa prvního prvku množiny M. Činnost funkce je zřejmá při porovnání s rekurzivní verzí.

Byla zahájena také implementace mediánového filtru využívající řazení pomocí algoritmu Quick Sort, ale nebyla dotažena do konce, neboť již při ladění vykazovala „viditelně“ (několikanásobně) horší výsledky, než implementace využívající algoritmus Find. Nepříznivost těchto výsledků byla patrně způsobena skutečností, že byla použita rekurzivní verze algoritmu. Avšak i v případě efektivní implementace algoritmu Quick Sort nelze pravděpodobně očekávat „výrazně lepší“ výsledky, než nabízí realizace filtru používající algoritmus Find.

4 Zhodnocení implementace filtrů

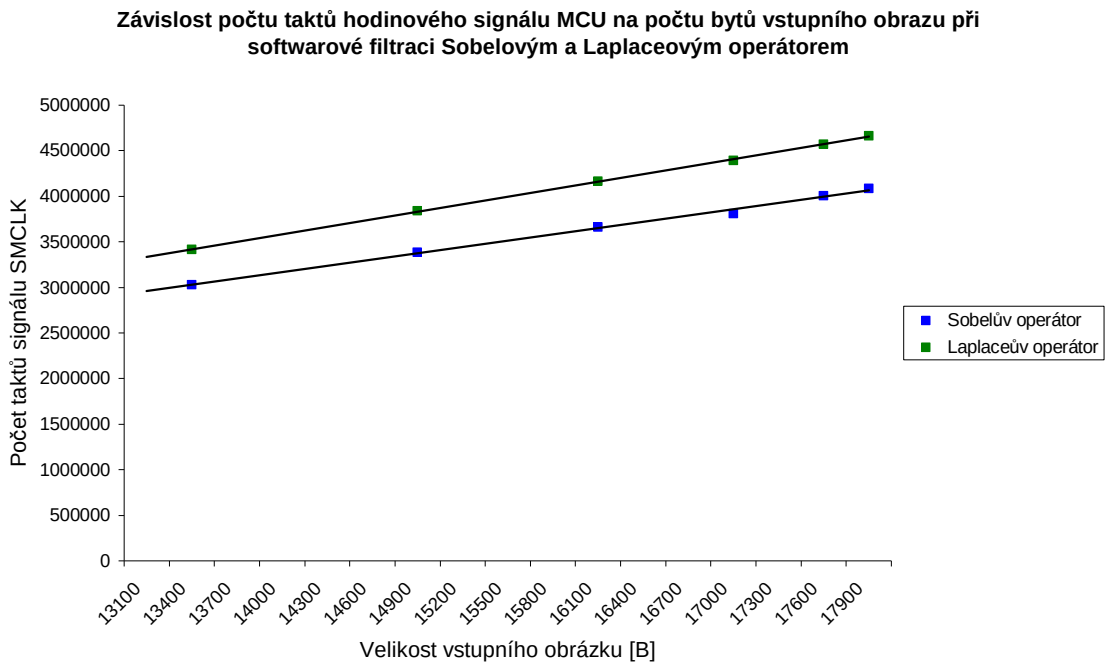
V této kapitole je pojednáno o tom, jak se zdařilo na přípravku FITkit implementovat jednotlivé filtry z hlediska doby filtrace a množství využitých prostředků. Pro každý filtr byl změřen počet taktů MCU, který byl vykonán při filtraci každého z testovacích obrázků. Počet taktů hodinového signálu byl měřen pomocí čítače realizovaného v FPGA. Na začátku hlavní funkce každého filtru byla vždy povolena jeho činnost a po skončení filtračního procesu byla opět zakázána. Údaje o výsledcích syntézy VHDL popisu filtrů (jako je např. množství využitých LUT nebo maximální zpoždění obvodu) byly získány ze souboru output.srp, který je při syntéze generován.

4.1 Softwarové filtry

Získané výsledky jsou zobrazeny v grafech s vyznačenými naměřenými hodnotami i s křivkou, která aproximuje hodnoty metodou nejmenších čtverců a zviditelňuje druh závislosti času zpracování na množství vstupních dat.

4.1.1 Sobelův a Laplaceův operátor

Křivky závislostí jsou podle očekávání pro oba filtry přímkami a jedná se tedy o lineární závislost. Přímka závislosti Laplaceova operátoru je strmější, než přímka závislosti Sobelova operátoru, protože se zde projevuje vyšší počet výpočetních operací Laplaceova operátoru. Obrázek s grafem včetně tabulky naměřených hodnot je na následující straně. Konkrétní rovnice aproximovaných křivek a z nich patrné hodnoty směrnic přímek nejsou zobrazeny, neboť aproximace křivky ze šesti relativně blízkých hodnot se ukázala jako nedostatečně směrodatná a vyšla poměrně nepřesně (přímka by protínala osu y = 0 v záporných hodnotách).

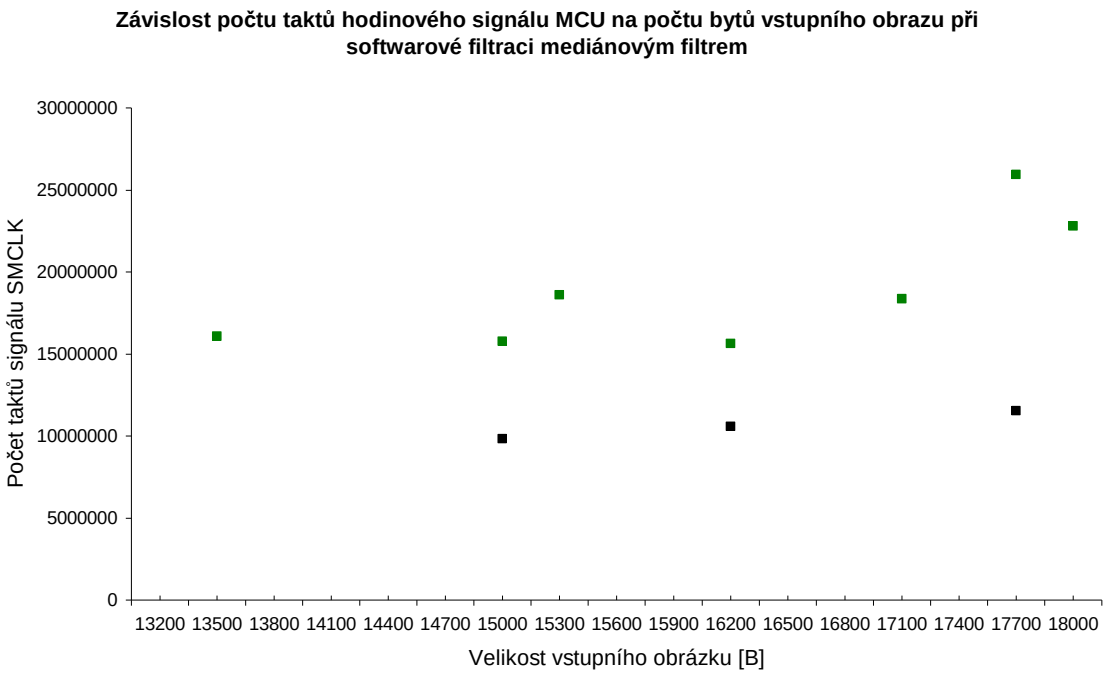


Velikost obrázku [B]	13500	15000	16200	17100	17700	18000
Počet taktů MCU (SO)	3025804	3385354	3660304	3806869	4007330	4083304
Počet taktů MCU (LO)	3414388	3838198	4162288	4394495	4572440	4660888

Obrázek 38 Naměřené hodnoty počtu taktů při softwarové filtraci Sobelovým a Laplaceovým operátorem.

