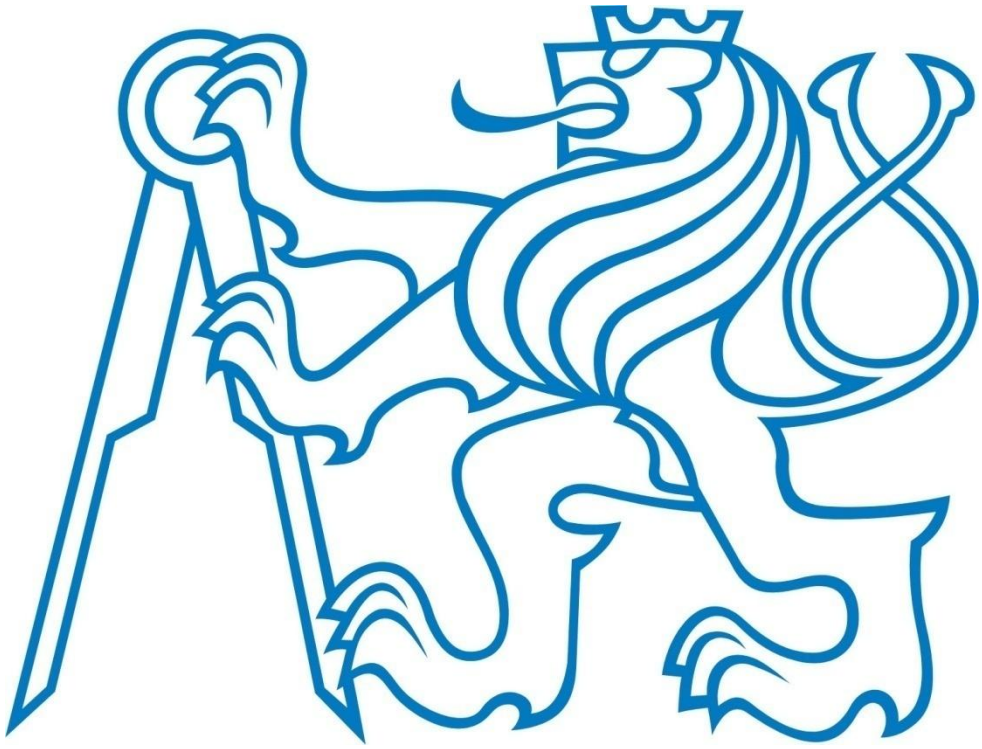
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra měření



Obrazový modul s FPGA

Diplomová práce

Studijní program: Kybernetika a robotika
Studijní obor: Senzory a přístrojová technika

Vedoucí práce: doc. Ing. Jan Fischer, CSc.

Bc. Staněk Jan

Praha 2015

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra měření

Akademický rok **2013-14**

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	**Bc. Jan Staněk**
Studijní program:	**Kybernetika a robotika**
Obor:	**Senzory a přístrojová technika**
Název tématu česky:	**Obrazový modul s FPGA**
Název tématu anglicky:	**FPGA Based Image Acquisition Module**

Pokyny pro vypracování:

Navrhněte a realizujte obrazový modul, který bude podporovat připojení řádkových senzorů CCD a plošných obrazových senzorů CMOS. Použité hradlové pole - FPGA firmy Xilinx – má zajistit jak řízení obrazových senzorů, tak i zpracování jejich signálů. Obrazový modul má umožnit nejen autonomní rychlé zpracování obrazu, ale i činnost ve spolupráci s nadřazeným PC, které bude sloužit pro nastavování parametrů modulu a pro záznam sejmutého obrazu. Dále vytvořte programové vybavení pro FPGA a potřebnou aplikaci pro nadřazené PC.

Seznam odborné literatury:

[1] Xilinx.: Spartan-3 Generation FPGA User Guide, UG331, v1.8, 2011

[2] Kovács, P.: Diplomová práce, ČVUT- FEL, Praha

[3] Fischer, J.: Optoelektronické senzory a videometrie. Skripta ČVUT - FEL, Praha

Vedoucí diplomové práce:	doc. Ing. Jan Fischer, CSc.
Datum zadání diplomové práce:	25. listopadu 2013
Platnost zadání do[1]:	31. srpna 2015

L.S.

Prof. Ing. Vladimír Haasz, CSc.
vedoucí katedry

Prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 25. 11. 2013

[1] Platnost zadání je omezena na dobu tří následujících semestrů.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 5.1.2015 ……

Staněk Jan

Poděkování

Na tomto místě děkuji doc. Ing. Janu Fischerovi, CSc. za odborné vedení práce, cenné rady a vstřícnost v poskytnutých konzultacích, také děkuji své rodině za celoživotní podporu během studia. Dále bych rád poděkoval Ing. V. Kastnerovi a firmě ConTeK s.r.o. za vstřícnost a pracovní zkušenosti získané během studia.

Abstrakt

Cílem této práce je navrhnout a realizovat modul pro rychlé zpracování obrazu z CMOS snímače s využitím prostředků hradlových polí Xilinx řady Spartan 3. Práce začíná realizací přenosu obrazových dat do PC skrze EZ-USB periferii v mikropočítači Cypress a jejich zobrazením, pak pokračuje návrhem algoritmu pro zpracování obrazu a jeho implementací do hradlového pole. Dále byla řešena aplikace pro nadřazené PC v jazyce C# a autonomní start modulu. V jazyce VHDL je navržen a realizován algoritmus určování objektů založený na metodách labelingu s minimalizací nároků na potřebnou paměť a počet taktů hodinového signálu. Závěrem prací bylo využito 8051 kompatibilního jádra EZ-USB mikropočítače ke čtení souborového systému FAT z vyměnitelného úložiště formátu Secure Digital a inicializaci modulu z uložených souborů.

Abstract

Aim of this thesis is to develop and implement module to fast image data processing using Xilinx Spartan 3 FPGA resources. Task started with transferring and displaying image data in PC using EZ-USB peripheral controller Cypress, then continues with designing image processing algorithm and its adjustment to FPGA logic. The next part is solving PC application in C# language and dealing with autonomous module startup. In VHDL is designed and implemented labeling-based algorithm optimized to memory usage and processing speed. Task is finished by programming 8051 compatible core in EZ-USB controller to read FAT filesystem in Secure Digital memory card and to boot entire module from stored configuration files.

0.1 Seznam tabulek

0.2 Seznam obrázků

1. Úvod

V dnešní době se již běžně setkáváme s aplikacemi strojového vnímání s užitím kamerových systémů. Nejčastěji najdeme užití pro bezkontaktní měření, rozpoznávání objektů nebo kontrolu a řízení procesů. Informace z obrazového snímače, typicky CMOS či CCD, je konvenčními metodami zpracovávána jako celek, tj. po celistvých snímcích.

Zpracování obrazu se v různých aplikacích liší, ale jako jádro můžeme považovat rozpoznávání zadaných vzorů, určování jejich polohy/rozměru či jejich porovnání s referenčním vzorem a vyhodnocování shody. Zpracování obrazu touto cestou může být robustní, spolehlivé či jednodušší z pohledu implementace, ale nelze opomenout celou řadu jeho nevýhod, zejména připomeňme paměťovou a výpočetní náročnost, která znemožňuje nasazení všude tam, kde chceme rychlé zpracování obrazové informace, ale zároveň nemůžeme či nechceme do řešení integrovat drahé komponenty s potřebným výkonem.

Stále více se setkáváme se zpracováním obrazu on-the-fly[2], kde není obrazová informace uložena do paměti, ale okamžitě zpracována. Toto řešení rapidně snižuje nárok na ukládání dat, naproti tomu ale klade nárok na rychlost jejich zpracování, zejména v aplikaci kde nemůže zdroj informace čekat na dokončení předchozího výpočtu. Rychlosti nutné ke zpracování obrazu, zejména ve vysokém rozlišení, je však prakticky nemožné dosáhnout sekvenčním výpočtem.

Stejně jako v moderní výpočetní technice je i zde správnou volbou rozdělit zpracovávání dat na co možná největší počet souběžně běžících procesů a dosáhnout tak vysoké míry paralelismu. Z principu funkce se jasně nabízí použít hradlová pole, kde paralelismus je jejich předností. Obvody realizované hradlovým polem mohou ve stejném čase data přijímat, vyhodnocovat přítomnost objektů ve snímané scéně a tyto objekty dále zpracovávat.

Jak již bylo uvedeno úvodními odstavci, je záměrem využít hradlová pole k rychlému zpracování obrazových dat. Težko lze ale vyvíjet zařízení, o jehož funkci nemáme ucelenou představu nebo si nejsme jisti, proč jsou požadavky kladeny v patřičné výši. Znalost konkrétního problému také může sama zodpovědět nějaké otázky, které mohou v průběhu práce vyvstat. Zpracování obrazu je široký pojem, stanovme si tedy jednu z možných aplikací jako motivační úlohu, do které bychom náš produkt mohli potřebovat.

Nechť motivační úlohou je určování objektů na snímané scéně, která se velmi rychle mění. Takovým případem může být i sledování dopravníkového pásu a řízení manipulátoru pro práci s přepravovanými předměty. Tato úloha v sobě zahrnuje nároky na zpracování obrazu do podoby čitelných a stručných informací, například poloha těžiště a rozměry předmětu, ale stejně tak na rychlost a aktuálnost vypočtených informací. (Chceme informaci kde se předmět právě nachází, ne kde byl před okamžikem). Formulujme tedy motivační úlohu do podoby jednoznačných požadavků.

[2] „On the fly“ – www.freethesaurus.org/dictionary : za běhu, ve spěchu, během pohybu.
V kontextu výpočetního zpracování rozumíme okamžitý výpočet, pro který v daném okamžiku není třeba znát (a ukládat) celistvá data.

2. Stanovení cílů práce

Jedním z možných pohledů na stanovení cílů je splnění motivační úlohy, nicméně je vhodné dát našemu řešení konkrétní rozměr. Pokusme se v následující práci o splnění těchto kritérií.

- Metodika zpracování obrazu by měla být použitelná pro signály z řádkového i plošného obrazového snímače. Nechť maximální délka řádku je v rozlišení 4096 obrazových bodů a uvažujme nezávislost algoritmu na počtu řádků.
- Zpracování by nemělo zpomalovat snímání obrazu.
- Výstupem zpracování bude množina dat reprezentující nalezené objekty, pro zajištění funkce uvažujme rozpoznávání do počtu minimálně 16 disjunktních objektů. Umožní-li to komponenty, bude snaha tento počet navýšit.
- Modul bude pracovat autonomně, bude možné ho konfigurovat nadřazeným PC a zpracovaná data budou dostupná i skrze komunikaci aplikovatelnou v průmyslu.
- Pro účely konfigurace bude možné zobrazit snímaný obraz v nadřazeném PC.
- Modul by měl být do budoucna připraven pro nahrávání aplikace z externího média, ideálně implementací SD paměťových karet a souborového systému FAT.
- Zpracování by nemělo být příliš náročné a mělo by pracovat i na malých (a levných) hradlových polích.

3. Rozbor řešení

Představme si řetězec tvořený výše uvedenými komponenty. Klíčové články seřadíme tak, jak budou data postupovat. Na samém počátku budou data generována obrazovým snímačem, poté zpracována hradlovým polem a v závěru pomocí mikropočítače odeslána do nadřazeného PC. U generování dat naši práci začneme. Již výše bylo nastíněno použití obrazového snímače typu CMOS, na ten se zaměříme v následující podkapitole.

3.1 Snímání obrazu

3.1.1 CMOS obrazový snímač

Návrh, realizace a testování bude probíhat primárně pro CMOS obrazový snímač Aptina MT9V032C. Úvod do problematiky obrazových snímačů je přehledně popsán v předchozí práci (V.Nádvorník, [4]), zde pouze shrneme krátký výčet ze specifikací výrobce [7].

- Optický formát: $^1/_3$ palce (11:7)
- Rozlišení: Formát Wide-VGA, 752 x 480 aktivních obr. bodů
- Typ barev: Černobílý, maximální citlivost pro 550nm
- Aktivní plocha: 6.0 x 6.0 um pixel, 4.51 x 2.88 mm snímek
- Závěrka: Global shutter
- Snímková frekvence: 60 snímků za sekundu v plném rozlišení
- Rychlost čtení: 13 – 27 MHz @ Pixel
- ADC převodník: Interní 12 bit
- Požadavky napájení: 3.3 V, 200 mA
- Pouzdro: 48 pin CLCC
- Konektivita: Konfigurace - I^2C
  Data - 10 bit paralelní sběrnice
- Funkce: Výběr aktivní oblasti, vynechání řádků/sloupců, externí trigger, zrcadlení, generování testovacích dat

3.1.2 Nastavení obrazového snímače

Obrazový snímač je možné provozovat v mnoha různých režimech a nastaveních. Ještě před započetím prací je dobré prodiskutovat všechny přípustné varianty a zvolit výchozí řešení.

Prakticky nejpodstatnější bude správná orientace obrazu. Již od počátku televizní techniky je obraz vykreslován po řádcích, s počátkem v levém horním rohu obrazovky. Tento smysl se nezměnil ani s nástupem výpočetní techniky a budeme se ho nadále držet i v této práci. Orientaci snímku ilustruje fig. 3.1. Na první pohled se může zdát, že počátek snímku v pravém horním rohu neodpovídá našemu požadavku. V tomto směru je ilustrace zavádějící, neboť neukazuje orientaci snímku, ale orientaci obrazu promítnutého na plochu snímače.

Otočíme-li snímač jeho plochou směrem ke snímané scéně, obraz se překlopí a jeho počátek bude právě v levém horním rohu. Uvážíme-li i přetočení obrazu na čočce objektivu, dojde k otočení obrazu o 180° kolem jeho středu. Rotace obrazu již smysl čtení nemění, záleží pak pouze na umístění snímače, zda bude žádoucí obraz otočit zpět či nikoliv. Otáčení obrazu je možné nastavit v registru **„Read Mode 2“**.

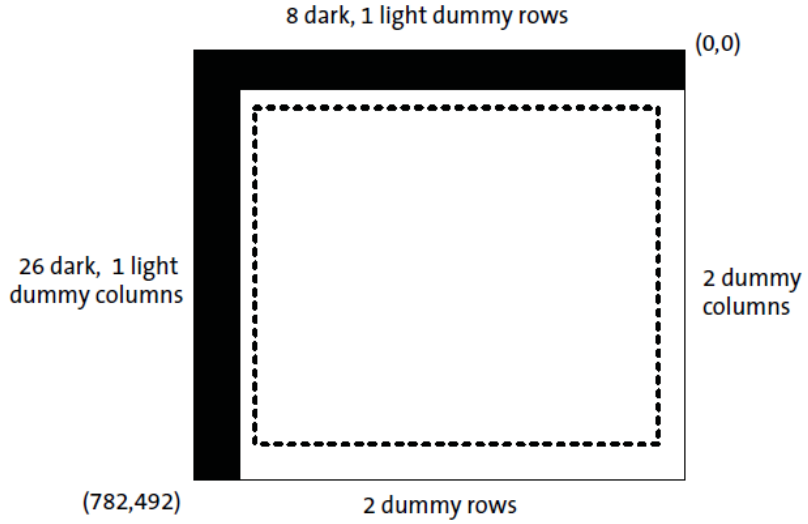


Fig. 3.1 Uspořádání snímku [7]

Nastavení aktivního okna snímku je podstatné s ohledem na formu jeho zpracování. Nechceme-li zanášet prodlevu mezi čtení řádků/snímků můžeme využít dobu čtení zatemněných obrazových bodů a zachovat tak kontinuální čtení dat. Úroveň jasu zatemněných bodů je nulová nebo přinejmenším podstatně nižší než může být v aktivních obrazových bodech, toho by bylo možno využít i pro řádkovou a snímkovou synchronizaci, podobně je do obrazu vložena synchronizace v televizní technice. O konfiguraci synchronizačních signálů bude psáno níže v kapitole „Čtení dat obrazu“.

Nejprve bude snímač využit v jeho plném rozlišení. Bude-li dosaženo uspokojivých výsledků, pokusíme se snížit rozměr aktivního okna a celý systém zrychlit. Posledním aspektem obrazu je nastavení jeho úrovní, to lze úpravou zesílení či posunutím jasové složky. Pro dosažení lepších vlastností a přizpůsobivosti na různé světelné podmínky lze aktivovat automatické zesílení a řízení doby expozice (AGC a AEC). Funkce po aktivaci v registru **„AGC/AEC Enable“** přizpůsobují snímaný obraz a vyrovnávají jeho jas dle hodnoty registru **„AEC/AGC Desired Bin“**. Výchozí nastavení tohoto registru je téměř v jeho maximu a bude pravděpodobně nutné ho adekvátním dílem snížit. Všechny konfigurační parametry budou uloženy do nevolatilní paměti, ze které se po startu zařízení snímač nakonfiguruje. Samozřejmě budou přístupné z nadřazeného PC, kterým se budou tyto parametry upravovat.

3.1.3 Signály obrazového snímače, časování

Než budou popsány jednotlivé prostředky, je vytvořen přehled všech vývodů, které jsou těmto účelům určeny. Statické signály jsou pro jednoduchost vynechány.

Pin	Orientace	Popis
SCLK	Vstup	Hodinový signál sériové komunikace
SDATA	Obousměrný	Datový vodič sériové komunikační sběrnice
CLKIN	Vstup	Hodinový signál pro funkci snímače
DOUT <0-9>	Výstup	Paralelní výstup obrazových dat
FRAME_VALID	Výstup	Signalizace platných dat v rámci snímku
LINE_VALID	Výstup	Signalizace platných dat v rámci řádku
PIXCLK	Výstup	Hodinový signál generovaný synchronně s daty

Tab. 3.1 Vstupní a výstupní piny komunikace

3.1.3.i Konfigurace CMOS po I²C

První diskutovaná komunikace CMOS snímače je sběrnice protokolu I^2C (IIC). Kompletní dokumentace protokolu [8] je dostupná i z webových stránek nxp.com, nicméně přiblížíme si konkrétní příklad pro konfiguraci snímače.

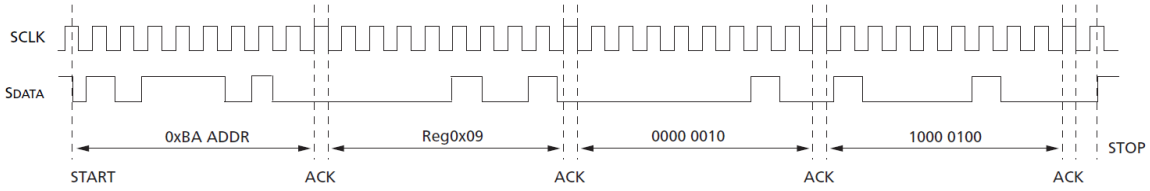


Fig. 3.2 Průběh zápisu do snímače po I^2C sběrnici [7]

Tento obrazový snímač se na sběrnici identifikuje pod adresou 0x5D. Na snímku fig. 3.2 je zakreslen průběh zápisu konfiguračního slova 0x0284 do registru 0x09. Z pohledu časování je kromě definovaných START a STOP podmínek jakožto sekvence hran určena perioda hodinových pulzů. Frekvence hodinových pulzů SCLK musí být vyšší než 10kHz [8] a nižší než cca 1/20 CLKIN [7]. Vydáme-li se standardní cestou, bude pro naše účely vhodnou frekvencí 100kHz (*I^2C standard mode*).