4.1.2 Mediánový filtr

V grafu na obr. 39 jsou zobrazeny pouze naměřené hodnoty, jelikož zde není patrný žádný známý druh závislosti. Chování tohoto filtru z hlediska doby zpracování vstupních dat se odvíjí od vlastností použitého algoritmu (Find). V grafu je zachycena i závislost jeho výpočetní složitosti na hustotě změn jasu ve vstupním obrázku. Hodnoty bodů nakreslených černou barvou jsou získány z filtrace jednobarevných snímků. V souladu s očekáváním vykazuje potom algoritmus lineární řád složitosti. Zbývající hodnoty se pohybují mezi mezní přímkou, která odpovídá nejkratší možné době zpracování a mezi křivkou, která určuje nejhorší dobu zpracování, jež je polynomiálního řádu.



Velikost obrázku [B]	13500	15000	15300	16200	17100	17700	18000
Počet taktů MCU	16086152	15754585	18594007	15642294	18365158	25940500	22793736
Jedna barva	-	9818072	-	10588644	-	11527469	-

Obrázek 39 Naměřené hodnoty počtu taktů při softwarové filtraci mediánovým filtrem.

4.2 Hardwarové filtry

Pro komponenty veškerých čítačů, sčítaček a odčítaček byly při syntéze nalezeny odpovídající zařízení FPGA. Registry pro uložení bytových hodnot byly syntetizovány osmicemi klopných obvodů typu D. Přehled celkového počtu využitých zařízení čipu Spartan3 pro jednotlivé filtry je mimo jiné uveden v následujících podkapitolách.

Komponenty využité pro realizaci řetězené linky společné pro všechny filtry dle informací vrácených při syntéze:

Počet bitů	Komponenta	Počet ks
1	KO D	339
4	čítač	2
11	čítač	8
1	třístavový buffer	38
11	sčítačka	2
11	odčítačka	2
11	komparátor <>	4

Z výsledků je patrné, že dva komparátory na rovnost jejichž jedním vstupem je konstanta (obr. 30), byly realizovány pomocí jednodušších logických komponent.

4.2.1 Sobelův operátor

Souhrn komponent potřebných pro realizaci kombinačního obvodu filtru:

Tabulka 1

Počet bitů	Komponenta	Počet ks
12	sčítačka	5
12	odčítačka	1
12	komparátor <	1

Využití komponent čipu Spartan3:

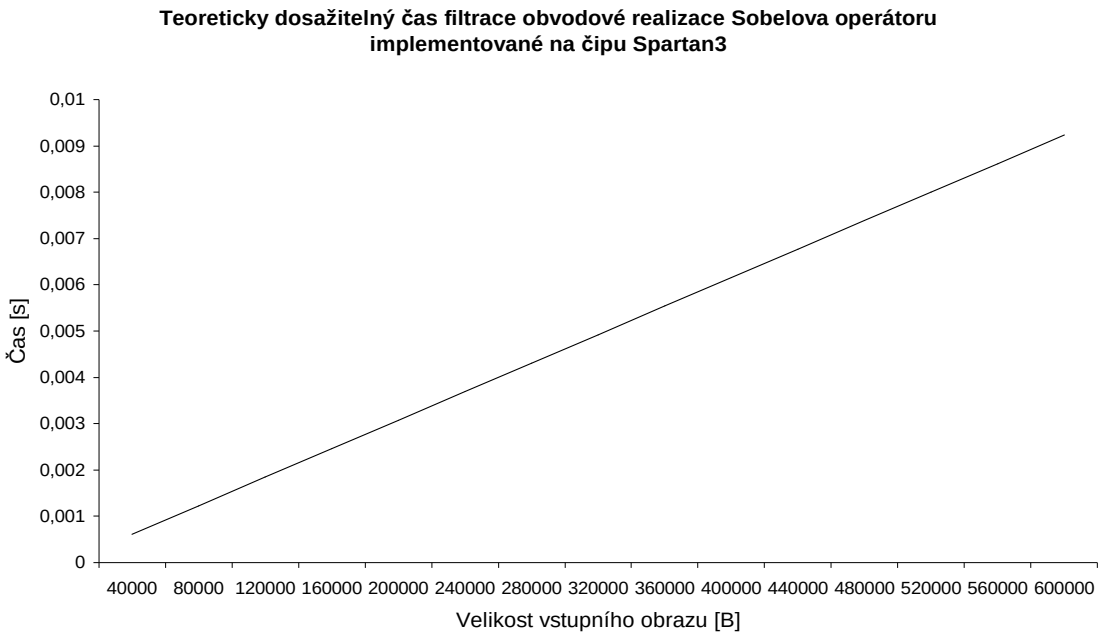
Tabulka 2

Komponenta	Využito	Celkově	Využito v %
Slice	289	768	37%
4 – vstupové LUT	320	1536	20%
MUXCY	164	1536	11%
MUXF5	5	768	1%
XORCY	141	1536	9%
VCC	1	-	

Nahrazení 12-bitových sčítaček sčítačkami s nižším rozsahem vede po syntéze na snížení zpoždění kombinační cesty s největším zpožděním o pouhou setinu ns a dojde dokonce k využití většího počtu Slice. Pro jednoduchost byla tedy ponechána původní verze implementace s 12-bitovými sčítačkami.

Zpoždění kombinačního obvodu a tedy i nejkratší možná perioda hodinového signálu činí T = 15.415ns, tj. f = 64.872MHz. Z toho 9.042ns připadá na zpoždění logických hradel a 6.373ns na zpoždění sítě.

Na přípravku Fitkit nelze potenciál, který nabízí výše uvedená frekvence v praxi využít, protože komunikace pomocí SPI rozhraní funguje ještě na nižším kmitočtu, než jakým je taktován MCU. Nejlepší dosažitelný teoretický čas filtrace obvodu, kterého by bylo dosaženo, kdyby nebyla činnost obvodu zpomalována komunikací, je zobrazen v grafu na obr. 40. Ve výpočtu křivky je zahrnuta skutečnost, že pro provedení filtrace části obrazu připadající na hardwarové zpracování je třeba počet taktů obvodu rovnající se počtu bytů vstupního obrazu zmenšený o počet bytů připadající na jeden řádek (ten poslední) obrazu a poté ještě zmenšený o 3. Na ose x je tedy vynesena celková velikost vstupního obrázku a na ose y minimální čas, po který trvá hardwarová filtrace části obrázku připadající na hardwarové zpracování. Poměr mezi šířkou a výškou filtrovaných obrázků se uvažuje 4:3.



Obrázek 40 Nejkratší dosažitelný čas filtrace implementované hardwarové realizace Sobelova operátoru

4.2.2 Laplaceův operátor

Pro realizaci kombinačního obvodu filtru bylo třeba:

Tabulka 3

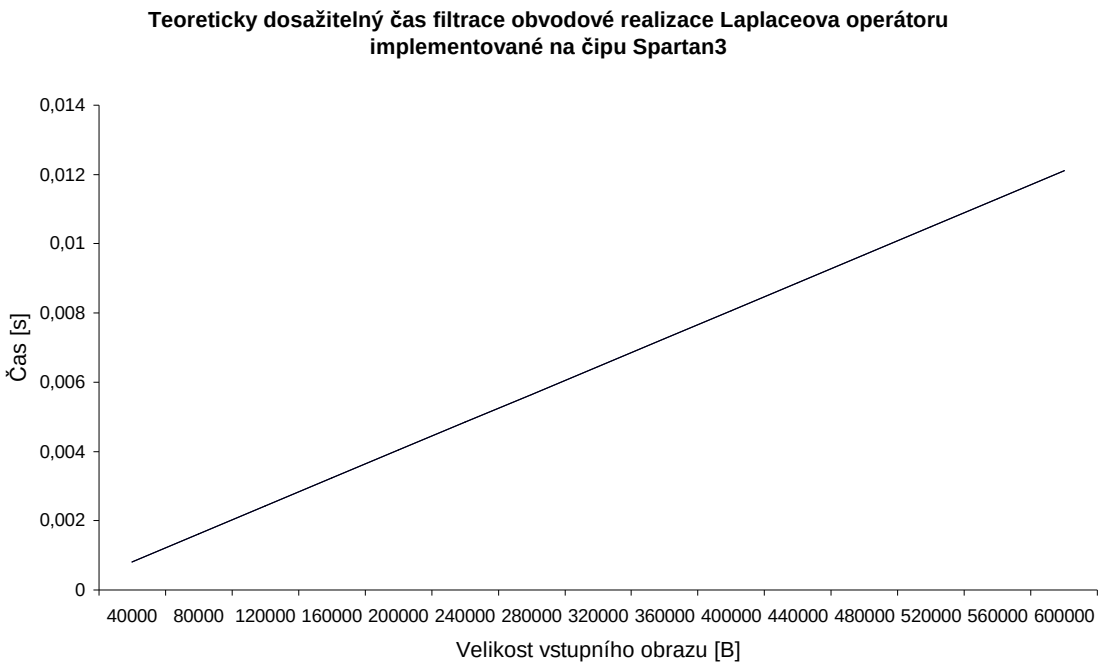
Počet bitů	Komponenta	Počet ks
12	Sčítačka	8
12	Odčítačka	1
12	komparátor <	1

Využití komponent čipu Spartan3:

Tabulka 4

Komponenta	Využito	Celkově	Využito v %
Slice	303	768	39%
4 - vstupové LUT	370	1536	24%
MUXCY	187	1536	12%
MUXF5	5	768	1%
XORCY	163	1536	11%
VCC	1	-	

Největší zpoždění kombinačního obvodu je T = 20.223ns, tj. f = 49.449MHz. Z toho 12.726ns připadá na zpoždění logických hradel a 7.497ns na zpoždění sítě.



Obrázek 41 Nejkratší dosažitelný čas filtrace implementované hardwarové realizace Laplaceova operátoru

Na obr. 41 je opět možné vidět graf znázorňující dobu filtrace při využití maximální taktovací frekvence. Výpočet křivky byl proveden podobně jako v případě Sobelova operátoru. Směrnicí přímky je minimální doba periody, či-li je logicky strmější než přímka závislosti Sobelova operátoru. Pro vstupní obrázek o velikosti 624871 bytů dojde vzhledem k Sobelovu operátoru k prodloužení času filtrace, připadajícímu na obvod, o 3ms.

4.2.3 Mediánový filtr

Pro realizaci kombinačního obvodu filtru bylo třeba:

Tabulka 5

Počet bitů	Komponenta	Počet ks
12	komparátor >	19

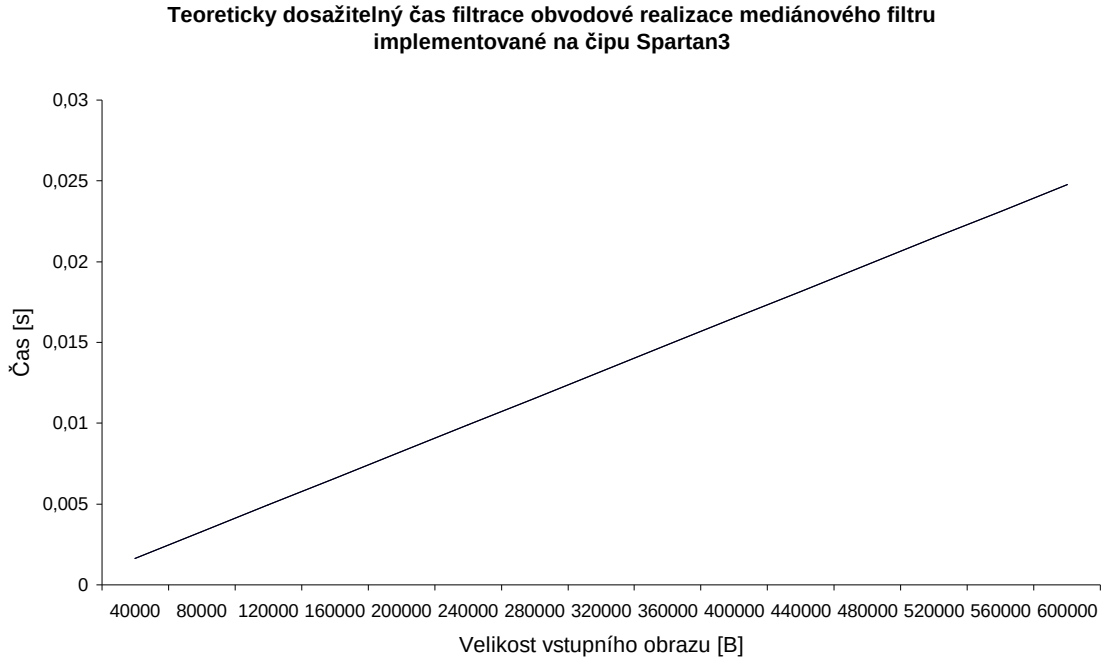
Využití komponent čipu Spartan3:

Tabulka 6

Komponenta	Využito	Celkově	Využito v %
Slice	430	768	55%
4 - vstupové LUT	635	1536	41%
MUXCY	276	1536	12%
MUXF5	5	768	1%
XORCY	102	1536	11%
VCC	1	-	

Vzhledem k předchozím dvěma filtrům zde dochází kvůli použití velkého počtu komparátorů k většímu využití LUT a tím i Slice. Nahrazení 12-bitových komparátorů 8-bitovými nevede po provedení syntézy (behaviorální VHDL popis) k žádnému časovému zlepšení, ani ke změně počtu využitých komponent.

Největší zpoždění kombinačního obvodu je T = 41.349ns, tj. f = 24.185MHz. Z toho 20.152ns připadá na zpoždění logických hradel a 21.197ns na zpoždění sítě.



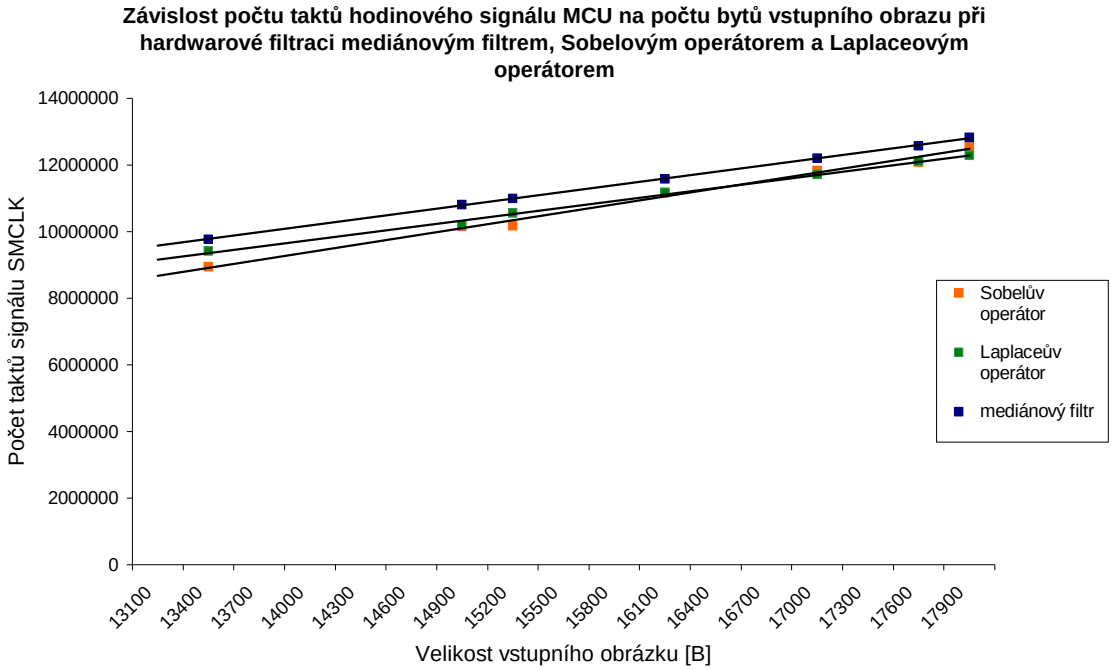
Obrázek 42 Nejkratší dosažitelný čas filtrace implementované hardwarové realizace mediánového filtru

Výpočet křivky byl proveden stejně jako v předchozích případech. Přímka je ve srovnání s předchozími opět strmější. Při filtraci obrázku o velikosti 624871 bytů dojde nyní vzhledem k Sobelovu operátoru k navýšení času filtrace již o 16.182ms.