Formát zpráv, kterými se budou konfigurační slova zapisovat, je jednoduchý. První byte je vždy 7 bitů adresy zařízení a příznak čtení/zápis, následuje adresa registru, do kterého chceme zapsat a jeho 16 bitový obsah. Hodnota je odeslána vyšším bytem napřed. Zápis nové hodnoty se uskuteční pouze v případě úspěšného příjmu obou datových bytů. Dojde-li k úspěšnému zápisu přijatých dat, je vnitřní čítač adresy inkrementován a lze odeslat data do následujícího registru bez nutnosti znovu inicializovat transakci zápisu. Takto je možno během jediné transakce zapsat do paměti celý blok.

3.1.3.ii Čtení dat obrazu

Nyní se podíváme podrobněji na zbylé signály z tabulky tab. 3.1. Data ze snímače jsou vedena paralelní sběrnicí DOUT, synchronní dle signálu PIXCLK a jsou platná při jeho sestupné hraně. Snímač dále poskytuje signály řádkové a snímkové synchronizace LINE VALID a FRAME VALID. Jejich očekávaný průběh zachycuje ilustrace fig. 3.3. Proporce signálu PIXCLK mají pouze ilustrativní charakter a neodrážejí skutečný počet hodinových cyklů.

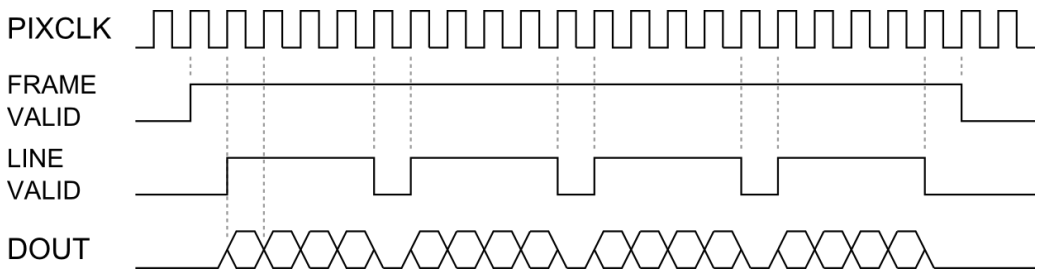


Fig. 3.3 Očekávaný tvar výstupních signálů CMOS snímače

Úroveň synchronizačních signálů odráží skutečnost, zda se obrazový bod nachází v aktivní ploše snímaného obrazu nebo v zatemněných obrazových bodech. Pokud je signál FRAME VALID v logické 1, nachází se v aktivním okně obrazu a získáváme obrazová data. V opačném případě se nachází ve snímkovém zatemnění a výstupní hodnota je nulová. Signál LINE VALID plní podobnou funkci, ale uvnitř každého řádku. Rozlišujeme tedy podle něj aktivní oblast řádku (signál v log. 1) a řádkové zatemnění (log. 0). Tvar signálu řádkového zatemnění je možné ovlivnit konfigurací v registru „**Read Mode 2**“. Výchozí volba ponechává signál LINE VALID neaktivní během celé doby snímkového zatemnění, signál si ale lze zprostředkovat volbou „**Continuous LINE_VALID**“ a mít tak přehled i o zatemněných řádcích.

3.2 Zpracování obrazu

3.2.1 Použité hradlové pole - FPGA

Data získaná z obrazového snímače bude nutno zpracovat, rozlišit aktivní obrazové body od zatemnění, přiřadit jim souřadnice, nalézt v nich klíčové motivy a ze všech dostupných informací vypočíst výsledek. Celý tento proces je výpočetně náročný a běžně dostupné procesory či mikropočítače již nedokážou tento objem dat v reálném čase vyhodnotit. Spoustu rutinních předzpracování je možno uskutečnit na nižší úrovni, respektive na úrovni logických členů.

Pro naše účely byla vybrána hradlová pole firmy Xilinx z řady Spartan-3, zařízení by se mělo osadit polem XC3S200, což dává návrhu následující možnosti a omezení:

- Počet hradel: 200K
- Logických buněk: 4320
- Logických bloků: 480
- RAM paměť: 216Kbit + 30Kbit distribuovaná
- Pouzdro: VQ100, TQ144

3.2.2 Metodika zpracování obrazových dat

V rámci zpracování obrazových dat bude klíčové sestavit posloupnost operací, kterou se oddělí obrazy snímaných objektů od pozadí snímané scény. V obraze bude nutno nalézt hrany, ze kterých se složí obrysy objektů. Poté bude nutno vyvinout metodu, jejímž použitím lze z obrysů zrekonstruovat objekt a určit jeho vlastnosti. Vzorem takové metody by se mohl stát některý z algoritmů Labelingu[3].

[3] „Connected-component labeling“– Algoritmus vycházející z teorie grafů, který hledá souvislé regiony v binárním obraze.

Doposud hodnocené algoritmy jsou vyhovující z hlediska robustnosti, nicméně jsou jen těžko aplikovatelné pro prostředky hradlových polí. Obsahují spoustu sekvenčního zpracování, vyžadují více průchodů zdrojovými daty nebo implementují složité rozhodovací stromy. Vhodnou implementací je možno eliminovat většinu náročných operací bez vážné újmy na robustnosti, nicméně úplně se jim vyhnout nelze. V této práci budeme hledat vhodný kompromis napříč zmíněnými metodami.

3.2.3 Návrh logiky pro FPGA

Pro aplikaci hradlových polí je k dispozici volně dostupné vývojové prostředí ISE Design Suite ve verzi 14.6. Návrh probíhá na úrovni schémat a jazyka VHDL. V jazyce VHDL budou popsány především matematické operace a složitější bloky, schematický návrh pak bude sloužit pro propojení všech takovýchto bloků a drobné logické funkce. Hotový návrh je nutno opatřit definicemi vstupů a výstupů, to lze uskutečnit ručně nebo použít nástroj PlanAhead.

Nástroje vývojového prostředí umožňují i simulaci návrhu nebo jeho částí. Této funkcionality bude využíváno pro ověření matematických operací, funkce paměti, ladění vstupních/výstupních bloků apod.

Z návrhu, který úspěšně projde všemi kroky syntézy, je vygenerován programovací soubor obsahující tzv. „konfigurační bitstream [4] “. Bitstream je nutno nahrát do hradlového pole při každém jeho startu. V rámci vývoje a testování se nabízí použití rozhraní JTAG, nebo v případě moderních vývojových kitů i přístup jako k USB zařízení. Ve výsledné aplikaci ale budeme požadovat autonomní start zařízení bez účasti PC, programovací soubor bude nutné uložit do paměti a při startu z něj hradlové pole inicializovat.

3.2.4 Bootování hradlového pole

3.2.4.i Možnosti bootování hradlového pole

Obraz návrhu[4] je možné do hradlového pole nahrát celkem 8 různými způsoby. Každý případ najde využití v jiné aplikaci, zohlednit je potřeba především šířku sběrnice, přítomnost hostitele a zdroj konfiguračního souboru. V přehledu se jedná o tyto možnosti [9]:

- Master serial
- Master SPI
- Master BPI
- Master parallel
- Internal master SPI
- JTAG
- Slave parallel
- Slave serial

[4] „Bitstream“ – Složenina slov bit a tok. V této složené formě nejčastěji chápána jako výraz pro konfigurační data FPGA. Zakládá se na způsobu bootování FPGA, kde je binární obraz návrhu přenášen ze sériové PROM nebo FLASH paměti jako souvislý tok bitů. V kontextu hradlových polí považujme za rovnocenné i výrazy „konfigurační data“ nebo „obraz návrhu“.

Pro první čtyři možnosti se hradlové pole chová jako hostitel, čte data z externí paměti a konfiguruje se bez nutnosti externích členů. V pátém případě „Internal master SPI“ je konfigurační paměť umístěna přímo uvnitř pouzdra FPGA. Tyto možnosti konfigurace jsou využity především v aplikaci, kde je hradlové pole konfigurováno pouze z jednoho možného zdroje a je nadřazené všem ostatním součástem zařízení.

Konfigurace pomocí rozhraní JTAG slouží především pro testování návrhu ve fázi vývoje, uplatnění pro finální aplikaci ale příliš nenachází.

Posledními možnostmi jsou režimy „Slave“, ve kterých se do hradlového pole konfigurační data nahrávají pomocí nadřazeného hostitele, nejčastěji mikropočítače. Tyto možnosti nás zajímají, neboť při startu zařízení můžeme rozhodnout, jakým obrazem budeme hradlové pole konfigurovat. Dle šířky sběrnice můžeme vybrat sériové či paralelní rozhraní, s ohledem na počet použitých pinů a skutečnost, že i s externí pamětí budeme komunikovat sériově, je režim „***Slave serial***“ nejlepší možná volba.

3.2.4.ii Bootování – Slave serial

Tato možnost je jedna z nejjednodušších, nejužitečnějších a zároveň je implementována na všech hradlových polích řady Spartan-3. Uveďme si přehled pinů, které budeme během konfigurace používat.

Pin	Orientace	Popis
HSWAP	Vstup	Povolení pull-up rezistorů na I/O pinech. Aktivní v 0.
M[2:0]	Vstup	„Mode select“ – Kombinace na této bráně volí režim konfigurace hradlového pole. Pro „Slave serial“ volíme M[2:0] = <1,1,1>
DIN	Vstup	Sériová data od hostitele, synchronní s CCLK.
CCLK	Vstup	Hodinové impulzy sériové komunikace
INIT_B	Obousměrný	Indikátor inicializace, aktivní v 0. Před inicializací indikuje mazání paměti, během inicializace signalizuje chybu CRC.
DONE	Obousměrný	Indikátor dokončení konfigurace. Logická 0 indikuje probíhající konfiguraci.
PROG_B	Vstup	Příkaz konfigurace, aktivní v 0. Restartuje konfiguraci hradlového pole a smaže jeho konfigurační paměť.

Tab. 3.2 Piny konfigurace FPGA

Nejprve je určena úroveň statických signálů. Nastavením všech pinů M[2:0] do úrovně log.1 je vybrán režim „Slave serial“. Signál HSWAP slouží během konfigurace k nastavení pull-up rezistorů na všech I/O pinech, je-li žádoucí aby v naší aplikaci byly pull-up rezistory připojeny je nutné držet tento signál v log.0 během celého procesu inicializace.

Sériové rozhraní je kompatibilní se standardem SPI v konfiguraci *CPOL = CPHA = 0* [2].

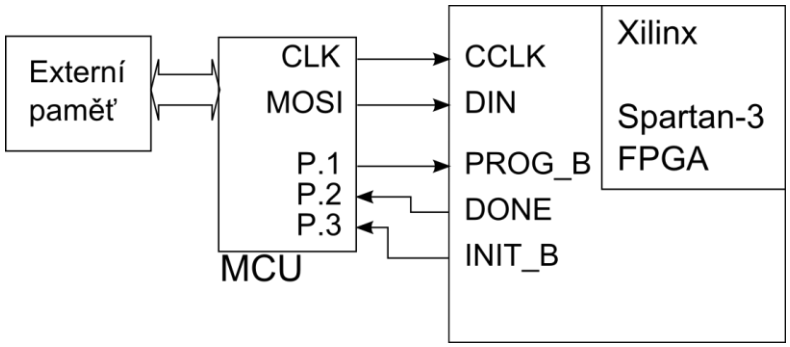


Fig. 3.4 Konfigurace FPGA sériovým rozhraním

3.2.4.iii Konfigurační sekvence

Vlastní inicializace hradlového pole sestává z posloupnosti celkem osmi kroků.

1. Reset
2. Smazání konfigurační paměti
3. Určení režimu konfigurace
4. Synchronizace
5. Kontrola ID
6. Nahrání konfiguračních dat
7. Kontrola CRC
8. Spouštěcí sekvence

Kroky 1-3 nastaví hradlové pole do příslušného režimu, během kroků 4-7 se nahraje příslušná konfigurace.

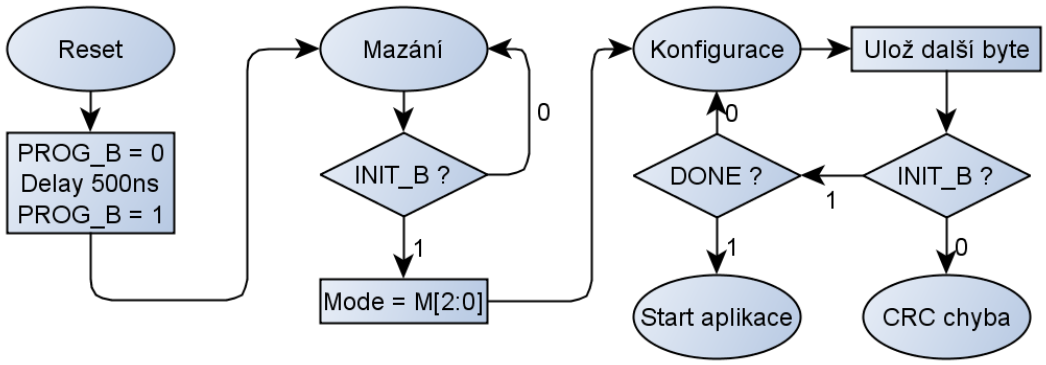


Fig. 3.5 Stavy konfigurace FPGA

Reset je proveden přivedením nulového pulzu na pin PROG_B, pulz musí trvat nejméně 500ns, po navrácení úrovně do log.1 pokračuje proces konfigurace. Po zpracování události resetu se začne mazat předchozí konfigurace, během této doby je výstupní pin INIT_B držen v log.0. Při náběžné hraně INIT_B signálu je čtením pinů M[2:0] zjištěn režim konfigurace, ve kterém proces okamžitě pokračuje.

Synchronizace spočívá v načtení synchronizačního slova, v případě řady Spartan-3 je to 32 bitové slovo 0xAA995566. Toto slovo označuje začátek konfiguračních dat.

Konfigurační data taktéž obsahují identifikaci cílového zařízení, při použití hradlového pole XC3S200 se v datech objeví řetězec 0x1414093. Tento mechanizmus zabrání konfiguraci z nepatřičného zdroje. Po nahrání všech konfiguračních dat je

proveden kontrolní součet. V případě kladného výsledku je konfigurace považována za dokončenou.

Spouštěcí sekvence se provádí velmi jednoduše, na definovaný vstup se přivede hodinový signál. Výchozím vstupem je CCLK pin, nicméně neuškodí prověřit nastavení generátoru konfiguračních dat (parametr *StartupClk*).

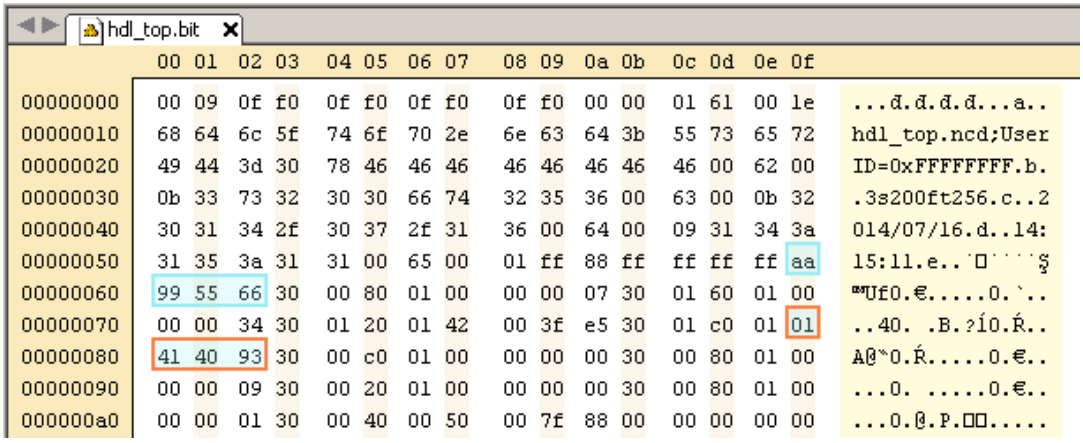


Fig. 3.6 Synchronizační slova v bitstreamu

3.3 Řidicí mikropočítač Cypress Cy7c68013A

V předchozích odstavcích bylo nastíněno, že hradlové pole nebude pracovat autonomně, tato skutečnost je odůvodněna především načítáním konfigurace hradlového pole z externí paměti. K takové činnosti je nutno do návrhu vložit kontrolér, který bude všechny potřebné operace provádět. S potřebou komunikace s nadřazeným PC se nabízí myšlenka sloučit všechny tyto funkce do jednoho mikropočítače. Bohužel implementace USB není příliš jednoduchá a přenos velkého množství dat při rychlostech Hi-Speed[5] by se takto realizoval velmi obtížně. Přistoupíme-li k našim potřebám z opačného konce, najdeme alternativu v podobě obvodu Cy7c68013 firmy Cypress. Tento obvod je primárně řadičem pro Hi-Speed USB, obsahuje ale i jádro mikropočítače 8051, které může od USB periferie pracovat odděleně.

Toto řešení nám umožňuje sestavit plnohodnotnou USB komunikaci, ale zároveň obsluhovat obrazový snímač a hradlové pole při použití pouze jediné komponenty.

[5] Hi-Speed USB – Režim USB dle specifikace 2.0. Pracuje s přenosovou rychlostí 480MBit/s.

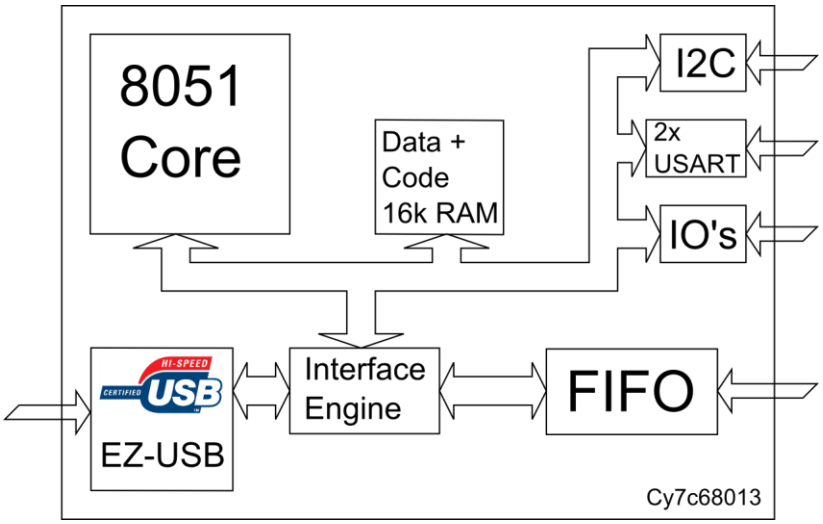


Fig. 3.7 Logické uspořádání v obvodu Cypress

Zjednodušené logické schéma (fig. 3.7 logické uspořádání v obvodu cypress) zachycuje nejdůležitější komponenty obvodu. V horní polovině je znázorněno jádro 8051, jeho paměť a periferie podstatné pro naši aplikaci. I^2C periferie bude sloužit ke konfiguraci obrazového snímače, USART jednotky se uplatní pro nahrávání obrazu do hradlového pole a čtení z externí paměti. Spodní polovina představuje obsluhu USB periferie, ta lze nakonfigurovat k autonomní funkci s řadičem FIFO bez účasti jádra mikropočítače. Nemůže tedy nastat, aby přenos USB transakcí byl vytížením jádra zpomalován.

3.3.1 EZ-USB

Obsažená USB periferie dokáže pracovat až s 8 endpointy, z toho 4 jsou konfigurovatelné v široké škále možností pro použití s FIFO řadičem (EP 2,4,6,8). Zbylé 4 jsou vstupní a výstupní endpointy 0 a 1, do kterých lze přistupovat pouze skrze mikropočítač 8051.

Uveďme si v přehledu, jaké možnosti jsou při výběru endointů k dispozici.

	Velikosti a počet bufferů dle zvolené konfigurace					
No.	**1**	**2**	**3**	**4**	**5**	**6**
EP 2	2x 512	2x 512	2x 512	4x 512	4x 512	4x 512
EP 4	2x 512	2x 512	2x 512	-	-	-
EP 6	2x 512	4x 512	2x 1024	2x 512	4x 512	2x 1024
EP 8	2x 512	-	-	2x 512	-	-
No.	**7**	**8**	**9**	**10**	**11**	**12**
EP 2	2x 1024	2x 1024	2x 1024	3x 512	3x 1024	4x 1024
EP 4	-	-	-	-	-	-
EP 6	2x 512	4x 512	2x 1024	3x 512	-	-
EP 8	2x 512	-	-	2x 512	2x 512	-

Tab. 3.3 Konfigurace paměti endpointů

V tab. 3.3 se můžeme orientovat dle nároků na počet použitých endpointů, nebo na jejich očekávané vytížení. Vidíme, že některé endpointy je možné bufferovat až

čtyřnásobně, což pro vysokou datovou propustnost oceníme. Endpointy 0 a 1 mají ve všech konfiguracích pevně přidělenou paměť o velikosti 64 bytů.

Aplikace by měla vstupní a výstupní endpoint 1 využívat pro komunikaci textových zpráv, jakožto příkazů pro nastavení a řízení, jeden z výše uvedených endpointů (tab. 3.3) pak bude sloužit pro odesílání obrazových dat.

Podrobnější popis problematiky endpointů lze nalézt ve specifikaci výrobce [10] nebo v předchozí práci [5].

3.3.2 Bootování mikropočítače

Uvnitř použitého mikropočítače není obsažena nevolatilní programová paměť. Spuštění procesoru předchází spuštění zavaděče, který zkopíruje spustitelný kód z externí paměti do vnitřní paměti RAM (mimo stavu EA=1) a spustí provádění programu. Spuštění lze provést celkem 4 různými způsoby, v závislosti na obsahu připojené externí I^2C EEPROM. Přítomnost této paměti a její obsah je zavaděčem automaticky zkontrolován po startu obvodu.

Režim	Popis
Výchozí USB zařízení	Obvod se na USB sběrnici přihlásí pod deskriptorem výrobce, pomocí USB lze nahrát kód do RAM paměti a obvod z něj spustit. **Poznámka:** Tento režim není v aplikaci dovoleno používat, neboť se hlásí pomocí VID Cypress.
„C0 Load“	První byte paměti připojené na I^2C sběrnici obsahuje kód 0xC0. Obvod se spustí stejně jako v prvním případě, ale do USB deskriptoru vloží identifikaci uloženou v I^2C paměti.
„C2 Load“	První byte paměti připojené na I^2C sběrnici obsahuje kód 0xC2. Obvod načte program z připojené I^2C paměti do interní RAM a spustí. Za enumeraci USB sběrnice odpovídá uložený firmware.
Externí spuštění EA = 1	Není-li na I^2C sběrnici připojena paměť nebo neobsahuje patřičná data, je přečten stav pinu **EA**. Úroveň log.1 signalizuje vnitřní logice, aby pracovala s externí pamětí připojenou na paralelní sběrnici. Kód je poté spuštěn z této paměti. Vnitřní paměť RAM není v takovém případě využita jako programová a slouží pouze pro ukládání dat. **Poznámka:** Sběrnice a pin EA jsou dostupné pouze ve 128 pinovém pouzdře.

Tab. 3.4 Režimy spuštění obvodu cypress

Program uložený v I^2C paměti musí mít patřičnou formu. Jako první je již diskutovaná identifikace spouštěného režimu, pak následují hodnoty ID pro USB enumeraci (režim „C0 Load“), konfigurační byte a v případě „C2 Load“ režimu i bloky programových dat. Situaci zachycuje tab. 3.5.

Adresa v paměti	Obsah
0	0xC0 / 0xC2
1	USB Vendor ID (spodní byte)
2	USB Vendor ID (horní byte)
3	USB Product ID (spodní byte)
4	USB Product ID (horní byte)
5	USB Device ID (spodní byte)
6	USB Device ID (horní byte)
7	Konfigurační byte
8	Délka bloku (horní byte)
9	Délka bloku (spodní byte)
10	Počáteční adresa bloku (horní byte)
11	Počáteční adresa bloku (spodní byte)
…	Blok Dat
…	
…	Délka bloku (horní byte)
…	Délka bloku (spodní byte)
…	Počáteční adresa bloku (horní byte)
…	Počáteční adresa bloku (spodní byte)
…	Blok Dat
…	
…	0x80
…	0x01
…	0xE6
…	0x00
konec zápisu	0x00

Tab. 3.5 Uložení spouštěcích dat v paměti

Mikropočítač má být spuštěn s námi vytvořeným programem, v průběhu řešení se už budeme zabývat pouze variantou „***C2 Load***“

3.3.3 Inicializace komponent

Mikropočítač obsažený v obvodu Cy7c68013 bude po vlastním startu inicializovat i všechny ostatní komponenty. K tomu bude hojně využito sériových komunikací, které jsou jednoduché, podléhají standardům a na plošném spoji nezabírají jejich spoje příliš místa.

Celý proces je rozdělen do několika po sobě jdoucích operací.

1. Spuštění mikropočítače Cypress z I^2C paměti
2. Spuštění USB komunikace s nadřazeným PC
3. Nalezení programovacího souboru FPGA v externí paměti
4. Nahrání konfiguračních dat do FPGA pomocí SPI
5. Nakonfigurování obrazového snímače přes I^2C

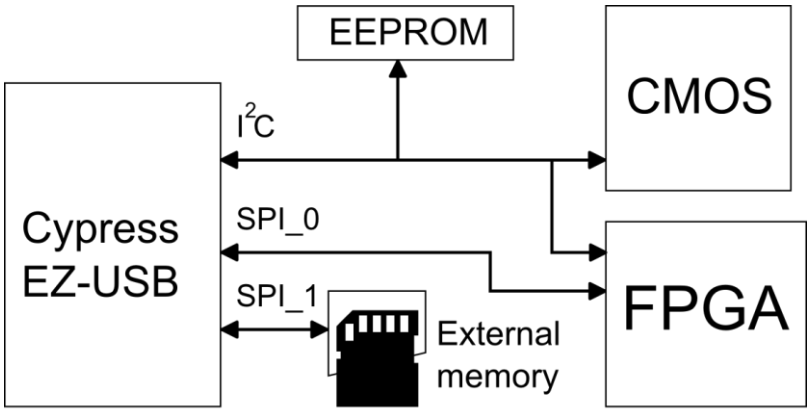


Fig. 3.8 Propojení sběrnic mezi komponenty

3.4 Vyměnitelné úložiště dat formátu Secure Digital

Pro snadnou změnu konfigurace a nastavení je v rámci této práce vyvíjena snaha přesunout všechna data spojená s inicializací na externí médium, které je široce dostupné, malé, levné a bude možné k němu přistupovat běžnými prostředky. Všechny tyto požadavky splňují karty standardu SD[6] [12].

3.4.1 Konektivita SD, signály

Připojení SD karet se liší dle jejich formátu (Full / Mini / Micro) a dle typu použité komunikace. Pro naše účely budeme uvažovat dva možné formáty. První z nich je takzvaný „Full SD“ formát. Jedná se o nejstarší rozměr SD karet, je fyzicky největší ale prakticky nejvíce podporovaný. Druhá varianta je formát „Micro SD“, tento formát se používá do mobilních zařízení a je oblíben právě díky malým rozměrům. Bohužel jen málo universálních čtecích zařízení obsahuje slot tohoto formátu a často je nutné použít kartu s adaptérem pro full-SD slot.

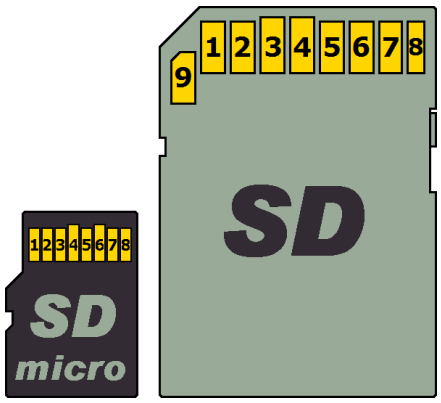


Fig. 3.9 Porovnání formátu Full-SD a MicroSD, číslování kontaktů

Z pohledu transakcí jsou rozlišena tři možná připojení, z toho dvě jsou specifikována asociací SD jako jednobitová nebo čtyřbitová sběrnice. Poslední možností je komunikovat s kartou dle standardu rozhranní SPI, čehož bude využito. Zapojení karty v SPI módu uvádí tab. 3.6.

[6] „Secure Digital“ – Standard spravovaný stejnojmennou asociací. Založeno r. 2000 společnostmi SanDisk, Toshiba a Panasonic. http://www.sdcard.org

Pin SD[7]	Pin µSD	Funkce SPI	Orientace	Popis
1	2	CS	Vstup	Povolovací pin, výběr cílového SPI zařízení.
2	3	MOSI	Vstup	Vysílaná data SPI
3	-	GND	Napájení	Zem
4	4	VCC	Napájení	3.3V napájení
5	5	CLK	Vstup	Hodinový signál SPI
6	6	GND	Napájení	Zem
7	7	MISO	Výstup	Přijímaná data SPI
8	8	-	-	
9	1	-	-	

Tab. 3.6 Vývody pro připojení SD karet v SPI módu

Po připojení je karta konfigurována v režimu SD sběrnice, do SPI režimu přejde uvedením signálu CS do log.0 a odesláním příkazu RESET. Signál CS je uvnitř karty připojen k pull-up rezistoru, nabízí se tak možnost detekovat přítomnost karty sledováním jeho úrovně.

3.4.2 Formát transakcí SD karet

Průběh čtení a zápisu do paměti podléhá stanoveným pravidlům [12] a pouze v rámci podporovaných příkazů. Režim SPI definuje formát příkazu jako zprávu o délce 6 bytů. Příkaz vždy začíná start bitem a transmission bitem, následuje kód příkazu a jeho argument. Poslední byte patří CRC součtu a zakončovacímu bitu. Celý paket znázorňuje fig. 3.10.

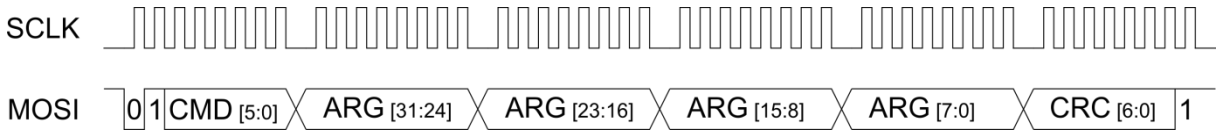


Fig. 3.10 Rámec SD SPI příkazu

Za každým zaslaným příkazem odpovídá SD karta jedním či více byty, ve kterých informuje o stavu zařízení a provádění příkazu. Formát odpovědi záleží na typu příkazu, který byl zaslán. Nejčastější odpověď je formátu R1 o délce jeden byte a obsahuje pouze příznaky chyb [12].

Z pohledu transakcí nastíníme čtení a zápis jednoho bloku dat. Posloupnost je složena z odeslání příkazu, přečtení odpovědi SD zařízení a přenosu datového bloku o stanovené délce. Délka bloku se nastavuje jako argument příkazu CMD16.

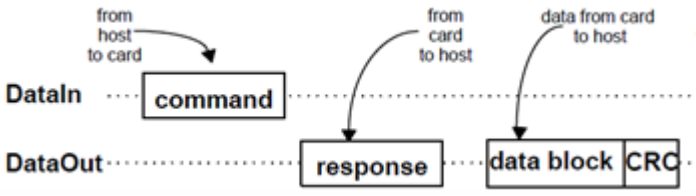


Fig. 3.11 Transakce čtení bloku SD (CMD17) [12]

[7] Zapojení je kompatibilní i pro komunikaci s kartou typu MMC.

Oproti čtení je transakce zápisu zakončena ještě odpovědí zařízení na přijatá data a příznakem vytížení. Příznak je signalizován držením úrovně signálu MISO v log.0 dokud není transakce plně zpracována.

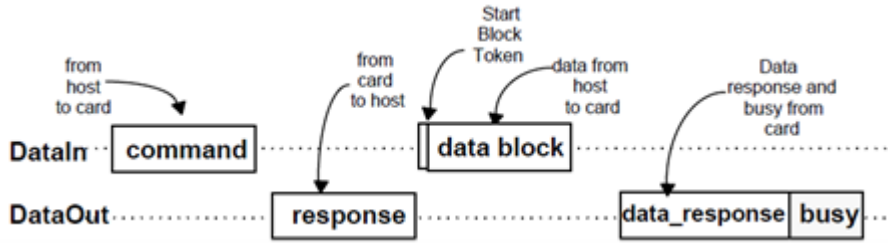


Fig. 3.12 Transakce zápisu bloku SD (CMD24) [12]

3.4.3 Souborový systém

Účelem implementace souborového systému je zpřístupnění dat a konfiguračních souborů uložených na výměnné médium standardní cestou, například uložených z běžného PC ve formátu FAT.

Souborový systém, který bude aplikace obsahovat, by měl být schopen zahájit či ukončit komunikaci s pamětí a přinejmenším zprostředkovávat transakce čtení. Schopnost zapisovat na SD kartu není nezbytně nutná a bude zvažována v průběhu řešení.

Minimální uvažovaná realizace by aplikaci měla poskytnout komfortní nástroj pro pohyb v adresářích a čtení souborů ve struktuře FAT, přičemž by ale neměla být příliš složitá či náročná na prostředky mikropočítače.

4. Řešení úlohy

V první fázi řešení se zabýváme funkčností jednotlivých komponent s využitím již existujících prostředků. Po ověření funkčních celků budou vývojové moduly vzájemně propojeny a řešeny jako celek.

4.1 Propojení komponent

4.1.1 Cypress EZ-USB

Pro vývoj a ladění USB komunikace je použit modul EZ-USB (L.Grepl, [19]), osazený obvodem Cypress Cy7C68013A ve 128 pinovém pouzdře. Potřebné signály včetně konfiguračních jsou vedeny konektorem JP1 a plochým kabelem připojeny k modulu hradlového pole „Spartan3 – Starter kit“ na konektor A2.

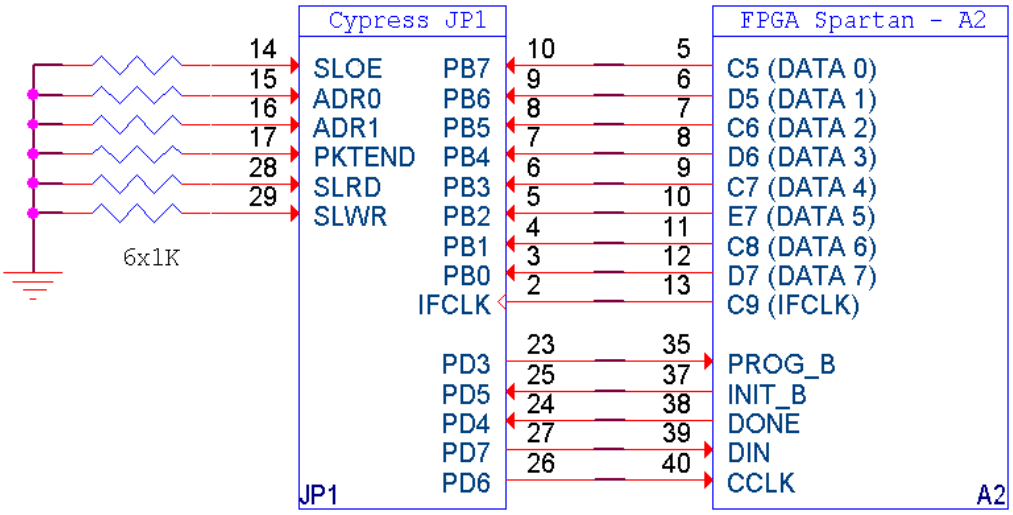


Fig. 4.1 Propojení modulu Cypress USB ke kitu Spartan3

Statické řídicí signály pro obsluhu FIFO jsou přes odpory uvedeny do stavu log.0 a během inicializace jim je přiřazena patřičná polarita. Během inicializace je také provedeno nastavení zdroje hodin pro jádro 8051, přiřazení řadiče FIFO k požadovanému endpointu a zahájení USB komunikace.

Po inicializaci pracuje FIFO řadič autonomně a je konfigurován pro čtení dat při náběžné hraně hodinového signálu, tedy v okamžiku kdy jsou platná data na výstupní bráně hradlového pole.

4.1.1.i Připojení slotu paměťové karty

Připojení paměťové karty bylo zamýšleno na piny SPI periferie, nicméně bylo požadováno využít pouze piny dostupné v 56 pinovém pouzdře, z nichž je dostupná část brány PD a poslední pin brány PA. Slot paměťové karty je připojen dle fig. 4.2.

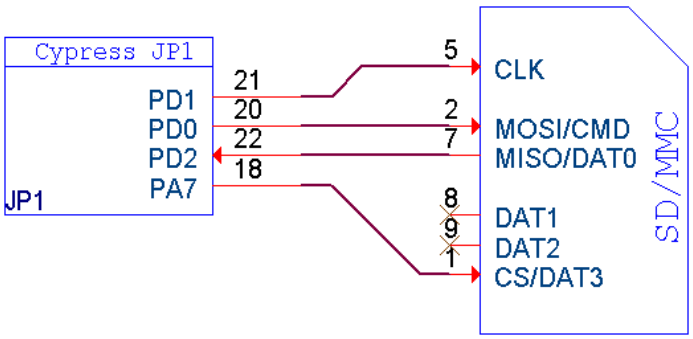


Fig. 4.2 Připojení SD/MMC slotu k mikropočítači Cypress

4.1.2 Připojení obrazového snímače k FPGA

Obrazový snímač ke své funkci požaduje pouze napájecí napětí a přivedení hodinových pulsů, poté na svém výstupu generuje obrazová data, synchronizační signály a výstupní hodinový signál.

Během oživování obrazového snímače byl datový výstup snímače a hodinový signál přiveden přímo k EZ-USB periferii, poté již byla cesta signálu vedena skrze hradlové pole. Propojení je opět realizováno plochým kabelem, vedení signálů zachycuje ilustrace fig. 4.3 .