4.2.4 Srovnání se softwarovou realizací

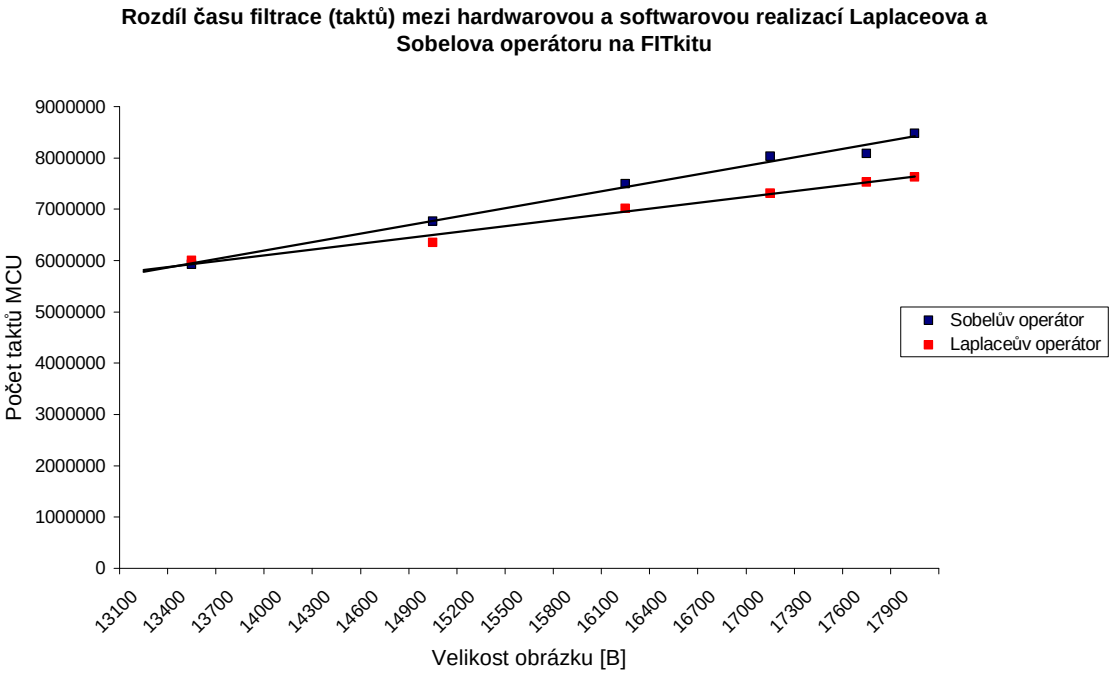
Nyní bude provedeno porovnání času zpracování implementovaných hardwarových filtrů, který je na přípravku FITkit prakticky dosažitelný, s časem filtrace prováděné softwarovými filtry. Srovnávacím měřítkem bude počet taktů hodinového signálu MCU (SMCLK). Na obr. 43 jsou vyneseny křivky závislosti počtu taktů signálu SMCLK na velikosti obrázku. Z grafu je zřejmě patrná lineární závislost, která byla pro hardwarovou realizaci všech filtrů očekávaná. Doba zpracování se liší pouze režií softwarové části realizace, neboť počet bytů odesílaných do FPGA je pro všechny filtry stejný a hodinový kmitočet hardwarových filtrů nyní taktéž. Strmost křivky znázorňující závislost Sobelova operátoru by měla být spíše menší, než větší, ve srovnání se strmostí křivky závislosti Laplaceova operátoru. Tato odchylka je pravděpodobně způsobena nepřesností výpočtu rovnice křivky z důvodu nedostatečného množství testovacích hodnot. Grafy na obrázcích 43, 44 a 45 umožňují snadné porovnání časů zpracování hardwarové a softwarové realizace filtrů na FITkitu. Křivky grafu na obr. 44 znázorňují rozdíl počtu taktů naměřených při filtraci s využitím hardwarové a softwarové realizace Sobelova a Laplaceova

operátoru. Rozdíl je získán odečtením počtu taktů odpovídajících softwarové realizaci od počtu taktů odpovídajících hardwarové realizaci. V případě těchto filtrů nebylo dosaženo žádného zrychlení. Rozdíl roste pro Sobelův operátor se zvyšující se velikostí obrázků rychleji.



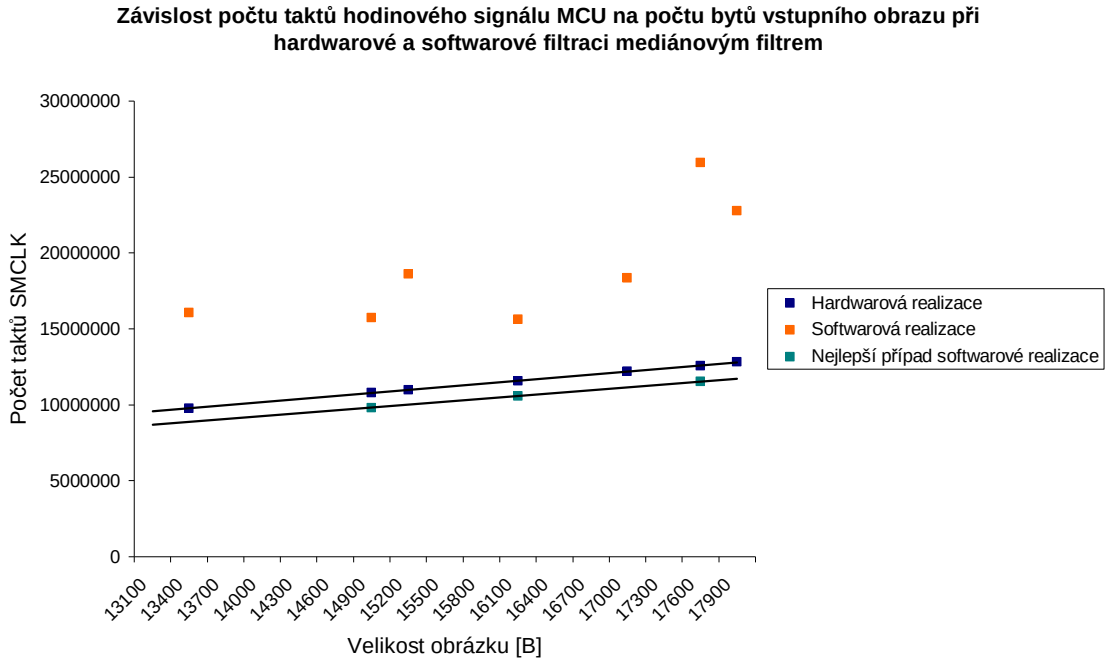
Velikost obrázku [B]	13500	15000	15300	16200	17100	17700	18000
Počet taktů MCU (SO)	8945028	10150426	10174534	11158139	11837984	12088904	12561223
Počet taktů MCU (LO)	9411001	10185071	10559401	11174096	11837984	12102064	12282001
Počet taktů MCU (MF)	9763799	10802669	10985999	11584036	12208199	12575462	12819299

Obrázek 43 Naměřené hodnoty počtu taktů při hardwarové filtraci



Obrázek 44 Rozdíl počtu taktů naměřených při hardwarové a softwarové filtraci Laplaceovým a Sobelovým operátorem.