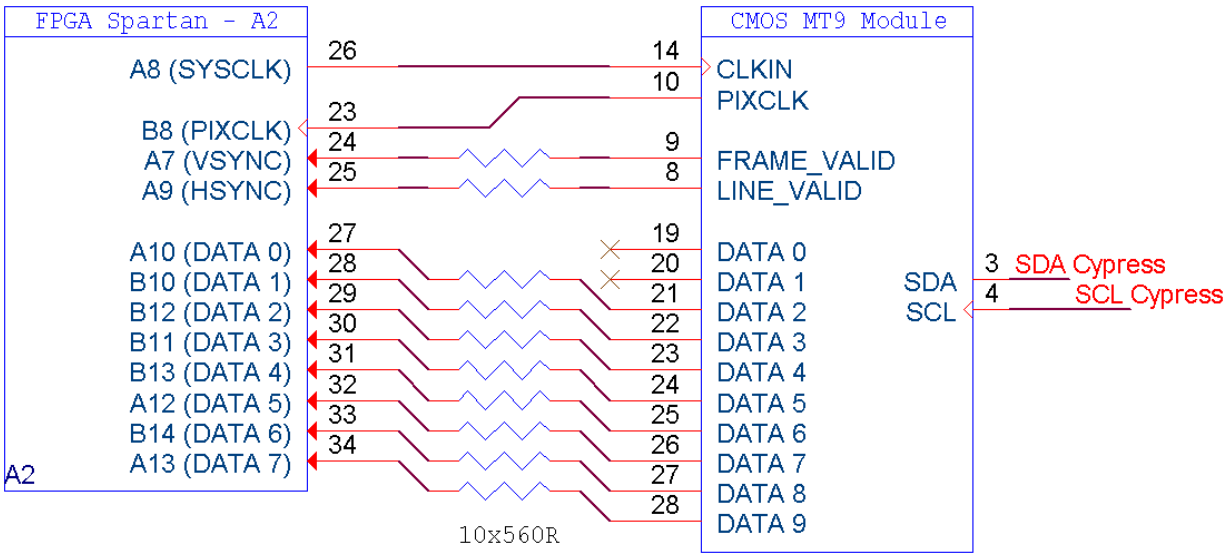


Fig. 4.3 Propojení obrazového snímače ke kitu Spartan3

4.1.3 Výstupní piny FPGA

Výstupní signály hradlového pole jsou vedeny v konektoru B1. Sběrnice I^2C by sice mohla být vedena konektorem A2, v počátcích práce ale nebylo připojení hradlového pole na I^2C sběrnici uvažováno a pozdější připojení bylo snadněji realizováno v konektoru B1.

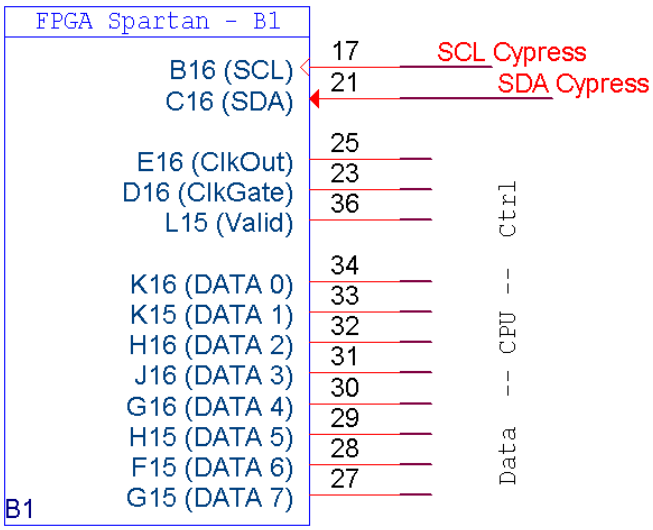


Fig. 4.4 Výstupní piny FPGA

4.1.4 Vliv vedení hodinových pulzů na vady obrazu

Propojení plochým kabelem, které se od počátku nabízelo jako nejjednodušší, mělo základní nedostatek v počtu zemnících vodičů. Kvůli úspoře místa nemá žádný z použitých modulů signály proloženy zemnícími vodiči a napájecí větve jsou vedeny zpravidla na krajních vodičích. Vznikají tak nejen velké smyčky indukující rušení, ale není dosaženo ani korektního vedení signálu pro vysoké kmitočty.

Kombinace odrazů na vedení a přeslechů z okolních vodičů generovala falešné hodinové pulzy, které deformovaly obraz nejen při jeho přenosu do PC aplikace. Tyto vady se projevovaly ve dvou případech, a to vždy při přesvětlení snímané scény.

V prvním případě začal řádek obrazu přesvětleným bodem, tj. na všech datových vodičích se logická úroveň překlopila naráz z log.0 na log.1 a v hodinovém signálu byl zaznamenán impulz navíc. Tento pulz způsobil posunutí celého přenosu o jeden takt napřed a rozpad synchronizace.

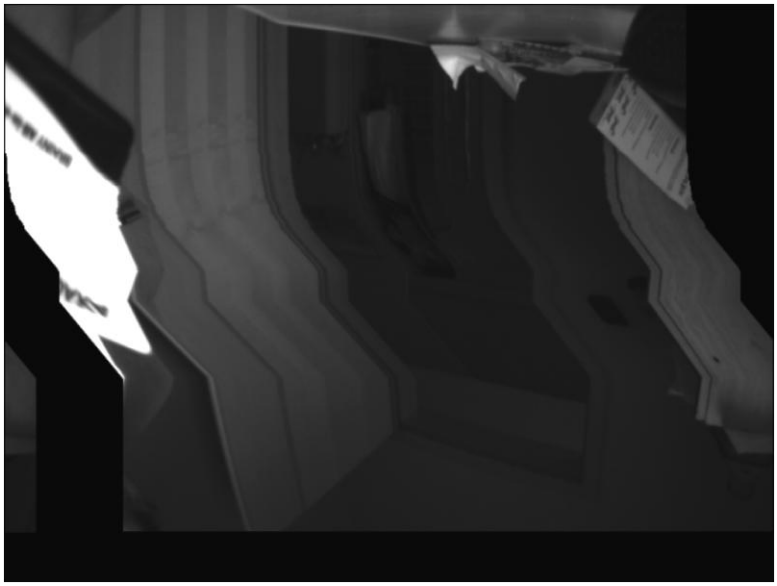


Fig. 4.5 Rozpad obrazu – přesvětlení na začátku řádku

V případě druhém se přesvětlený bod objevil někde dále v obraze a situace byla přesně opačná. Na rozdíl od předchozího případu zde došlo k potlačení hodinového pulzu a vynechání obrazového bodu.

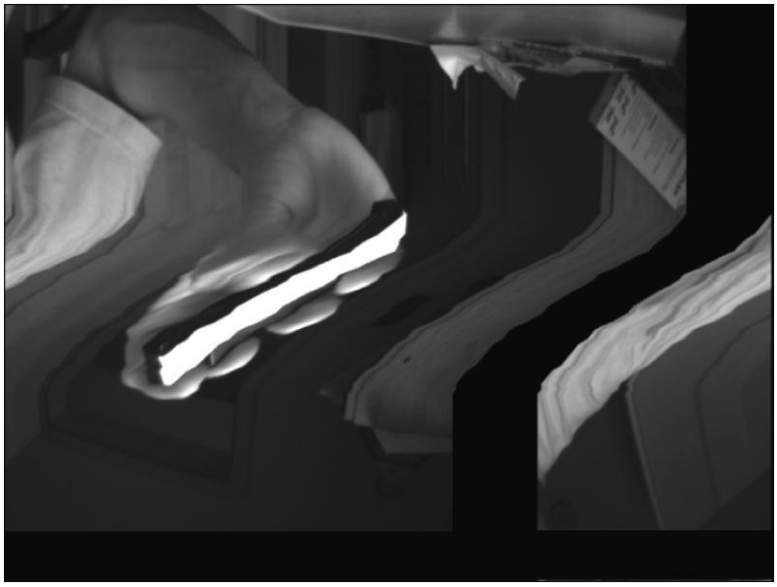


Fig. 4.6 Rozpad obrazu – přesvětlení uvnitř řádku

4.1.5 Stínění hodinových signálů

Po zjištění problémů s hodinovým signálem byl kladen důraz na kvalitu jeho vedení a řádné propojení zemnícími vodiči mezi jednotlivými moduly. Propojení je i nadále realizováno plochými kabely, ale každý z hodinových signálů je stíněn zemnícím vodičem po obou stranách. Propojení všech komponent i s rozlišením přidaných vodičů zachycuje ilustrace fig. 4.7, přidané zemnící vodiče jsou zvýrazněny červeně.

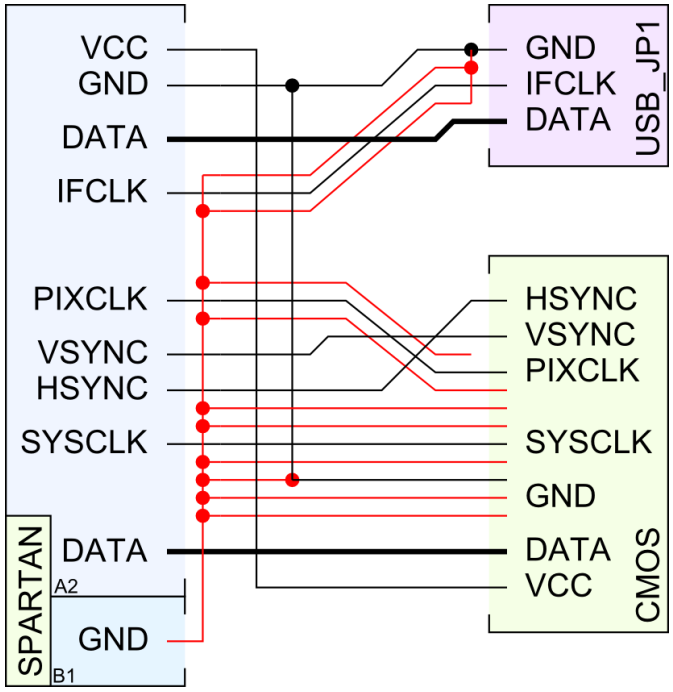


Fig. 4.7 Korektní propojení modulů

4.2 PC aplikace

Aplikace je vytvářena v jazyce C#, knihovny pro přístup k USB zařízení společně s ovladači dodal výrobce EZ-USB periferie. Obslužný program bude zprostředkovávat dvě základní úlohy. První z nich je náhled na obrazová data, druhá pak umožní zásah obsluhy do nastavení obrazového snímače.

4.2.1 Komunikace s EZ-USB, přenos dat

Pro získání snímku zahájí aplikace čtení blokové zprávy z 2. endpointu. Velikost bloku se volí dostatečně velká, aby bylo možné přenést celý snímek, stejně tak má velikost bloku významný dopad i na datovou propustnost endpointu typu „Bulk“. Nyní se data přenáší po blocích o velikosti 1MB, přičemž bylo dosaženo maximální přenosové rychlosti přes 42MB/s. Ve snaze dosáhnout co nejvyšší datové propustnosti bylo experimentováno i s endpointy typu „Interrupt“, ty mají vyšší prioritu a v režimu Hi-Speed umožňují zasílat pakety vícekrát během jednoho rámce. Bohužel USB řadiče ve starších operačních systémech (jako například WinXP) tyto mikro rámce nepodporují, čímž tato konfigurace ztrácí výhodu a dosažená rychlost podstatně klesá. Načtený blok dat je v počítači zpracován, je nalezen počátek snímku a data jsou převedena do formátu zobrazitelného jako obraz.

Mimo přenosu obrazových dat jsou v zařízení i v aplikaci napsány funkce pro zasílání a příjem textových zpráv v obou směrech na endpointu EP1. Textový řetězec odeslaný z PC aplikace je mikropočítačem přijat, zpracován a o výsledku operace lze odeslat zprávu zpět do nadřazeného PC.

4.2.2 Okno aplikace, GUI

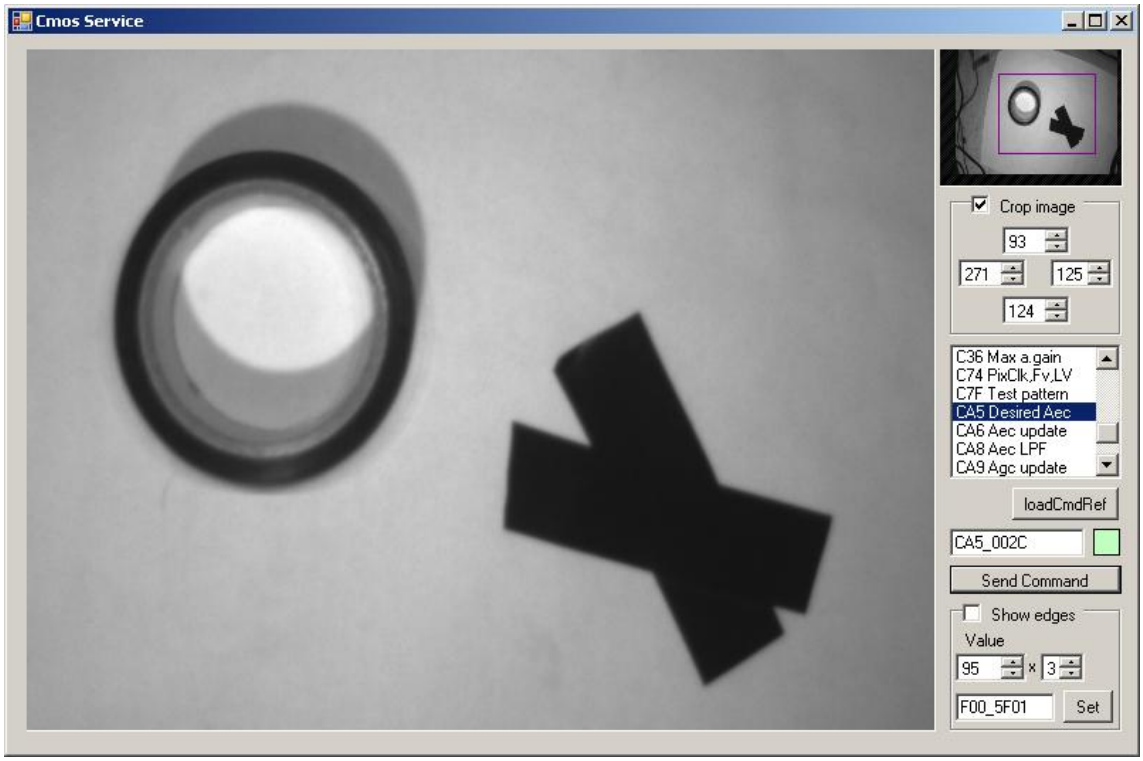


Fig. 4.8 Obslužná aplikace

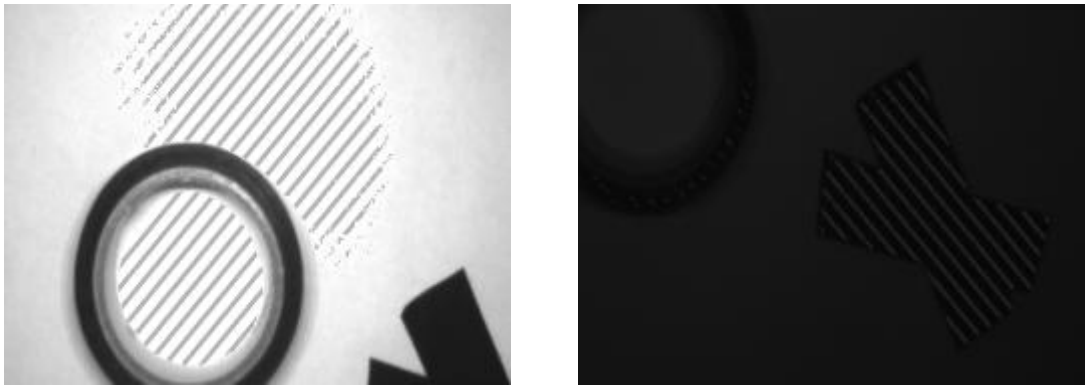
Cílem bylo dosáhnout co možná nejjednodušší aplikace, která bude přehledná a intuitivně obsluhovatelná. Grafické rozhraní je zobrazeno na snímku fig. 4.8.

4.2.2.i Přenos a vykreslení obrazu snímané scény

Primárním účelem aplikace je zobrazení snímaného obrazu. Obrazová data jsou skrze USB periferii přenášena včetně zatemněných obrazových bodů, doplněna do formátu obrazu použité grafické třídy a vykreslena do miniatury v pravém horním rohu. Do hlavní zobrazovací plochy (vlevo) je vykreslen výřez obrazu zvolený ovládacími prvky umístěnými vpravo pod miniaturou obrazu. Při zvolené možnosti „Crop image“ je obraz před vykreslením z každé strany oříznut o zadaný počet obrazových bodů a pro lepší orientaci je hranice zvoleného výřezu vykreslena do obrazu miniatury. Není-li funkce aktivní, je do hlavní zobrazovací plochy vykreslen neoříznutý obraz.

4.2.2.i -a Zobrazení jasových extrémů

Na základě předchozích zkušeností s kamerovou a studiovou technikou je rozhodnuto vložit do programu funkci pro zobrazení příliš světlých a příliš tmavých ploch obrazu. Princip spočívá v překrytí těchto ploch šrafovanou, pohybující se šedou maskou, díky které lze špatně osvětlené plochy velmi snadno rozlišit. Zobrazení je zachyceno v fig. 4.9.



a) Přesvětlená scéna b) Nedosvětlená scéna

Fig. 4.9 Zobrazení jasových extrémů

4.2.2.ii Textové zprávy pro konfiguraci zařízení

Nastavení obrazového snímače a hradlového pole se provádí prostřednictvím textových zpráv. Těmto zprávám byl definován triviální formát, na jehož základu je mikropočítačem sestavena I^2C transakce a příkaz zpracován. Zpráva se sestává z identifikátoru zprávy, adresy zapisovaného registru a datového bloku. Oddělovacím znakem mezi adresou a datovým blokem je znak podtržítka ‘_’.

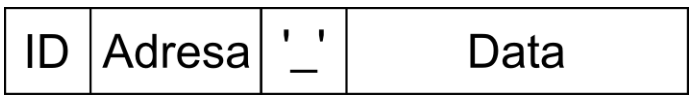


Fig. 4.10 Formát textových zpráv

V obou případech zasílaných zpráv je identifikátorem jeden ASCII znak, adresa i data jsou ve formě znaků reprezentujících šestnáctkovou soustavu {0-9 , a-f, A-F}.

4.2.2.ii -a Příkazy CMOS snímače

Identifikátorem zápisu do registru CMOS snímače je znak '**C**', po jeho přijetí je adresa I^2C zřízení nastavena na obrazový snímač z tabulky uložené ve firmwaru. Adresní byte není nijak převáděn a přímo adresuje registr uvnitř obrazového snímače. Datový blok jsou dva byty, jejichž interpretace záleží na adresovaném registru. Pro přehled všech adres, registrů a jejich přípustných hodnot viz katalogový list výrobce [7]. Příkladu transakce uvedenému výše (viz fig. 3.2) by odpovídal řetězec „C09_0284".

4.2.2.ii -b Definice příkazu pro nastavení FPGA

V logice hradlového pole je pro běh algoritmu podstatný jediný parametr, kterým je nastavení prahové hodnoty pro rozlišení hran v obraze. Hodnota parametru je osmibitové číslo, které je násobeno až čtyřikrát. Tím je pokryt i nejvyšší rozdíl jasu, který může být v řádu deseti bitů. Pro korektní nastavení je nutno si hrany zobrazit, za tímto účelem je v datovém bloku obsažen i bit, kterým se volí mezi přenosem snímaného obrazu a přenosem výstupu hodnocení hran.

Konfigurační slovo je rozděleno na 2 byty, první byte nese 8 bitovou prahovou hodnotu, druhý byte slouží pro doplňující nastavení.

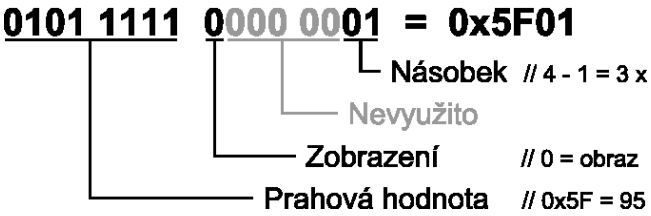


Fig. 4.11 Příklad konfiguračního slova FPGA

Na příkladu fig. 4.11 je v prvním bytu obsažena prahová hodnota 95, v druhém bytu je 7. bitem zvolen přenos obrazu a nejnižšími dvěma bity nastaven násobek prahové hodnoty. Násobek je v rozsahu 1-4 reprezentován číslem 3-0. Výsledná prahová hodnota se spočítá dle vzorce

$$Threshold = Value \cdot (4 - cfg[1:0]) = 95 \cdot (4 - 1) = 285 \quad ,$$

kde *Value* je obsah prvního bytu a *cfg[1:0]* je hodnota posledních dvou bitů druhého bytu.

Zpráva určena pro nastavení FPGA začíná identifikátorem '**F**'. S příjmem I^2C transakce není uvnitř hradlového pole spojena žádná paměť, adresní byte je ve firmwaru mikropočítače ignorován a je přeskočen. Transakce probíhá stejně jako v případě CMOS snímače, pouze s rozdílem že adresa registru je vynechána a ihned po adrese I^2C zařízení následují data. Adresa FPGA na I^2C sběrnici je pevně nastavena na 0x4D a také uložena ve firmwaru mikropočítače. Příkladem zprávy může být řetězec „F00_5F01".

4.2.2.ii -c Ovládací prvky textových zpráv

Pro psaní a zasílání zpráv jsou v pravém spodním rohu okna umístěny ovládací prvky, které rozdělíme na tři pomyslné útvary.

a) b) c)

Fig. 4.12 Ovládací prvky textových zpráv

Prvním z nich (fig. 4.12–a) je seznam vybraných registrů obrazového snímače. Tento seznam je nahrán do aplikace z externího souboru a slouží jako nápověda a ulehčení pro uživatele. Existuje-li ve spouštěcím adresáři soubor „cmdRef.cmdr“, aplikace nahraje jeho obsah do seznamu. Uživatel má samozřejmě možnost nahrát položky nápovědy z libovolného souboru, stisknutím tlačítka „loadCmdRef“ se otevře dialogové okno pro výběr souboru. Položky seznamu je možno procházet, nebo výběrem některé z položek vyplnit textové pole příkazu identifikátorem, adresou vybraného registru a dělícím znakem. Formát položek seznamu je navržen tak, aby k tomuto účelu stačilo kopírovat první tři znaky vybrané položky a doplnit je podtržítkem.

Vprostřed (fig. 4.12–b) je umístěno textové pole příkazu a signalizace úspěšné transakce. Do textového pole lze zadat libovolný řetězec a stiskem tlačítka „Send Command“ ho odeslat ke zpracování mikropočítačem v USB periferii. Úspěšné provedení I^2C transakce je textovou zprávou potvrzeno zpět do aplikace. Čtverec vpravo od textového pole se při úspěšné transakci zbarví do odstínu zelené, při neúspěšné transakci se vybarví odstínem červené.

Poslední ovládací prvky (fig. 4.12–c) slouží jako pomůcka pro nastavení hradlového pole. Textové pole a tlačítko „Set“ jsou z pohledu funkce duplikátem předchozích ovládacích prvků a jsou odděleny spíše pro pohodlí uživatele. Zbylé tři ovládací prvky usnadňují konstrukci textové zprávy. Zaškrtávacím polem „Show edges“ se volí mezi přenosem hran a přenosem obrazu. Obsah dvou numerických prvků „Value“ představuje prahovou hodnotu rozpoznávání hran. Při změně libovolného z těchto tří prvků je z jejich hodnot zkonstruováno konfigurační slovo a odpovídající řetězec vložen do textového pole. Odeslání se provede stiskem tlačítka „Set“.

4.3 Konfigurace hradlového pole

4.3.1 Oživení FPGA

Oživení hradlového pole se sestává z posloupnosti několika málo kroků. Postupně bylo realizováno vlastní propojení modulů, oživení/ověření napájecích napětí a v posledním kroku vytvoření konfiguračního obrazu. Tento prvotní obraz zprostředkovává pouze zdroj 25MHz hodinového signálu na pinu SYSCLK pro funkci

obrazového snímače a propojení výstupních signálů snímače na vstupní bránu EZ-USB periferie (DATA→DATA, PIXCLK→IFCLK).

Během oživování přišla na řadu i otázka nezapojených vstupů a také zvážení konfigurace PULL-UP či PULL-DOWN rezistorů u vstupních pinů. První otázku již vyřešil výrobce hradlového pole, nevyužitým pinům je při generování bitstreamu nastaven vnitřní pull-up rezistor a není nutné se jimi zabývat. V otázce vstupních pinů narážíme na poměrně značné proudy při připojení pull-up či pull-down rezistorů (až 1,8mA/pin) a dopad této zátěže na okolní moduly. V hraničních situacích došlo až k přetížení výstupu obrazového snímače a jeho výpadkům, z těchto důvodů nebyla žádná varianta vnitřních rezistorů použita.

4.3.2 Zpracování obrazových dat

4.3.2.i Zavedení konvencí pro návrh

Na samém začátku je vhodné zanést do návrhu jisté konvence, které budeme v průběhu realizace dodržovat. Tento krok by měl pomoci k úspěšné a především přehledné realizaci.

- Návrh bude rozdělen do navzájem navazujících funkčních bloků
- Hodinový signál bude skrze logické bloky veden spolu s obrazovými daty.
- Vstupní signály jsou čteny během náběžné hrany vstupního hodinového signálu
- Výstupní signály se mění při sestupné hraně výstupního hodinového signálu.
- Veškerá data a informace se předávají jen mezi přímo navazujícími bloky.

Poslední pravidlo je důležité pro zachování synchronnosti s hodinovým signálem, obecně uvažujeme i bloky, kde se fáze výstupních signálů liší od fáze vstupních.

4.3.2.ii Krok 0 – Vstup dat, souřadnice

První článek řetězu má za účel zpracovat synchronizační signály z obrazového snímače a odvodit souřadnice přijímaného obrazového bodu. Souřadnice sloupce se navýší s každým přijatým obrazovým bodem, souřadnice řádku pak s každým řádkovým synchronizačním pulsem. Ideovou implementaci s užitím čítačů naznačuje fig. 4.13.

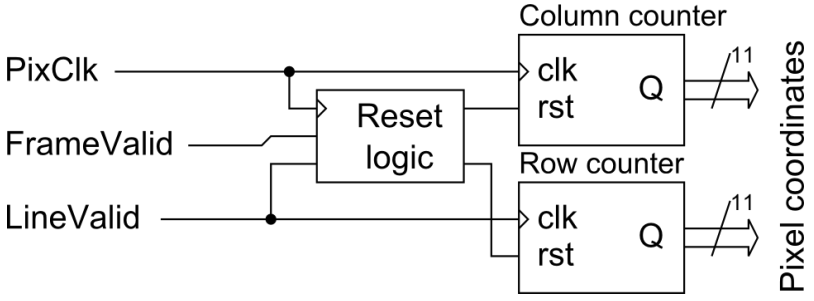


Fig. 4.13 Ideový návrh výpočtu souřadnic

Resetovací logika musí v takovémto uspořádání zajistit včasné nulování příslušných souřadnic. Nejjednodušším řešením by bylo řídit reset čítačů hranami synchronizačních pulsů, syntéza návrhu ale nedovolí přístup k hodnotě čítače dvěma obecně nezávislým událostem. Lepší řešení je vzorkovat synchronizační signály společně s daty a s použitím

další logiky hledat v průbězích patřičné vzory. Signály složené dle těchto vzorů budou taktéž synchronní s hodinovým signálem.

Pozice v rámci řádku (souřadnice sloupce) je odvozena od počtu hodinových pulsů a náběžné hrany signálu LINEVALID, která určuje počátek řádku a nuluje čítač sloupců. Souřadnice řádku je naopak odvozena od počtu pulsů signálu LINEVALID a náběžné hrany signálu FRAMEVALID, která značí počátek snímku a nuluje čítač řádků.

Průběh takto odvozených signálů ROWRST, LINERST a výstupních hodnot čítačů je zachycen v fig. 4.14.

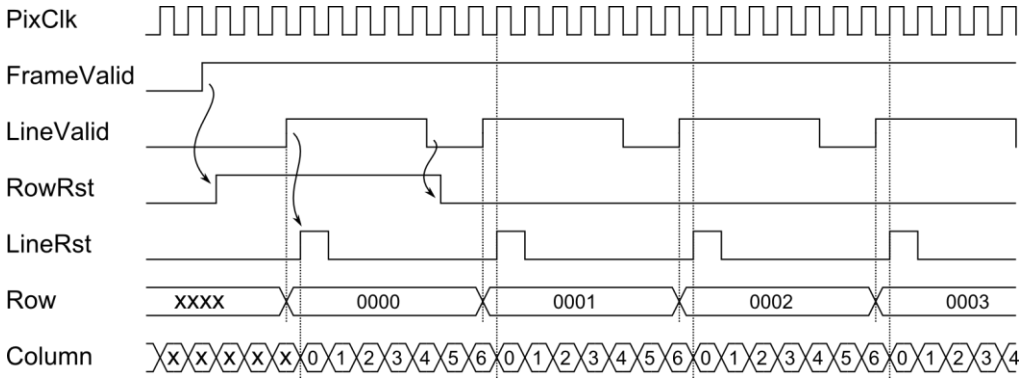


Fig. 4.14 Signály čítačů souřadnic

Výstupní signály ale nelze v tomto tvaru použít. Naše vlastní pravidlo vyžaduje synchronnost všech výstupních signálů, což zde není dodrženo. Jak vidíme, souřadnice sloupce je opožděna o půl taktu hodinových pulzů oproti souřadnici řádku, zároveň však i oproti vstupním obrazovým datům. Jediné přípustné řešení je tyto signály ještě před výstupem uměle zpozdit.

4.3.2.iii Krok 1 – Vytvoření podkladů pro labeling

Algoritmy labelingu vyžadují vstupní data v podobě binárního obrazu. Každý pixel takového obrazu poskytuje pouze informaci, zda je či není součástí objektu. Naším úkolem je v tomto kroku převést snímanou scénu na její binární obraz.

Nalezení hran není triviální záležitost a samozřejmě je na místě otázka, zda je nutné použít právě toto řešení. Nechť je obrazem snímané scény fig. 4.15.

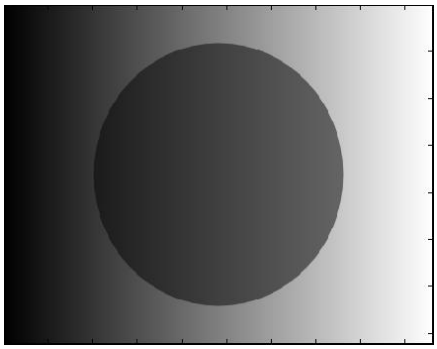


Fig. 4.15 Příklad nevhodně osvětlené scény

Objekt, který je reprezentován šedým kruhem, je zastíněn transparentní maskou napodobující nerovnoměrné osvětlení scény. Přestože je objekt dobře rozeznatelný,

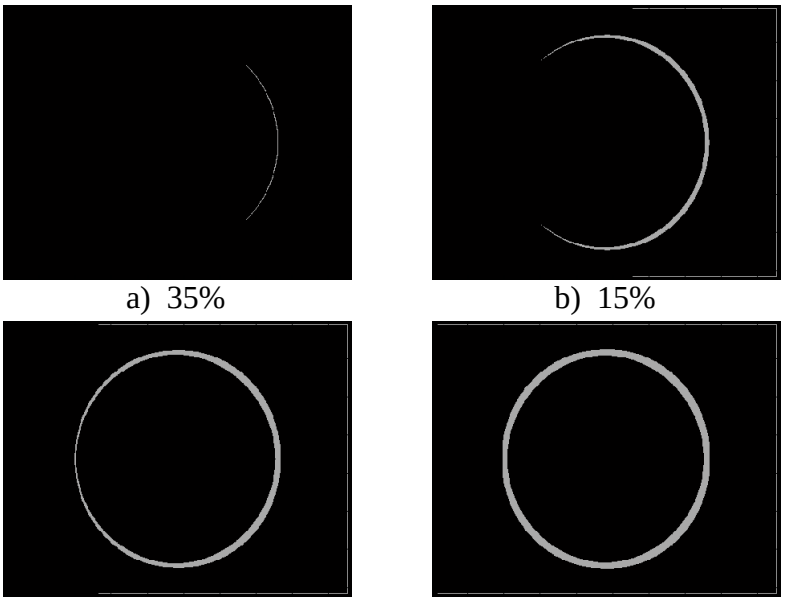
pokusíme-li se o triviální řešení pomocí prahování úrovně jasu, k žádanému výsledku nedojdeme. Výsledky pro tři vybrané komparační úrovně jsou zachyceny v fig. 4.16.

a) 20% b) 30% c) 40%

Fig. 4.16 Chyby komparační metody

Nyní ukažme, že naše řešení dokáže objekt rozlišit. Nad daným motivem byl spuštěn algoritmus hledání hran, který věrně napodobuje budoucí implementaci. Vlastní algoritmus bude popsán níže, zde pouze diskutujeme jeho výsledky.

Hranou zde rozumíme přechod mezi dvěma jasovými úrovněmi, který je strmější než vložená mez. Výsledky pro vybrané strmosti znázorňuje fig. 4.17.



a) 35% b) 15%

c) 7,5% d) 1%

Fig. 4.17 Výsledek algoritmu hledání hran

4.3.2.iii -a Metodika hledání hran

V problematice hledání hran bude navázáno na předchozí výsledky. Metodika hledání hran bude zachována, nicméně realizace dojde k jistým úpravám.

Z předchozí práce [2] je využita konstrukce matice 8x8 obrazových bodů a konvolučního jádra *[1 1 1 1 -1 -1 -1 -1]*, kterým budou hrany vyhodnoceny.

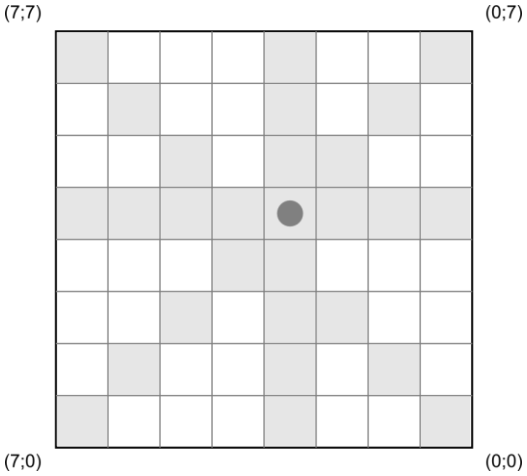


Fig. 4.18 Uspořádání obrazové matice

Souřadný systém uvnitř matice je zvolen tak, že aktuálnímu obrazovému bodu přísluší souřadnice (0;0). Tím se lze jednoduše pohybovat v souřadnicích jako v historii obrazu. Ve zvoleném smyslu čtení scény pak geometrie nastíněná v fig. 4.18 přímo reprezentuje výsek obrazových dat. Vztažným bodem hledání hran je střed matice nebo obrazový bod nejblíže středu (zvýrazněný bod na souřadnici (3;4)). Ve všech význačných směrech je vybrána osmice obrazových bodů tvořících úsečku procházející vztažným bodem, data ostatních obrazových bodů nejsou při výpočtu hran zahrnuta.

Díky jednoduchosti konvolučního jádra je výpočet uvnitř matice realizován prostým součtem. Uvedeme výpočet pro rozlišení horizontální hrany dle [2].

$$diff = |Matrix(4,4) - Matrix(3,4)| + |Matrix(5,4) - Matrix(2,4)| + |Matrix(6,4) - Matrix(1,4)| + |Matrix(7,4) - Matrix(0,4)|$$

Takto vypočtená hodnota prakticky představuje strmost hrany v blízkém okolí vztažného bodu. Bude-li tato hodnota vyšší než zadaná mez, prohlásíme o vztažném bodu, že se v něm nachází hrana objektu. Pro ostatní směry je situace obdobná.

4.3.2.iii -b Konfigurace RAM

Příchozí obrazová data jsou uložena v paměti, pro každý řádek je obsazena jedna bloková RAM. Na čipu XC3S200 je v každém bloku možné uložit 18KBit dat, tedy teoreticky nejdelší řádek o délce 2304 obrazových bodů. Využitím existujících komponent vývojového prostředí se spokojíme s délkou řádku 2048 bodů.

Před uložením dat do paměti přečteme nejprve její obsah, ten je z principu zpožděn přesně o jeden řádek. Takto lze zřetězit libovolný počet pamětí, v našem případě sedm.

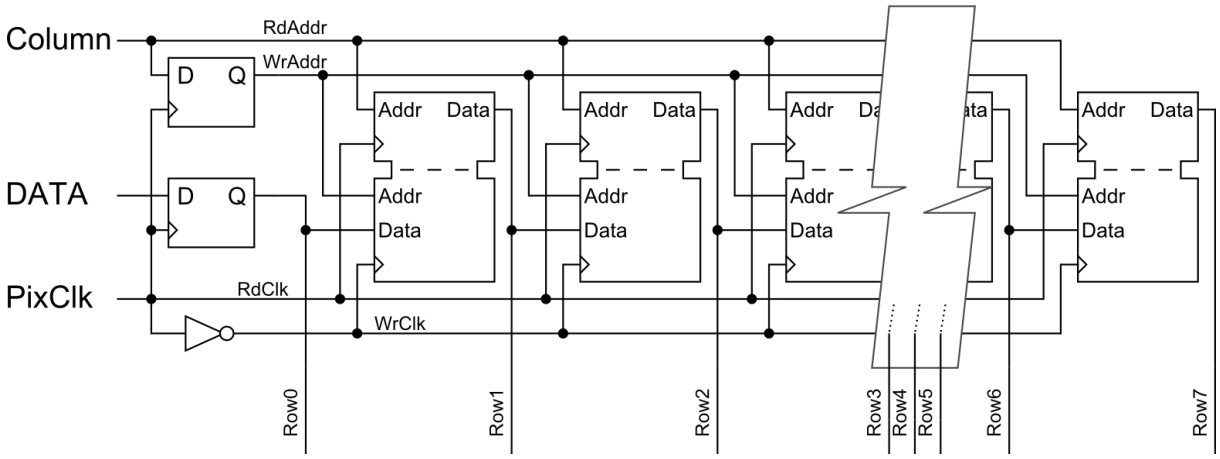


Fig. 4.19 Zřetězení RAM pamětí

Nyní se podívejme blíže na fig. 4.19. Na rozdíl od předchozích prací je zde kladen velký důraz na řešení hazardních stavů během zápisu do paměti. Přímé zřetězení pamětí – propojení výstupní brány přímo na vstup následujícího bloku, vnáší pochybnost o korektnosti použitého řešení. Nastává situace, kdy během jediné náběžné hrany čteme obsah paměti a zároveň usilujeme o jeho zápis v bloku následujícím, tedy vstupní data se v okamžiku zápisu mění.