V případě mediánového filtru je křivka závislosti počtu taktů hardwarové realizace „velmi podobná a blízká“ křivce, která by zachycovala závislost pro nejlepší případ počtu taktů softwarové realizace mediánového filtru. Křivky jsou téměř rovnoběžné a relativně málo vzdálené, či-li lze hardwarový čas zpracování považovat za přibližně stejný jako je nejlepší možný programově dosažitelný čas. Tato skutečnost je dobře viditelná z obr. 45. Zde lze tedy na základě grafu usuzovat, že bylo oproti průměrnému času zpracování programového filtru dosaženo jisté akcelerace. Přímka aproximující nejlepší případ softwarového řešení je však vypočtena na základě pouze tří hodnot a pro podložení výše uvedeného tvrzení by bylo vhodné provést větší počet měření. K tomu ale kvůli nedostatku času nedošlo.



Obrázek 45 Počet taktů naměřených při hardwarové a softwarové filtraci mediánovým filtrem s aproximovanými křivkami

4.2.5 Vliv rozhraní SPI na možnosti hardwarové akcelerace filtrace obrazu na přípravku FITkit

V tabulce 7 je uvedena režie funkcí jednotlivých filtrů (uvedených v kapitole 3.5.1), kterými je implementován výpočet nové hodnoty barevné složky reprezentativního pixelu na základě hodnot barevných složek pixelů v jeho okolí. Dále je zde zaznamenána režie funkce, která slouží ke komunikaci pomocí SPI, přičemž je změřena její nejúspornější varianta, která neobsahuje žádná volání funkcí knihovny libmsp, ale přímo příkazy jazyka C. Tato je použita i při implementaci filtrů. Režie je vyjádřena počtem taktů hodinového signálu MCU. Hodnoty byly opět naměřeny pomocí čítače realizovaného v FPGA (režie měřících funkcí je již odečtena a činí 570 taktů).

Tabulka 7

Funkce	Počet taktů
Sobelův operátor	124
Laplaceův operátor	311
SPI funkce (s odesláním)	293
SPI funkce bez zaslání dat	224

Položka SPI funkce v sobě zahrnuje implementaci akcí odeslání operačního kódu, adresy a samotných dat. Všechny tři fáze probíhají při odeslání každého bytu do FPGA. Obrazová data jsou do FPGA zasílána po dávkách a teoreticky by bylo možné implementovat komunikaci tak, aby první i druhá fáze proběhla pouze před odesláním prvního bytu každé dávky a pro každý byte dávky se prováděla pouze třetí fáze. Takto ale není rozhraní používáno zatím žádnou implementovanou aplikací a ani autorovi se výše uvedený způsob nezdařilo implementovat (na hlubší průzkum nevycházel již čas). V případě úspěšné implementace lze očekávat snížení režie připadající na SPI funkci asi o dvě třetiny. Frekvence komunikace (zasílání bitů) může být nanejvýš polovinou frekvence hodinového kmitočtu MCU. Nejmenší počet taktů MCU, který připadá na samotné odeslání operačního kódu, adresy a dat je tedy 48 taktů ($2\times8\times3$). Pokud by se odesílala pouze data, byl by nejmenší počet taktů 16.

V položce SPI funkce bez odeslání dat tabulky 7 je zaznamenána režie samotného programu implementujícího komunikaci. Z uvedených hodnot je zřejmé, že zejména implementace funkce obsluhující SPI rozhraní má zásadní vliv na omezení maximální možné dosažitelné hardwarové akcelerace na přípravku FITkit. Pro dosažení výraznější efektivity by byla nutná její implementace v jazyku symbolických instrukcí. K té je ale možné se uchýlit i v případě filtračních funkcí. Hardwarová akcelerace Sobelova operátoru na FITkitu se tedy nejeví jako slibná. V případě mediánového filtru by však bylo možné dosáhnout „zajímavých“ výsledků.

Doba filtrace hardwarových filtrů (zachycená např. na obr. 43) je výrazně ovlivněna zejména neoptimální implementací komunikace pomocí SPI, které je spouštěno z hlavní smyčky softwarové části řešení a nevyužitím maximální možné frekvence komunikace.

Nejcitelnější ztráty implementace hardwarových filtrů jsou zapříčiněny nevyužitím obousměrnosti komunikace SPI. Data jsou buď jen zapisována nebo jen čtena, SPI však umožňuje provádět tyto operace současně. Odstraněním tohoto nedostatku by bylo možné urychlit stávající implementaci hardwarových filtrů až dvojnásobně.

Nejnižší vzorkovací frekvence musí být dle teorie čtyřnásobkem frekvence SPI_SCK. V praxi ale fungovala komunikace až při šestkrát nižší frekvenci, než je frekvence MCU. Tento stav je asi způsoben nevyladěním všech parametrů SPI na optimální úroveň (na které již rovněž nezbýval čas). Jako základní synchronizační frekvence SPI byla užita frekvence MCU, ale bylo by možné využít generátor hodinového signálu v FPGA, který nabízí vyšší frekvence a umožnit tak dosažení plné přenosové rychlosti SPI avizované v kap. 2.7.2. Nejmenší počet taktů MCU, který připadá na samotné odeslání operačního kódu, adresy a dat, je tedy 144 taktů ($6\times8\times3$). Režie připadající na samotné vykonání programu SPI funkce je odesíláním bitů zhoršena o 69 taktů, čili asi o třetinu.

5 Závěr

Seznámili jsme se s významem obrazových filtrů v oblasti výpočetní techniky a informatiky a s jejich typickými aplikacemi, s teoretickými poznatky souvisejícími s problematikou filtrace obrazu navazujícími na teorii lineárních systémů a matematickou analýzou funkcí více proměnných, s odvozením konvolučních jader filtrů, na které je zaměřen tento diplomový projekt – tedy jader mediánového filtru, Sobelova a Laplaceova operátoru. Implementace byla směrována na přípravek FITkit, proto bylo nutné blíže se seznámit s jeho architekturou a jeho používáním. Kvůli umožnění dosažení efektivní implementace hardwarových filtrů na cílové architektuře FPGA bylo třeba i tuto blíže nastudovat a popsat.

V rámci projektu byl proveden, s využitím teoretických znalostí týkajících se časové složitosti algoritmů, návrh časově efektivních softwarových rutin pro výpočet operátorů nebo jiných vztahů, na jejichž základě filtry pracují. Dále byly s cílem dosažení nejkratší doby filtrace navrženy kombinační obvody provádějící výpočty ekvivalentní těm, které jsou uskutečňovány softwarovými filtračními rutinami. Následně byly implementovány softwarové filtry v jazyce C a hardwarové filtry v jazyce VHDL tak, aby mohla bez chyb proběhnout jejich syntéza. Doba zpracování obou přístupů byla změřena a porovnána. Z výsledků získaných přímým změřením doby filtrace několika testovacích obrázků vyplývá, že v případě Sobelova a Laplaceova operátoru nebylo na přípravku FITkit dosaženo žádné hardwarové akcelerace. Tento stav je zapříčiněn rychlostí komunikace mezi MCU a FPGA čipem FITkitu a také neoptimálním využitím komunikačního rozhraní v implementaci. V případě mediánového filtru je možné považovat uskutečněnou hardwarovou implementaci za akcelerující. Při plném využití potenciálu, který nabízí implementovaná hardwarová řešení, tzn. maximální frekvence hodinového signálu, by však došlo k „výraznému“ urychlení zpracování v případě všech filtrů.