Implementované řešení využívá paměti s duálním přístupem. První brány pamětí jsou konfigurovány pouze pro čtení a jsou aktivní při náběžné hraně hodinového signálu, spolu s klopným obvodem na vstupu zajišťují synchronnost dat všech osmi řádků potřebných pro obrazovou matici. Tato data jsou platná během sestupné hrany hodinového signálu, právě v tento okamžik se aktivují druhé brány RAM pamětí, kterými se tato data uloží vždy do následujícího bloku.

4.3.2.iii -c Konstrukce obrazové matice

Hodnoty všech buněk požadujeme mít kdykoliv k dispozici a přistupovat k nim paralelně, stejně tak chceme celou matici během jediného taktu hodin posunout o jeden sloupec vpřed. Ideálním řešením je realizovat matici klopnými obvody na úrovni sloupců. Nultý sloupec představuje vstupní data do matice, jeho hodnota je složena z aktuálních dat a uložených řádků, každý další sloupec je jen jeho zpožděným obrazem.

Časování plynule navazuje na takt RAM pamětí. Během vzestupné hrany hodin se do klopných obvodů uloží kopie dat, ta je poté posunuta o jeden sloupec a spolu s novými daty načtena zpět během sestupné hrany hodin.

```
--matrix datatype
type MatrixColumn  is array (0 to 7) of integer range 0 to 255;
type Matrix   is array (0 to 7) of MatrixColumn;
signal EdgeMatrix, ShadowMatrix : Matrix;

SHIFTMATRIX : process (ClkIn)
  begin
    if (ClkIn'event and ClkIn = '1') then
    --rising edge -> ulozeni dat do stinove kopie
      ShadowMatrix <= EdgeMatrix;
    end if;

    if (ClkIn'event and ClkIn = '0') then
    --falling edge -> platna data na vystupu RAM
      EdgeMatrix(7) <= ShadowMatrix(6);
      EdgeMatrix(6) <= ShadowMatrix(5);
      EdgeMatrix(5) <= ShadowMatrix(4);
      EdgeMatrix(4) <= ShadowMatrix(3);
      EdgeMatrix(3) <= ShadowMatrix(2);
      EdgeMatrix(2) <= ShadowMatrix(1);
      EdgeMatrix(1) <= ShadowMatrix(0);
    --vlozeni nejnovejsiho sloupce
      EdgeMatrix(0)(0) <= conv_integer(Row0);
      EdgeMatrix(0)(1) <= conv_integer(Row1);
      EdgeMatrix(0)(2) <= conv_integer(Row2);
      EdgeMatrix(0)(3) <= conv_integer(Row3);
      EdgeMatrix(0)(4) <= conv_integer(Row4);
      EdgeMatrix(0)(5) <= conv_integer(Row5);
      EdgeMatrix(0)(6) <= conv_integer(Row6);
      EdgeMatrix(0)(7) <= conv_integer(Row7);
    end if;
  end process;
```

Fig. 4.20 Plnění obrazové matice na úrovni VHDL

Vyhodnocení hran je realizováno paralelním výpočtem z dat obrazové matice, považujme ho pro naše účely za synchronní spolu s obrazovou maticí. Prahová hodnota, se kterou je výsledek výpočtu porovnáván, je získána příjmem I^2C transakce (viz kapitolu 4.3.3).

4.3.2.iv Krok 2 – Určování objektů

Před vlastní implementací je nezbytné si tento krok důkladně promyslet. Během hodnocení vlastností a použitelnosti zkoumaných algoritmů se do popředí dostávají dvě užívané možnosti realizace.

Metoda „hrubé síly“ – Každému labelu[8] je alokován příslušný prostor v paměti, rozhodovací strom přidělování labelů je optimalizován na rychlost. Slučování labelů probíhá až po uzavření snímku. Výhodou takového algoritmu je rychlost čtení obrazových dat, během čtení obrazu je veškerý výpočetní výkon směřován pouze na přidělování labelů a sčítání těžišť. Přední nevýhodou je vysoký nárok na paměť a její neefektivní využití.

[8] „Label“ – Anglický výraz pro „štítek, označení, jmenovka a pod.“.
V chápání algoritmu představuje segment plochy – jeho ohodnocení ukazuje na segment, kterého je součástí. Jeden objekt je složen z jednoho či více sousedících segmentů. V českých popisech se lze setkat i s pojmem „značka“.

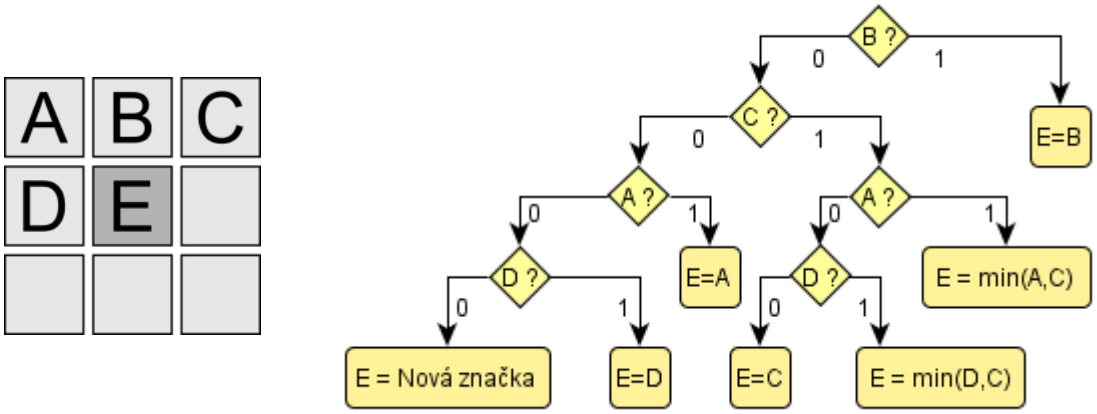
Metoda „výpočetní“ – Na rozdíl od předchozího příkladu se objekty slučují během zpracování, algoritmus je propracovanější a zpravidla využívá výhod objektového přístupu. Výhodou je nižší nárok na paměť a okamžité zpracování. Nevýhodou je jeho složitá implementace a nárok na výpočetní výkon.

Bohužel ani jedna z možností není vhodná pro implementaci na hradlových polích. Jak prokázala simulace vybraných algoritmů, i pro relativně jednoduchý tvar je zapotřebí desítek labelů, z nichž po sloučení bude pouze jeden reprezentovat výsledek. Žádný ze zkoumaných algoritmů ani neuvažoval uvolňování paměti a její další využití, ve většině případů by to ani z principu nebylo možné díky tomu, že hodnota labelu reprezentovala i jeho stáří a byla nutnou součástí při řešení kolizí.

S dostupnou pamětí je na poměrenou již práce v prostoru 255 labelů, které stačí na ohodnocení pouze jednoduchých snímků. Pro komfort při zpracování by bylo nutno mít prostor přinejmenším dvakrát větší.

4.3.2.iv -a Navržená metoda labelingu

Nejprve nastíníme obecný základ, který je společný i napříč různými algoritmy.



a) Okolí obrazového bodu b) Ukázka rozhodovacího stromu [18]

Fig. 4.21 Jedna ze zkoumaných implementací labelingu

Uvažujme algoritmus labelingu, který využívá osmi-okolí k ohodnocení daného bodu. V případě jednoprůchodového zpracování jsou k dispozici pouze čtyři již zpracované body A,B,C a D. Ze znalosti těchto bodů a přidružené logiky je odvozen label připadající aktuálně zpracovávanému bodu E a jsou řešeny případné kolize.

Řešení kolizí je v základu redukováno na tzv. tabulku ekvivalentních značek. Ta obsahuje informace o sousednosti nalezených ploch a je jediným vodítkem pro jejich slučování.

4.3.2.iv -a.i Použitelnost stávajících algoritmů

Značně omezený prostor v paměti nebyl nakonec tím nejsložitějším problémem. Za mnohem významnější je považováno řešení kolizí, respektive konstrukce tabulky ekvivalentních značek.

Metodou hrubé síly by byl zapotřebí enormní paměťový prostor, který by dokázal obsáhnout všechny možné kombinace. Takový úsek by v paměti zabral více místa než

vlastní data. Například již pro 256 labelů by se alokovalo 64 kB paměti, za cenu pomalého zpracování by šlo zredukovat tento nárok na „pouhých“ 8 kB. Víme ale, že 256 labelů stačí pouze pro jednoduché snímky a dostupný blok paměti je více než 3x menší.

Zpracování metodou výpočetní se tedy nezdá jako lepší řešení, ale jako jediné řešení. Bohužel nemáme k dispozici objektový přístup nebo práci s ukazateli. Hledání obdobných řešení často vedlo k nepřiměřeně složitým strukturám, ani v idealizovaných konceptech nebyly algoritmy spolehlivé nebo si zpracování žádalo příliš mnoho času.

Dříve nebo později bylo nutno najít radikální změnu stávajícího algoritmu. Do úvahy přicházela implementace procesoru MicroBlaze[9], s kterým by bylo možno navázat na předchozí výsledky a využít dosažených optimalizací. Problematika labelingu by se prakticky přeformovala na problematiku využití emulovaných procesorů. Cítíme však, že by to byl útěk od problému a necháváme tuto možnost až jako poslední možnou.

Čas věnovaný problematice tabulky ekvivalentních značek přinesl poznatky, ze kterých se pokusíme čerpat a upravit dle nich i zbytek algoritmu. Cílem této snahy je napsání algoritmu, který degraduje tuto tabulku na co nejmenší konečný počet sloupců bez újmy na obecnosti.

Ve snaze redukovat počet navazujících labelů bylo experimentováno s rozhodovacím stromem a způsobem, jakým je do tabulky ekvivalentních značek přistupováno. Vznikla celá řada algoritmů, většina z nich však byla již na papíře prokázána za nevyhovující.

Zkusme si náš problém formulovat. Již v definici algoritmu se nahlíží na labely jako na vrcholy grafu, tabulka ekvivalentních značek pak reprezentuje seznam všech existujících hran mezi vrcholy. Snažíme-li se redukovat nároky na velikost tabulky, snažíme se vlastně o algoritmus, jehož idealizovaným výsledkem je orientovaná hamiltonovská cesta pro každou komponentu souvislosti. V takovém případě by totiž tabulka obsahovala pro každý vrchol nejvýše jednu hranu, tedy zkolabovala by do jediného sloupce. Zároveň by bylo možné jediným průchodem sloučit labely do celistvých objektů.

Není těžké si domyslet, že takto ideální případy nastanou pouze výjimečně. Topologie vzniklého grafu přímo odráží tvar snímaného objektu, budeme se tedy mnohem častěji potýkat s grafem připomínající strom. Rozšiřme požadovaný výsledek na kořenový strom. Pro ten i nadále platí, že z každého vrcholu vede nejvýše jedna orientovaná hrana.

Napříč různými metodami se ukazuje, že libovolnému algoritmu lze najít opakující se motiv v obrazu, pro který se stává metoda nepoužitelná. Pro demonstraci tohoto tvrzení uvedeme dva příklady a dokážeme, že tabulku ekvivalentních značek není možné redukovat pouze úpravou rozhodovacího stromu.

[9] MicroBlaze™ - Produkt společnosti Xilinx, emulace mikropočítače uvnitř hradlového pole.

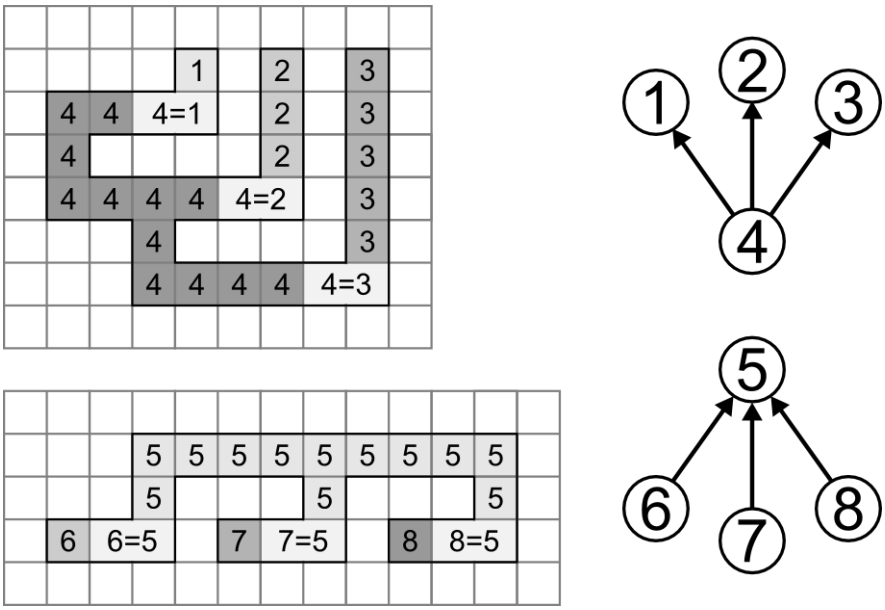


Fig. 4.22 Protichůdné výsledky vzorové implementace

Fig. 4.22 ilustruje dva podobné případy. Ohodnocení obrazců je v obou případech naprosto nezávislé na tvaru rozhodovacího stromu. Protichůdnost obou výsledků pak dokazuje, že jakkoliv zvolená orientace hran nezaručuje správný výsledek.

4.3.2.iv -a.ii Úprava algoritmu

Soubor znalostí by již měl být dostačující k navržení vhodných úprav. Nejprve budou nastíněny změny v oblastech, které již byly diskutovány.

První v pořadí je tedy rozhodovací strom pro ohodnocování obrazových bodů. V jádru se nebude příliš lišit od stávajících řešení, ale bude mít značný vliv na algoritmus jako celek. Tvar stromu je navržen pro schopnost recyklace starých labelů s respektem na řádkové zpracování obrazu. Tím je myšleno, že se snažíme o co nejmenší počet použitých labelů v rámci jednoho řádku. Bude-li řádek obsahovat souvislý objekt, bude ohodnocen jedním labelem po celé jeho šířce, a to bez ohledu na předešlé slučování. Toho je docíleno striktní předností zleva. Stejnou, ale možná srozumitelnější interpretací je skutečnost, že zleva či zprava sousedící obrazové body budou mít vždy stejný label. Tato skutečnost by v budoucnu mohla u rozměrných objektů ušetřit výpočetní čas.

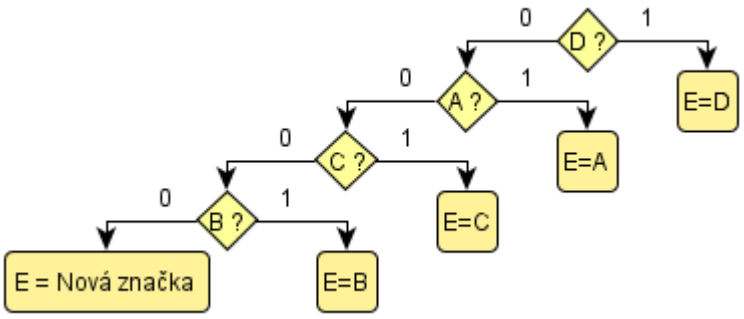


Fig. 4.23 Upravený rozhodovací strom

Jak ilustruje fig. 4.23, ohodnocení obrazového bodu záleží ryze na tvaru objektu, hodnota labelu již není přímou součástí algoritmu. Přednost mají labely na pozicích D a A (již diskutovaná přednost zleva), pak následují pozice C a B. Na první pohled jsme přednost zleva porušili, ale skutečnost že bod B je sousedící s body A a C znamená, že nachází-li se objekt v některém z nich, je z principu ohodnocen totožně.

Přímo v návaznosti na rozhodovací strom je nutno vyřešit další dvě otázky. Jedna z nich je přidělování volných labelů, ke které se dostaneme později. Druhá z nich je čtení labelu z předchozího řádku (obrazového bodu na pozici C). Než nastíníme výsledek, bude vhodné uvést princip řešení kolizí. Kolize může nastat pouze mezi dvojicemi labelů v pozicích A~C a D~C. Dle rozhodovacího stromu z fig. 4.23 se label na pozici C v obou případech neuplatní, z pohledu slučování labelů je tedy správným kandidátem k zániku. Nelze jej však jednoduše sloučit a zapomenout, v předešlém řádku může existovat ještě celá řada jeho výskytů, se kterými je nutno pracovat. Není ale možné přepisovat všechny položky při každém sloučení, taková operace je v našich podmínkách nemyslitelná. Alternativní způsob je vytvořit soubor odkazů (tabulky ekvivalentních značek) a při každém čtení se přesvědčit, zda je label stále aktivní nebo zda již byl sloučen a nahrazen. Vhodnou implementací jsme schopni dosáhnout námi požadovaného kořenového stromu a redukovat paměťové nároky na tabulku ekvivalentních značek. Příklad uvádí fig. 4.24.

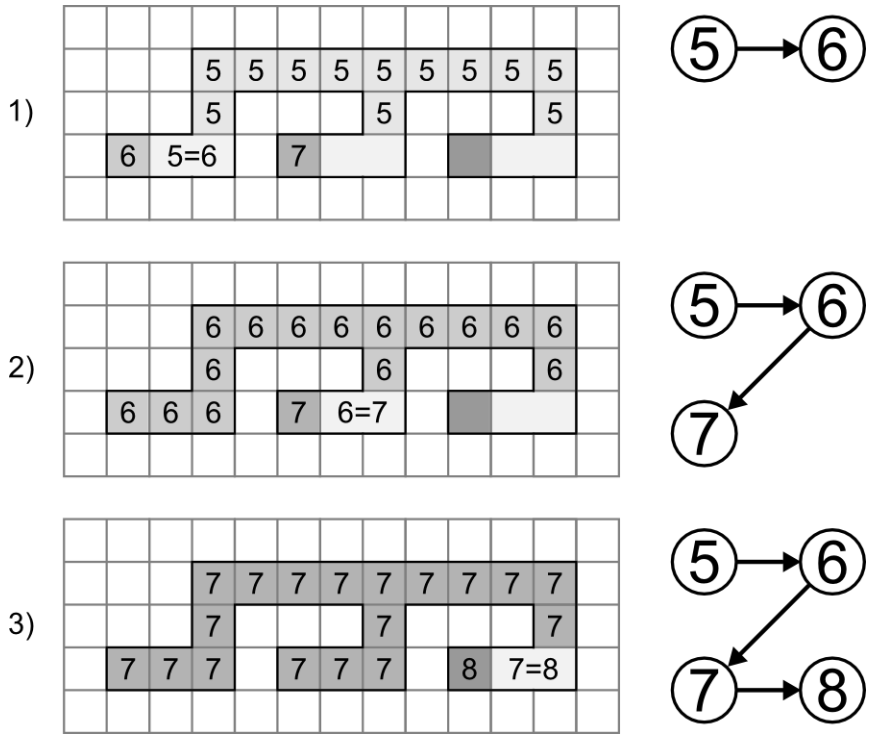


Fig. 4.24 Řešení kolizí - orientovaná cesta

Uvedeme příklad totožný s fig. 4.22, ale uplatníme nové principy. *V prvním kroku nastává kolizní stav mezi labely 5 a 6. Uplatněním nových pravidel je objekt č.5 sloučen do objektu č.6 a label 5 zaniká. Do tabulky odkazů je na pozici 5 uloženo číslo 6. V druhém kroku jsme v místě, kde je z předešlého řádku přečten label 5. Tuto hodnotu použijeme jako adresu v tabulce odkazů, ze které takto přečteme číslo 6. Jakýkoliv bod, který patřil objektu č.5 se nám nyní jeví jako objekt č.6. Teprve nyní přecházíme*

k rozhodovacímu stromu. V tomto případě nastala kolize mezi labely 6 a 7 a předchozí situace se opakuje. Objekt č.6 je sloučen do objektu č.7 a label 6 zaniká. Na jeho pozici v tabulce odkazů je uloženo č.7. V třetím kroku se jedná o identickou situaci. Z předešlého řádku je přečten label 5. Na pozici 5 je v tabulce odkazů přečtena hodnota 6, na pozici 6 je v tabulce odkazů přečtena hodnota 7. Jakýkoliv bod, který patřil objektům 5 a 6 se nyní bude jevit jako objekt č.7. Takto je možné sloučit i složité objekty, které by jinak v tabulce ekvivalentních značek vyžadovali více prostoru. Fig. 4.24 uvádí situaci k pochopení čtenářem, obsah paměti předešlého řádku se během slučování nemění.

Vlastní slučování objektů je jen algebraický součet a nulování, v bližším kontextu bude zmíněn v dalším textu.

4.3.2.iv -b Implementace algoritmu

V této podkapitole proberme návrh algoritmu a jeho části v bližším prozkoumání. Pro přehled nejprve rozdělíme návrh na funkční celky a popíšeme kroky nutné ke zpracování jednoho obrazového bodu na úrovni blokového schématu a jeho stavů.

4.3.2.iv -b.i Stavy zpracování, přehled

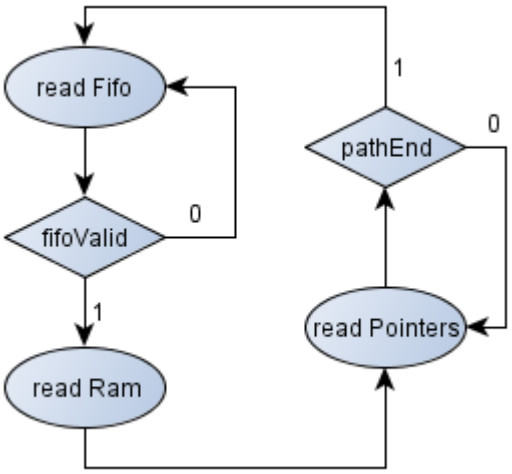


Fig. 4.25 Stavy jádra algoritmu

Bohužel i přes veškerou snahu věnovanou úpravě algoritmu, nelze dosáhnout takové míry paralelismu, aby se data zpracovávala stejně jako v předchozích krocích spolu s taktem obrazového snímače. Situace odráží fakt, že ke zpracování daného obrazového bodu musí být jasný výsledek zpracování bodu předchozího. Neboli že zpracovávání se pozastaví a na výsledek čeká, přičemž dopředu nevíme ani kolik taktů hodinového signálu bude zpracování daného bodu vyžadovat. Ilustrací k následujícímu textu je fig. 4.26.

Do cesty obrazovým datům „PixelData“ je zařazena vstupní fronta s pamětí až pro 16 obrazových bodů, která je plněna s taktem obrazového snímače, ale čtena hodinovým signálem „DcmClk“ s kmitočtem řádově vyšším. Tento signál je získán zapojením a konfigurací jednoho z DCM jader, jeho kmitočet je zvolen s odhadem na složitost obrazu. Nejrychlejší možný průchod stavů automatu vyžaduje tři takty, složité objekty si

ale mohou po jejich slučování žádat taktů mnohem více. Po úvaze nad velikostí vstupní fronty, ponecháním dostatečné rezervy a realizovatelností návrhu, je kmitočet volen jako šestinásobek kmitočtu čtení obrazových dat. Veškeré signály v následujícím návrhu jsou synchronní s hranami signálu „DcmClk“, pro čitelnost není tento signál v blokovém schématu zahrnut.

Názvy stavů prakticky odrážejí úkon, který se během nich vykonává. Začneme ve stavu „st_readFifo“. Tento úkon je spojen s nutností zpracovávat data rychlejším taktem hodinového signálu, než jakým data přicházejí z obrazového snímače. Během tohoto stavu je aktivní signál „FifoEnable“ a čeká se na platná data. Obsahuje-li fronta platná data, jsou přečtena při vzestupné hraně hodinového signálu a spolu s aktivním signálem „FifoValid“ zprostředkována na její výstup. Logika stavového automatu se posune a při sestupné hraně přechází do následujícího stavu.

Tím je stav „st_readRam“. Tento stav je ve zpracování ten nejjednodušší, jeho jedinou úlohou je přečíst label z paměti předešlého řádku. Adresa pro čtení a zápis je součástí vstupních dat, ukázalo se velmi příhodné řídit i tuto paměť povolovacím signálem vstupní fronty. Přechod do dalšího stavu nastává ihned při příští sestupné hraně hodinového signálu.

Kritický stav pro výše popsané fungování algoritmu je „st_readPointers“. Na začátku tohoto stavu nastavíme skrze signál „muxSel“ počáteční label, od kterého poté procházíme orientovanou cestu v tabulce odkazů, obdobně jak popisuje příklad v fig. 4.24. S každou vzestupnou hranou hodin se posuneme o jednu hranu dále. Label získaný v každém takovém kroku je bezprostředně přiveden na výstup jako signál „LabelRam“. Ten je odtud převzat k rozhodovacímu stromu a dle jeho vnitřní logiky okamžitě ovlivňuje výsledek ohodnocení. Synchronnost a řízení logiky pro určování labelů je řízena signálem „DtClk“, který je nulový právě po dobu trvání tohoto stavu. Mimo jiné se dá tento signál chápat jako příznak dokončeného ohodnocování. Nevede-li dál žádná další hrana, je aktivován signál „pathEnd“ a s další sestupnou hranou se vracíme do výchozího stavu.

Navrácení se do stavu „st_readFifo“ je spojeno s uložením právě získaného labelu do paměti předešlého řádku a spuštěním sekvence kroků pro výpočet objektu. Proces slučování a výpočtu objektu pracuje téměř nezávisle. Je odstartován náběžnou hranou signálu „DtClk“ (opuštění stavu „st_readPtr“), ke které jsou synchronní i signály obsažené v procesu. K těmto signálům patří labely a příznaky jejich souvislosti (ilustrovány jako signál „Link“). Všechny potřebné operace jsou poté provedeny během dvou následujících taktů.

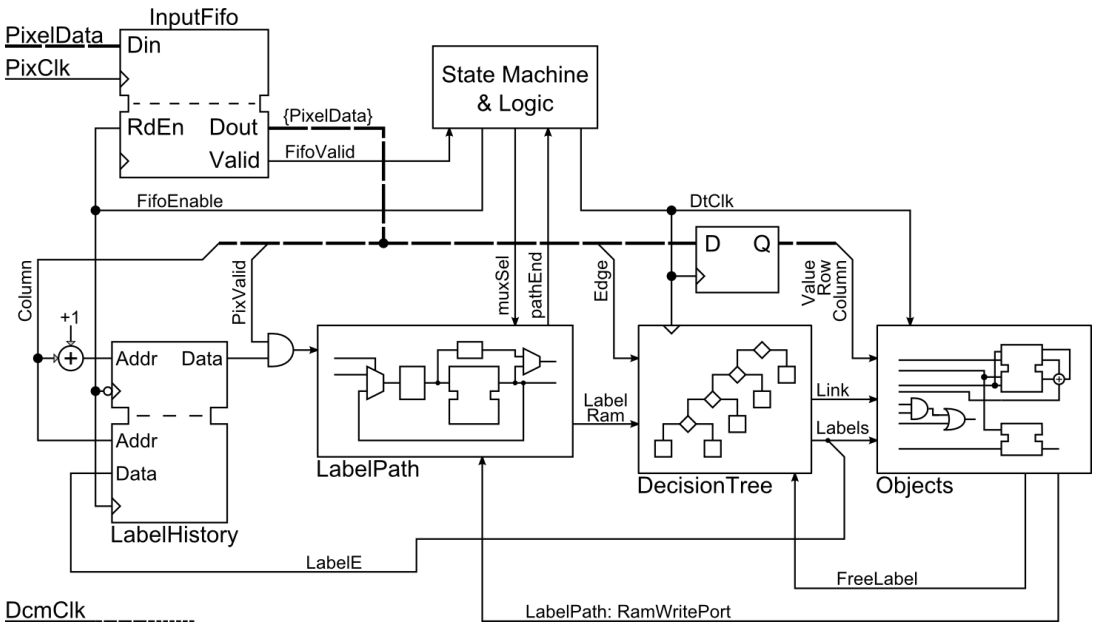


Fig. 4.26 Blokové schéma algoritmu

4.3.2.iv -b.ii Zapojení – popis implementace

K blokovému schématu v fig. 4.26 patří i průběhy signálů a jejich návaznosti (fig. 4.33). Záměrně ale odsuneme tento přehled na později, bez pochopení jednotlivých funkčních celků by mohl působit v krajním případě až nahodile. Další nutnou poznámkou je fakt, že v rámci optimalizace na počet taktů je využíváno vzestupných i sestupných hran hodinových signálů a vcelku složité vzájemné provázání některých celků. Přesto většina komponent je aktivní při vzestupných hranách, přechody stavového automatu naopak vždy při sestupných hranách.

Funkční blok „LabelPath“

Bližším popisem začneme u funkčního celku „LabelPath“, který má za úkol projít odkazy orientované cesty. Jeho zapojení včetně průběhů ilustruje fig. 4.27.

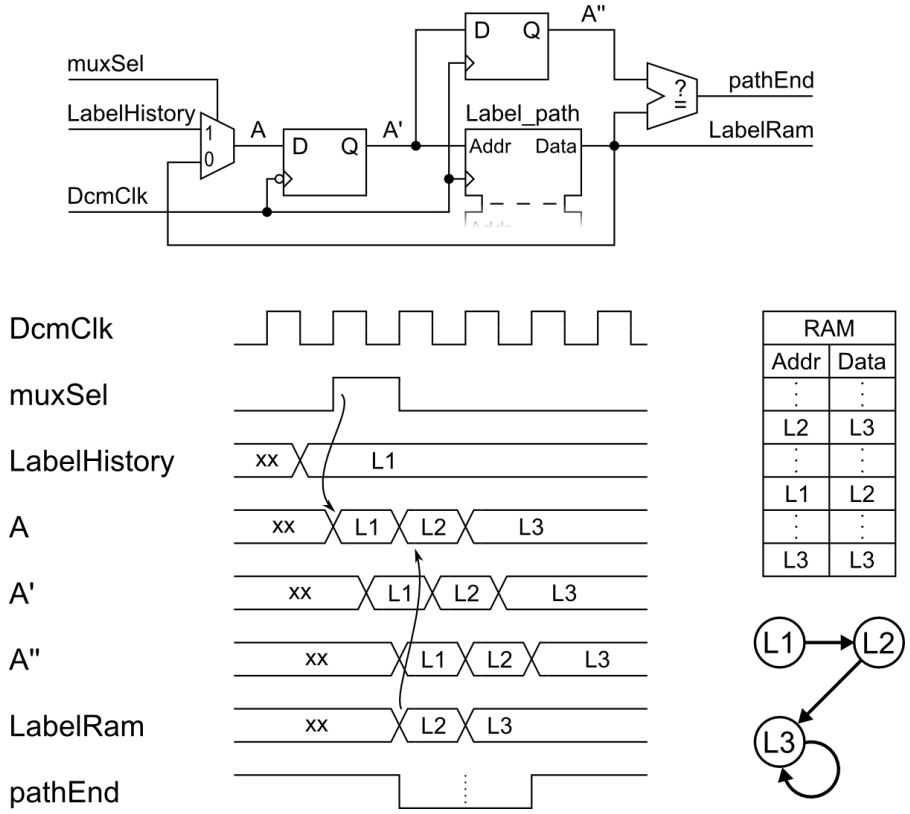


Fig. 4.27 Schéma a signály procházení orientované cesty

Jak je v návrhu patrné, dochází zde k uzavření smyčky kolem čtecího portu paměti s odkazy. Za výchozí stav považujme situaci, kde nevede hrana k žádnému dalšímu labelu, tj. kde položka paměti odkazuje sama na sebe. V tomto stavu se signály nemění a data na výstupu jsou platná.

Chceme-li restartovat procházení cesty, uvedeme její počáteční label „LabelHistory“ (název signálu je odvozen od jeho zdroje) a signálem „muxSel“ rozepneme smyčku adresy minimálně na dobu jedné sestupné hrany hodinového signálu. Během této hrany je adresa uložena na vstupní bránu paměti **A‘**. Následující vzestupná hrana přečte obsah paměti a porovná přečtená data „LabelRam“ s jejich adresou **A‘‘.** V případě shody je aktivován signál „PathEnd“ označující nalezení konce orientované cesty, v opačném případě je po opětovném uzavření smyčky (neaktivní signál „muxSel“) přivedena hodnota labelu skrze multiplexor zpět na adresní piny **A**. Klopný obvod mezi adresami **A** a **A‘** je vložem především jako pojistka proti hazardním stavům při čtení paměti.

Průběhy znázorněné v fig. 4.27 ilustrují průchod orientovanou cestou o délce dvou hran. Během aktivního signálu „muxSel“ je ukazatel (pro představu signál **A‘**) posunut na label L1 a při vzestupné hraně nalezena hrana vedoucí z L1 do L2. Následující sestupnou hranou je ukazatel posunut na pozici L2 a při vzestupné hraně nalezena hrana z L2 do L3. Při sestupné hraně se ukazatel posune na pozici L3, ve které je při vzestupné hraně nalezena hrana sama do sebe. Rovnost adresy a dat implikuje konec cesty, výstupem komparátoru pak bude aktivní signál „PathEnd“.

Funkční blok „DecisionTree“

Implementace rozhodovacího stromu se z pohledu množství logiky může zdát složitá, nicméně tato náročnost je dána prakticky pouze množstvím signálů a jejich multiplexováním. V jádru je zapotřebí pouze dvou kroků, nejprve si připravit všechny vstupní signály (labely A, B, C a D dle fig. 4.21-a) a poté je vyhodnotit podle logiky diskutované v fig. 4.23.

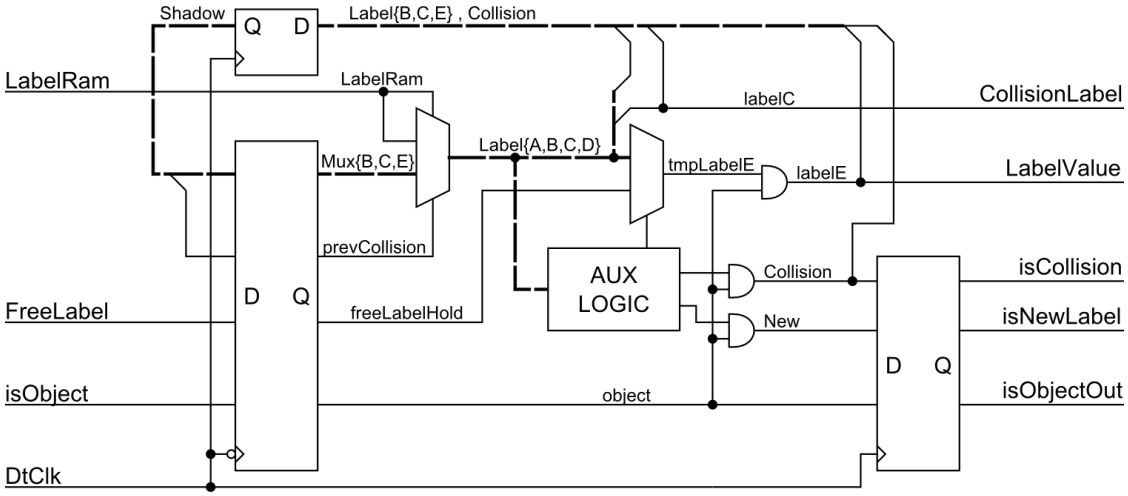


Fig. 4.28 Blokové schéma rozhodovacího stromu

Z ilustrace fig. 4.28 se nejprve podívejme na přípravu labelů. V tomto přiblížení jsou pro nás podstatné signály značené jako „Mux“, které jsou skrze multiplexory převáděny na trojici labelů k pozicím A, B a D. Tyto signály jsou uloženými kopiemi z předchozího cyklu. Při vzestupné hraně signálu „DtClk“ jsou labely z pozic B, C a E uloženy spolu s příznakem kolize jako „Shadow“ signály. Z této kopie jsou data vyvolána při sestupné hraně „DtClk“.

Nejjednodušší je získat labely na pozicích A a D, neboť při posunutí o pixel vpravo přímo odpovídají labelům pozic B a E z předchozího cyklu. Získání zbylých dvou labelů se značně komplikuje v případě, že bylo v předchozím bodě zaznamenáno slučování objektů. Toto slučování indikuje signál „prevCollision“. Získání labelu pro pozici B se řídí výhradně tímto signálem, pokud v předchozím bodě nebyla zaznamenána kolize objektů, zůstala platná i hodnota labelu na pozici C a je možné ji použít jako nový label pozice B. Došlo-li ale ke sloučení objektů, label z C zanikl a není možné ho použít. Řešením by tedy mohlo být znovu přečíst tabulku odkazů (včetně právě přidané hrany), obnovit label C a stejně jako u labelů pozic B a E ho posunout o jeden bod vlevo. Zde ale využijeme skutečnosti, že ke kolizi a tím možnému zániku labelu C mohlo dojít pouze v případě, kde zpracovaný obrazový bod byl součástí nějakého objektu. Tím existoval nenulový label v E, na který bude label z C odkazovat. Mnohem elegantnějším a efektivnějším způsobem je tedy vložit na pozici B přímo předchozí výsledek pozice E.