V projektu by bylo možné dále pokračovat různými směry. Bylo by možné odstranit rezervy stávající softwarové části realizace hardwarových filtrů, která nevyužívá obousměrnosti komunikačního rozhraní SPI. Dosavadní koncepce realizace hardwarových filtrů uvažuje pouze akceleraci výpočtu hodnot neokrajových oblastí snímku. Tuto by bylo možné rozšířit na hardwarové zpracování celého vstupního obrázku. Jinou možností rozvíjení projektu by mohla být hardwarová implementace zřetězení kombinačních obvodů filtrů a zjištění nejkratší dosažitelné délky periody hodinového signálu při realizaci filtrů na dostupné FPGA technologii. Značný podíl zpoždění kombinačních obvodů (v případě mediánového filtru až 50%) připadá na zpoždění na vodičích propojujících logiku. Namapování komponent popsaných ve VHDL prováděné při syntéze může být ovlivněno souborem s definovanými omezeními (constraints). Vhodnými definicemi těchto omezení by mohlo být docíleno namapování komponent tak, aby bylo zpoždění na vodičích, kterými jsou propojeny, zmenšeno. Zajímavé by mohlo být také provedení implementace jednotlivých filtrů využívajících pro výpočet filtrace větších okolí pixelu a porovnání časových složitostí zjištěných u těchto filtrů se stávajícími. Softwarový způsob řešení mediánového filtru by mohl být realizován s využitím efektivní implementace algoritmu Quick Sort nebo na základě algoritmu Select modifikovaného pro hledání mediánu v posloupnostech krátkých délek a srovnání doby filtrace s algoritmem Find.

Použitá literatura a zdroje informací

[1] Václav Hlaváč: Zpracování signálů a obrazů, 2000, ISBN 80-01-03110-1

[2] W. Niblack: An introduction to digital image processing, 1986, New Jersey

[3] Vladan Majerech: Úvod do složitosti a NP-úplnosti, pokusné skriptum UK (URL http://hq.alert.sk/~mandos/fmfi-uk/Informatika/mff.cuni.cz/Uvod%20Do%20Slozitosti%20A%20NP-Uplnosti/sloz.pdf)

[4] FITkit, FIT VUT Brno: URL http://merlin.fit.vutbr.cz/FITkit/

[5] Xilinx: Spartan-3 FPGA Family Complete Data Sheet (srpen, 2005)
[6] Xilinx: Virtex™FieldProgrammableGateArrays (duben, 2001)
[7] Xilinx: Virtex-II Field-ProgrammableGateArrays (září, 2002)
[8] Xilinx: Virtex-II Pro™PlatformFPGAs: CompleteData Sheet (duben, 2004)
[9] Xilinx: Virtex-4 FamilyOverview (červen 2005)
[10] Xilinx: Virtex-4 RocketIOMulti-GigabitTransceiver (červen 2005)
[11] Xilinx: XtremeDSPforVirtex-4 FPGAsUserGuide (říjen, 2005)
[12] Xilinx: PowerPCProcessorReference Guide (září, 2003)

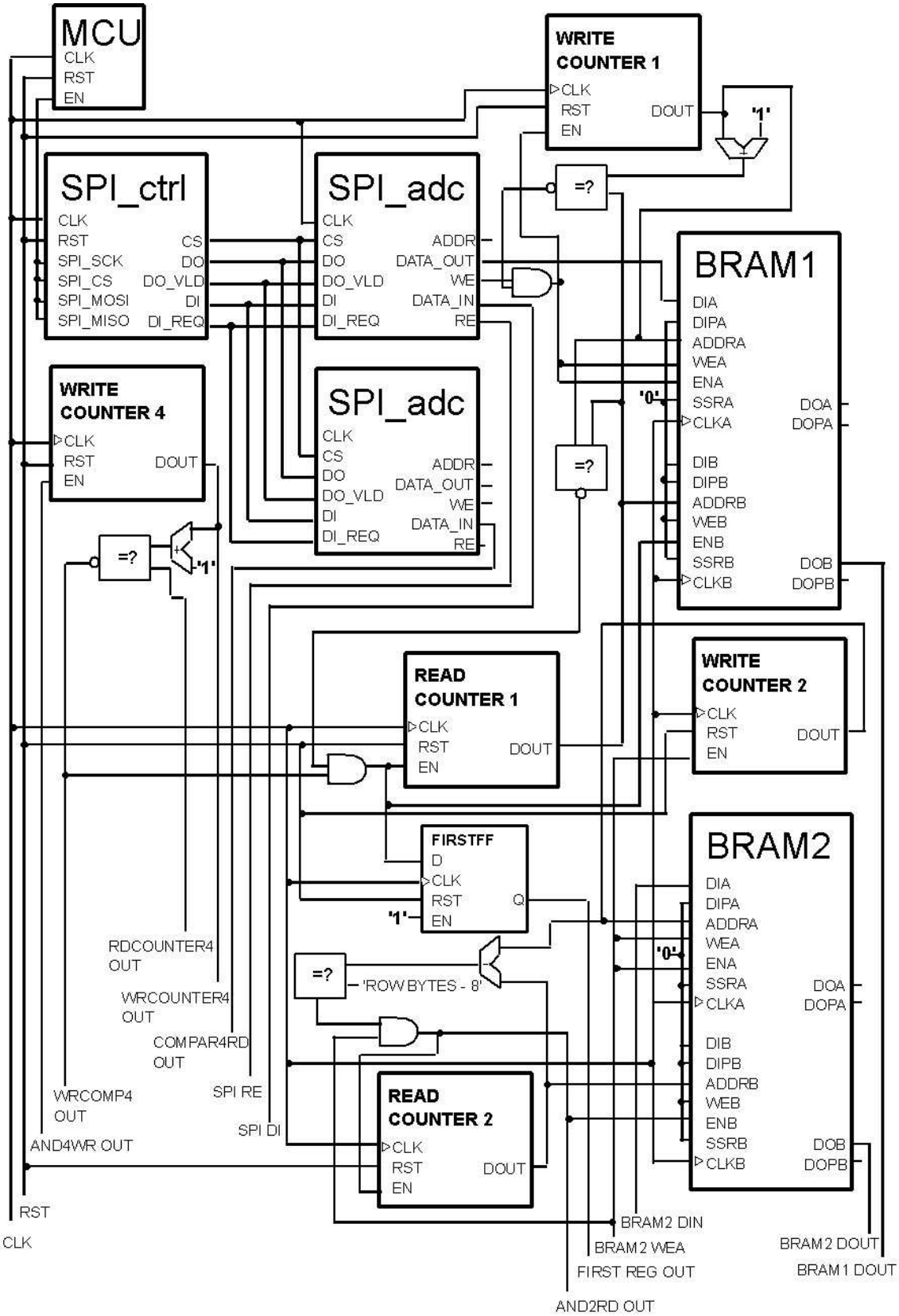
Internetová stránka Xilinx: http://www.xilinx.com

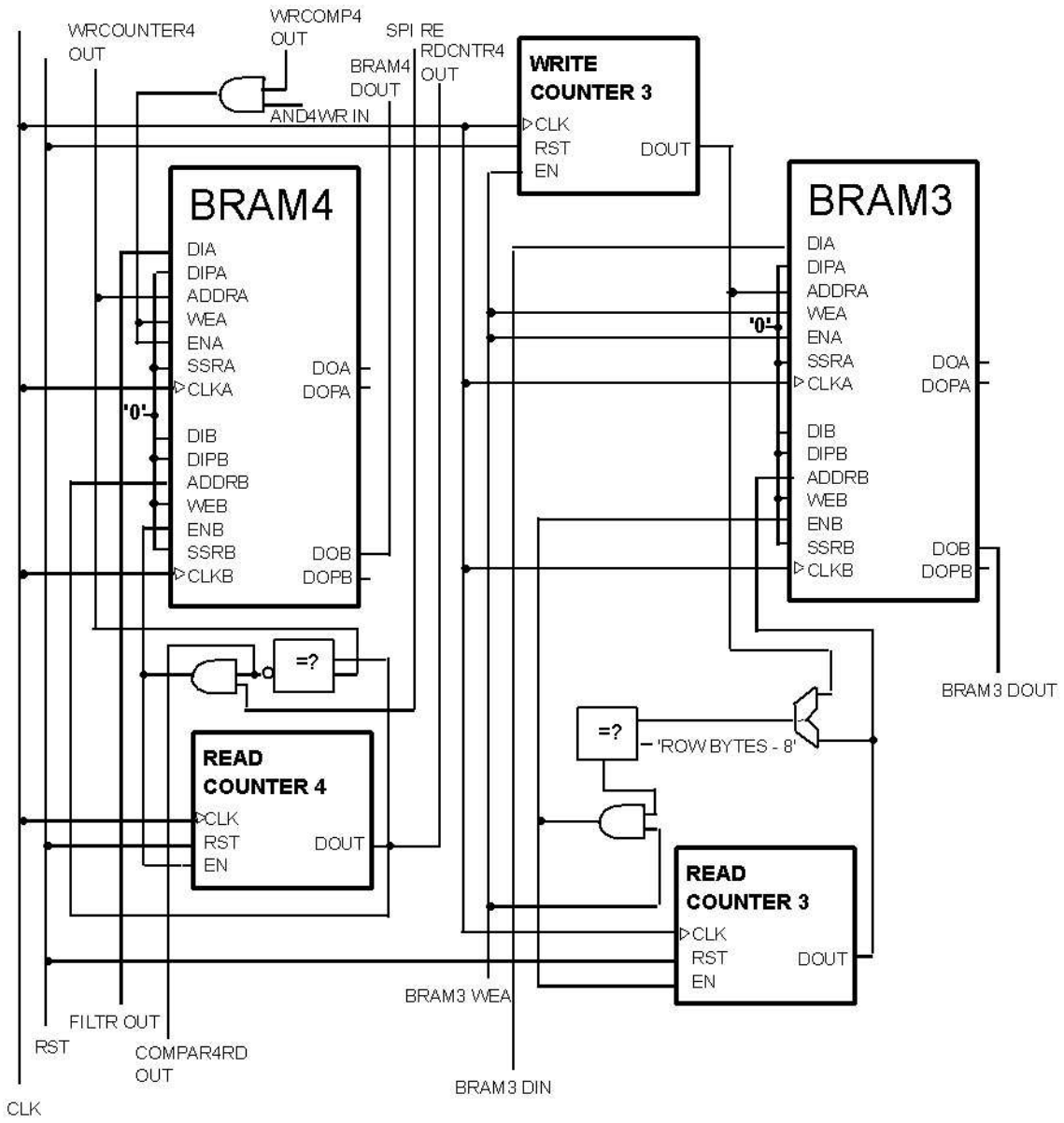
Seznam příloh

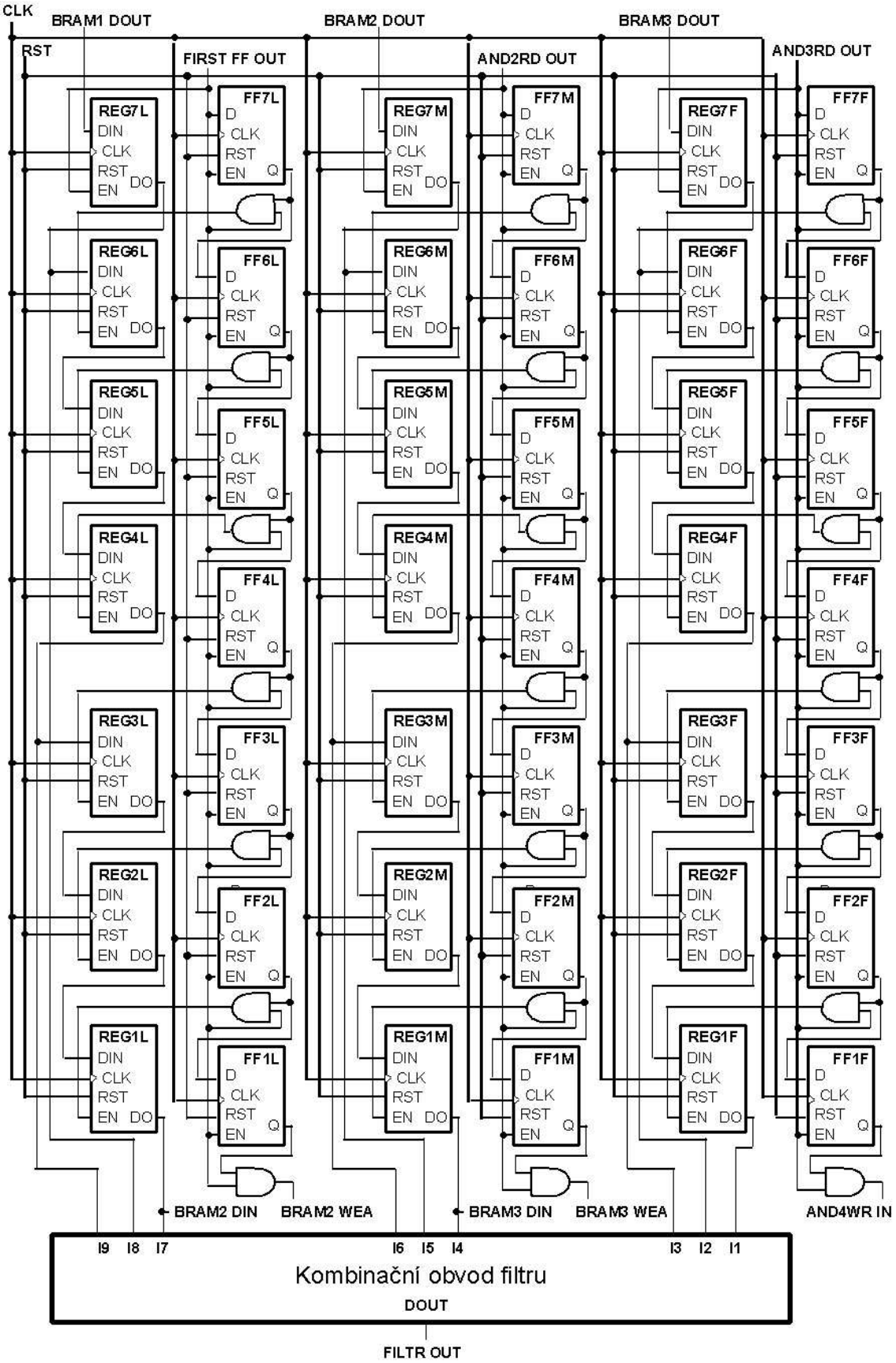
Příloha č. 1: Detailní schéma zapojení řetězené datové linky hardwarových filtrů

Příloha č. 2: Použitá nerekurzivní verze algoritmu Find

Příloha 1


MCU
CLK
RST
EN
WRITE COUNTER 1
CLK
RST
EN
DOUT
'1'
SPI_ctrl
CLK
RST
SPI_SCK
SPI_CS
SPI_MOSI
SPI_MISO
CS
DO
DO_VLD
DI
DI_REQ
SPI_adc
CLK
CS
DO
DO_VLD
DI
DI_REQ
ADDR
DATA_OUT
WE
DATA_IN
RE
=?
BRAM1
DIA
DIPA
ADDRA
WEA
ENA
SSRA
CLKA
DIB
DIPB
ADDRB
WEB
ENB
SSRB
CLKB
DOA
DOPA
DOB
DOPB
'0'
WRITE COUNTER 4
CLK
RST
EN
DOUT
SPI_adc
CLK
CS
DO
DO_VLD
DI
DI_REQ
ADDR
DATA_OUT
WE
DATA_IN
RE
=?
'1'
READ COUNTER 1
CLK
RST
EN
DOUT
WRITE COUNTER 2
CLK
RST
EN
DOUT
FIRSTFF
D
CLK
RST
EN
Q
'1'
BRAM2
DIA
DIPA
ADDRA
WEA
ENA
SSRA
CLKA
DIB
DIPB
ADDRB
WEB
ENB
SSRB
CLKB
DOA
DOPA
DOB
DOPB
'0'
RDCOUNTER4 OUT
WRCOUNTER4 OUT
COMPAR4RD OUT
WRCOMP4 OUT
AND4WR OUT
SPI RE
SPI DI
=?
'ROW BYTES - 8'
READ COUNTER 2
CLK
RST
EN
DOUT
RST
CLK
BRAM2 DIN
BRAM2 WEA
FIRST REG OUT
AND2RD OUT
BRAM2 DOUT
BRAM1 DOUT


WRCOUNTER4
OUT
WRCOMP4
OUT
SPI RE
RDCNTR4
OUT
BRAM4
DOUT
AND4WR IN
WRITE
COUNTER 3
CLK
RST
EN
DOUT
BRAM4
DIA
DIPA
ADDRA
WEA
ENA
SSRA
CLKA
DOA
DOPA
'0'
DIB
DIPB
ADDRB
WEB
ENB
SSRB
CLKB
DOB
DOPB
BRAM3
DIA
DIPA
ADDRA
WEA
'0'
ENA
SSRA
CLKA
DOA
DOPA
DIB
DIPB
ADDRB
WEB
ENB
SSRB
CLKB
DOB
DOPB
BRAM3 DOUT
=?
=?
'ROW BYTES - 8'
READ
COUNTER 4
CLK
RST
EN
DOUT
READ
COUNTER 3
CLK
RST
EN
DOUT
BRAM3 WEA
FILTR OUT
RST
COMPAR4RD
OUT
BRAM3 DIN
CLK


CLK
BRAM1 DOUT
BRAM2 DOUT
BRAM3 DOUT
RST
FIRST FF OUT
AND2RD OUT
AND3RD OUT
REG7L
REG6L
REG5L
REG4L
REG3L
REG2L
REG1L
FF7L
FF6L
FF5L
FF4L
FF3L
FF2L
FF1L
REG7M
REG6M
REG5M
REG4M
REG3M
REG2M
REG1M
FF7M
FF6M
FF5M
FF4M
FF3M
FF2M
FF1M
REG7F
REG6F
REG5F
REG4F
REG3F
REG2F
REG1F
FF7F
FF6F
FF5F
FF4F
FF3F
FF2F
FF1F
DIN
CLK
RST
EN
DO
D
Q
BRAM2 DIN
BRAM2 WEA
BRAM3 DIN
BRAM3 WEA
AND4WR IN
I9 I8 I7
I6 I5 I4
I3 I2 I1
Kombinační obvod filtru
DOUT
FILTR OUT

Příloha 2

```
unsigned char Find(unsigned char *addr)
{
    M1_cnt=0;
    M2_cnt=0;
    M3_cnt=0;
    ptr= addr;
    s=ptr[3+PIC_WIDTH_BYTES];

    //zarazeni prvniho bodu okoli do nektere z mnozin M1 M2 M3
    if(ptr[0]<s) {M1_cnt++;M1[M1_cnt]=ptr[0];}else
    if(ptr[0]==s) {M2_cnt++;M2[M2_cnt]=ptr[0];}else
    if(ptr[0]>s) {M3_cnt++;M3[M3_cnt]=ptr[0];}
    //zarazeni druheho bodu okoli do nektere z mnozin M1 M2 M3 -
    //atd.
    if(ptr[3]<s) {M1_cnt++;M1[M1_cnt]=ptr[3];}else
    if(ptr[3]==s) {M2_cnt++;M2[M2_cnt]=ptr[3];}else
    if(ptr[3]>s) {M3_cnt++;M3[M3_cnt]=ptr[3];}

    if(ptr[6]<s) {M1_cnt++;M1[M1_cnt]=ptr[6];}else
    if(ptr[6]==s) {M2_cnt++;M2[M2_cnt]=ptr[6];}else
    if(ptr[6]>s) {M3_cnt++;M3[M3_cnt]=ptr[6];}
    ptr=ptr+PIC_WIDTH_BYTES;
    if(ptr[0]<s) {M1_cnt++;M1[M1_cnt]=ptr[0];}else
    if(ptr[0]==s) {M2_cnt++;M2[M2_cnt]=ptr[0];}else
    if(ptr[0]>s) {M3_cnt++;M3[M3_cnt]=ptr[0];}

    if(ptr[3]<s) {M1_cnt++;M1[M1_cnt]=ptr[3];}else
    if(ptr[3]==s) {M2_cnt++;M2[M2_cnt]=ptr[3];}else
    if(ptr[3]>s) {M3_cnt++;M3[M3_cnt]=ptr[3];}

    if(ptr[6]<s) {M1_cnt++;M1[M1_cnt]=ptr[6];}else
    if(ptr[6]==s) {M2_cnt++;M2[M2_cnt]=ptr[6];}else
    if(ptr[6]>s) {M3_cnt++;M3[M3_cnt]=ptr[6];}
    ptr=ptr+PIC_WIDTH_BYTES;
    if(ptr[0]<s) {M1_cnt++;M1[M1_cnt]=ptr[0];}else
    if(ptr[0]==s) {M2_cnt++;M2[M2_cnt]=ptr[0];}else
    if(ptr[0]>s) {M3_cnt++;M3[M3_cnt]=ptr[0];}

    if(ptr[3]<s) {M1_cnt++;M1[M1_cnt]=ptr[3];}else
    if(ptr[3]==s) {M2_cnt++;M2[M2_cnt]=ptr[3];}else
    if(ptr[3]>s) {M3_cnt++;M3[M3_cnt]=ptr[3];}

    if(ptr[6]<s) {M1_cnt++;M1[M1_cnt]=ptr[6];}else
    if(ptr[6]==s) {M2_cnt++;M2[M2_cnt]=ptr[6];}else
    if(ptr[6]>s) {M3_cnt++;M3[M3_cnt]=ptr[6];}

    unsigned char count;
    unsigned char *ptr;
    int k;
    int index=5;

    while(1)
```

```
    {
       //rozdeleni mnoziny M
       if((M1_cnt+M2_cnt)<index)
       {
          ptr=M3;
          M3=address_free;
          address_free=ptr;
          index=index-M1_cnt-M2_cnt;
          count=M3_cnt;
          M1_cnt=0;
          M2_cnt=0;
          M3_cnt=0;
       }
       else if(M1_cnt<index) return (s);
       else
       {
          ptr=M1;
          M1=address_free;
          address_free=ptr;
          count=M1_cnt;
          M1_cnt=0;
          M2_cnt=0;
          M3_cnt=0;
       }
       s=ptr[index];
      //zarazeni bodu okoli do nektere z mnozin M1 M2 M3
       for(k=1;k<=count;k++)
       {
          if(ptr[k]<s) {M1_cnt++;M1[M1_cnt]=ptr[k];}else
          if(ptr[k]==s) {M2_cnt++;M2[M2_cnt]=ptr[k];}else
          if(ptr[k]>s) {M3_cnt++;M3[M3_cnt]=ptr[k];}
       }
    }

}
```