Poslední nevyřešené je plnění labelu na pozici C. Pro ten je v každém obrazovém bodě vyčten label z paměti předešlého řádku, z něj je procházena tabulka odkazů a výsledek je jako signál „LabelRam“ přiveden k rozhodovacímu stromu. Tato konfigurace je prakticky sama o sobě schopna zajistit správnou funkci, nicméně je zde ponechána funkcionalita navržená již při jednom z předchozích testovaných zapojení.

Tato funkce sloužila při předzpracovávání obrazových dat. Ve snaze o vysokou míru paralelního zpracování byla snaha přečíst a připravit obrazový bod ještě před dokončeným zpracováním bodu předchozího. V takovém zapojení bylo zapotřebí vyvinout mechanizmus, který i v případě slučování objektů (a zániku příslušného labelu) zajistí správná vstupní data pro rozhodovací strom. Toto řešení stojí přinejmenším za zmínku, neboť i práce na něm přinesla důležité poznatky, které se pozitivně promítly do tvaru a funkce rozhodovacího stromu.

Formulace problému je jednoduchá. Začneme-li zpracovávat obrazový bod ještě před dokončením předchozího, riskujeme, že při změně dat nebude algoritmus schopen korektně reagovat. Konkrétně v situaci slučování objektů, kde by výsledek předchozí kolize byl požadován dříve, než je znám. Obrazový bod bezprostředně po sloučení objektů by neměl platná data a došlo by ke ztrátě nebo zkreslení informací. Právě odtud pochází klíčová myšlenka přednosti zleva diskutovaná v předchozím textu. Jeden z jejich důsledků byl, že souvislý objekt je ohodnocen jedním labelem po celé jeho šířce, tedy že pokud jsme schopni určit hodnotu labelu pro pozici B, pak přímo sousedící pozice C musí mít již z principu hodnotu totožnou.

Nastane-li situace, kdy po sloučení objektů existuje nenulový „LabelRam“ a není jisté, zda je přečten správně, může být nahrazen stejně jako label pro B z předchozí hodnoty labelu pozice E. Funkce byla navržena pro zabezpečení jednoho obrazového bodu proti výpadku dat, ve stávajícím zapojení by ji bylo možno rozšířit a využít jako doplněk tabulky odkazů a redukovat její časové nároky u souvislých objektů.

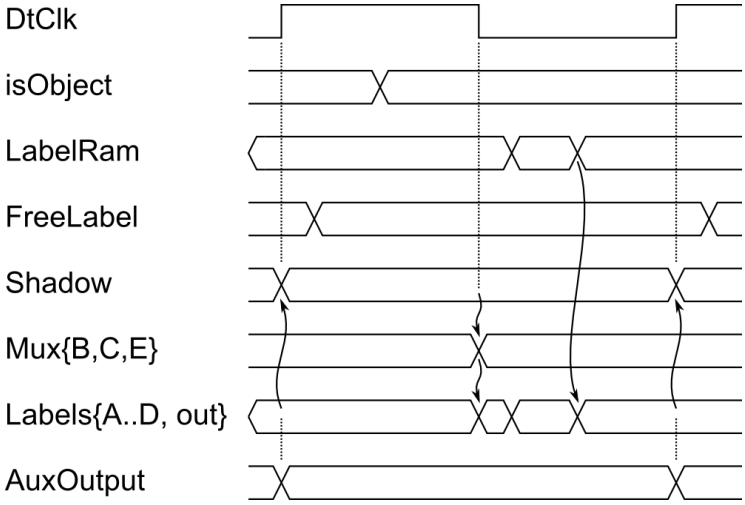


Fig. 4.29 Průběhy signálů rozhodovacího stromu

V pohledu na rozhodovací strom jako celek si představme průběhy význačných signálů, jak jsou zobrazeny v fig. 4.29. Proces je synchronní dle signálu „DtClk“. Z vnějšího pohledu popíšeme funkci ve čtyřech po sobě jdoucích bodech.

Nejprve vzestupná hrana „DtClk“. Vzestupná hrana znamená okamžik, ve kterém jsou všechny signály ustáleny, a proces vyhodnocování skončil. Signály jsou zprostředkovány na výstup a uloženy do svých kopií. Především jde o signály „AuxOutput“, které nesou informaci o přítomnosti objektu a jeho relaci s okolními pozicemi, a výše zmíněné signály „Shadow“. Labely pozic C a E, které jsou primárními výstupy tohoto bloku, byly původně překlápěny na výstup spolu s ostatními signály, ale

díky faktu že se po vzestupné hraně nemění, je můžeme vyvést již dříve a dopřát si tak více času pro překlápění navazující logiky.

Druhý bod je během držení úrovně signálu „DtClk“ v logické 1. Během celé této doby jsou výstupní signály platné a neměnné, naproti tomu může dojít ke změně vstupních signálů „FreeLabel“ po alokaci volného labelu, nebo signálu „isObject“ obsaženého v obrazových datech.

Z tohoto důvodu jsou tyto vstupní signály vzorkovány při sestupné hraně, aby byly platné během procesu a i v jeho výstupu. Sestupná hrana signálu „DtClk“ také zkopíruje signály „Shadow“ do signálů „Mux“, ze kterých se výše popsanými procesy odvodí hodnoty labelů.

Poslední bod je trvání úrovně logická 0. Během této doby se překlápí vnitřní logika dle vstupního signálu „LabelRam“ a uložených signálů „Mux“. Signál „LabelRam“ je jediný vstupní signál, který ovlivňuje výsledek mimo synchronnost se signálem „DtClk“.

V úvodu jsme nastínili, že je v tomto bloku spousta logiky, ale zatím jsme si ji nijak nepřiblížili. Pro zvídavé čtenáře je jádro bloku detailně zakresleno v fig. 4.30.

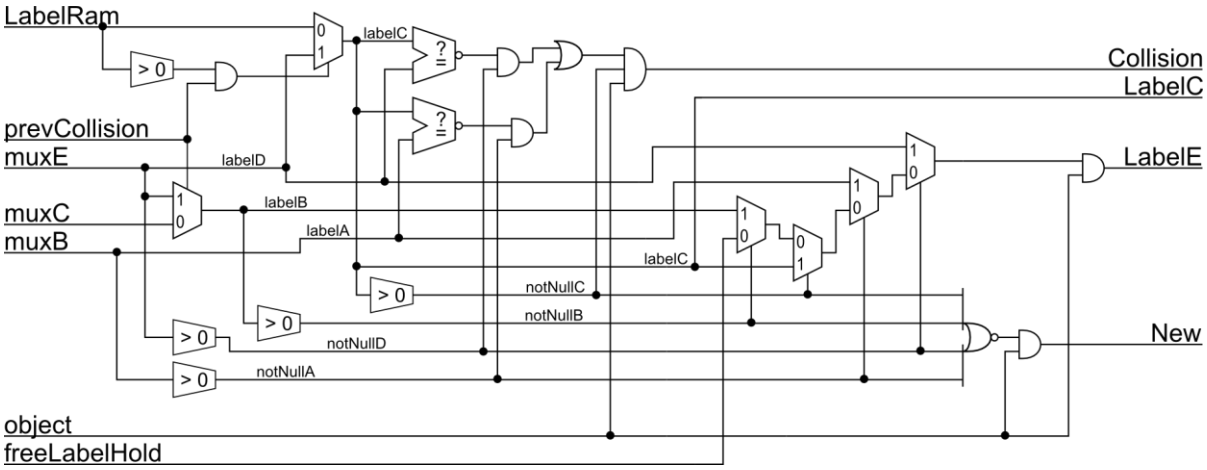


Fig. 4.30 Schéma rozhodovacího stromu – vnitřní logika

Toto schéma je alternativou k fig. 4.28. Pouze zde chybí klopné obvody, které by byly zakresleny totožně jako v předchozím případě, ale značně by kresbu znepřehlednily. Nyní bude snadné zrekapitulovat zpracování a popsat funkci „AuxOutput“ signálů. Těmi je myšleno výstupní signály „isObjectOut“, „isNewLabel“ a „isCollision“ jak jsou zakresleny v fig. 4.28.

V levé třetině se nachází logika multiplexorů pro určení labelů na pozicích A, B, C, D a čtveřice komparátorů vyhodnocující jejich nenulovost. Prostřední třetina patří k řešení kolizí. Kolize může nastat mezi dvojicemi pozic A~C a D~C, labely na těchto pozicích jsou vzájemně porovnány a v případě rozdílných hodnot generován aktivní výstup komparátoru. Ten sám o sobě nestačí, kolize dvou objektů má smysl pouze v případě, že se na zúčastněných pozicích skutečně nachází objekty. Signály komparátorů jsou blokovány skrze signály „NotNull“ dokud se všechny pozice nutné k hlášení kolize nezaplní objekty (jejich label bude nenulový). Teprve splněním všech podmínek kolizního stavu je aktivován signál „Collision“.

Pravá třetina je vlastní aplikace rozhodovacího stromu, jedná se prakticky jen o soustavu multiplexorů, která je seřazena v posloupnosti přesně v rámci stanovených

priorit. V pravém dolním rohu se nachází logika posledního signálu „New“. Tento signál má za úkol signalizovat stav, kdy byl přítomný objekt, ale v žádné jeho okolní pozici se objekt nenachází. Takovému objektu je totiž přiřazen nový label ze signálu „FreeLabelHold“, čímž se ale stává tento signál neplatným a je nutno ho obnovit dalším volným labelem v pořadí. Navazující logika musí na tento signál reagovat a dodat platná data v co nejkratším čase.

Funkční blok „Objects“

Poslední celek z jádra algoritmu je jakási „správa paměti“. Primární úlohou tohoto bloku je ukládání informací o nalezených objektech, v druhé řadě je tento celek odpovědný za uvolňování nepotřebných labelů a vytváření orientovaných cest v tabulce odkazů („LabelPath“).

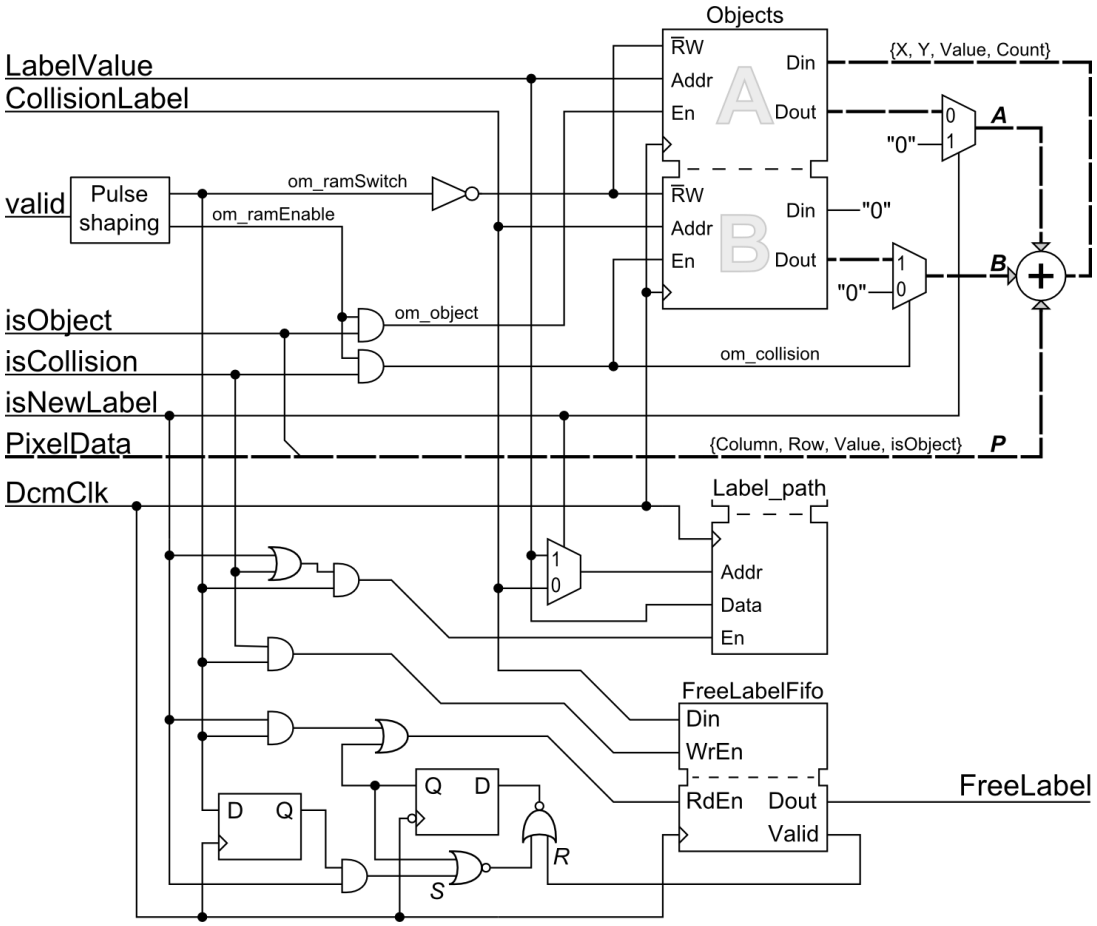


Fig. 4.31 Blokové schéma – správa objektů a paměti

Popišme nyní zapojení z fig. 4.31. Horní polovina zapojení je obsluha paměti, která v sobě ukládá veškeré informace o nalezených objektech. Velikost adresního prostoru je totožná s počtem používaných labelů, aktuálně tedy 0 až 127. Prakticky můžeme nyní na labely pohlížet jako na ukazatele. Nultá adresa je vyhrazena pro hrany objektů a její obsah je vždy nulový. Práce s daty na ostatních adresách se řídí jednoduchými pravidly.

Každý objekt je v paměti reprezentován součtem sloupcových souřadnic *X*, součtem řádkových souřadnic *Y*, součtem jasu *Value* a počtem obrazových bodů tvořících daný objekt *Count*. Je žádoucí, aby všechny položky byly obdrženy prostým součtem, pak lze

totiž i slučování objektů provést prostým součtem. Pro jednoduchost budeme na obrazový bod pohlížet také jako na objekt, který je však tvořen jen jedním bodem. Použití takového tvrzení nám umožňuje zavést, že vstupem do sčítačky jsou tři objekty a nemusíme se v úvahách zabývat rozdílnými formáty. Ukažme si, jak sčítačka a paměť v principu pracuje.

Začneme u signálu „LabelValue“. Tento signál nese výsledek z rozhodovacího stromu, jedná se o label přiřazený právě zpracovávanému obrazovému bodu a právě nad objektem, kterému tento label přísluší se budou provádět početní operace. Prvním portem paměti, označme ho jako port A, čteme data tohoto objektu. V případě, že má objekt teprve vzniknout a toto bude jeho první zápis, jsou data v paměti nulová. Jako nezávislá pojistka je v cestě zařazen ještě multiplexor řízený signálem „isNewLabel“, který u nových objektů případná data zablokuje. Data již existujících objektů jsou přivedena do sčítačky v nezměněném tvaru.

V jistém smyslu podobná situace panuje i u druhého portu paměti, nechť je označen jako port B. Adresou pro tento port je label „CollisionLabel“. Jeho hodnota může být nulová, stejná jako „LabelValue“ nebo rozdílná. Jen v posledním případě se skutečně jedná o kolizi dvou objektů a signál „isCollision“ bude aktivní. Tímto signálem je povoleno čtení dat z portu B na adrese kolidujícího objektu a propuštění výstupních dat skrze multiplexor na vstupu sčítačky. Tento multiplexor zajišťuje nulovost dat z portu B v jakémkoliv nekolizním stavu.

Objekty, které již existují, chceme sečíst s novým objektem. Jeho data obdržíme ze vstupních signálů „PixelData“ a přivedeme na třetí vstup sčítačky. Tento objekt není třeba nulovat či kontrolovat, protože jakákoliv operace včetně slučování objektů probíhá již z principu pouze za jeho přítomnosti. Není-li tento objekt přítomen a signál „isObject“ není aktivní, nedojde ani k povolení portu A a žádná z operací neprobíhá. Port B je v takovém případě také zakázán, neboť signál „isCollision“, který ji povoluje, je již v předchozím bloku přímo svázán s aktivním signálem „isObject“.

Tím jsme vysvětlili výstupy paměti a vstupy sčítačky, nyní se podívejme na signály vedoucí opačným směrem. Sčítačkou provedeme prostý součet všech tří objektů, tj. pro každou ze čtyř položek dat provedeme součet tří položek. Výsledek reprezentuje objekt, který obsahuje všechny předchozí. Staré objekty nyní představují duplicitu a je třeba se jich zbavit. Objekt s labelem „LabelValue“ má být výsledkem této operace, jednoduše přivedeme výstup sčítačky na vstup portu A a zapíšeme do tohoto objektu nová data. Objekt na adrese „CollisionLabel“ byl určen ke sloučení, jeho label bude uvolněn a data vymazána. Nejtriviálnějším řešením je zapsání nul do portu B. Třetí objekt byl reprezentován pouze ze vstupních dat a není třeba se jím zabývat, v příštím cyklu bude sám nahrazen.

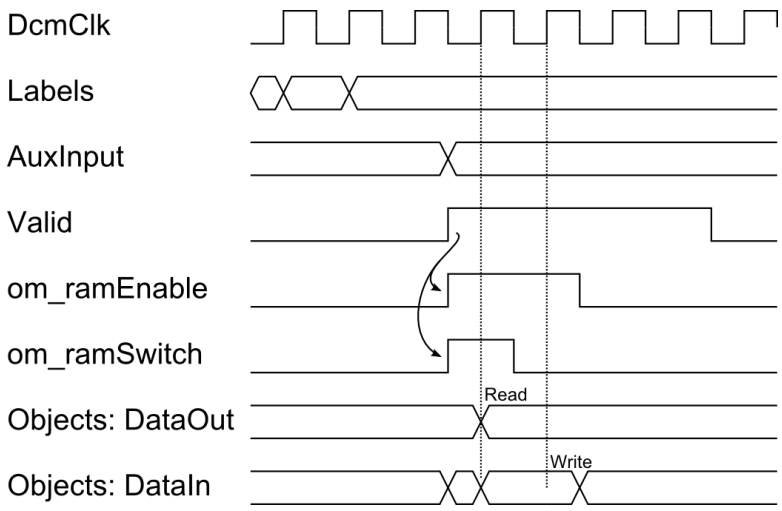


Fig. 4.32 Správa objektů - průběhy signálů

V ilustraci fig. 4.32 jsou zachyceny tvary vstupních signálů, průběh řídících signálů a znázornění přístupu do paměti.

Pro úplnost je do průběhů zakreslen i měnící se signál „LabelValue“ ještě před náběžnou hranou signálu „Valid“ („DtClk“ z předchozího bloku). Po náběžné hraně signálu „Valid“ jsou k dispozici vyhodnocené labely a jejich relace k okolí („AuxInput“), zde začneme popisovat časování výše popsaných operací.

Změna vstupních signálů připadá na sestupné hrany hodin „DcmClk“, operace v pamětích záměrně provádíme při vzestupných hranách. Přístup do pamětí je řízen signály „om_ramEnable“ a „om_ramSwitch“, které jsou generovány v reakci na vzestupnou hranu signálu „Valid“. Signál „om_ramEnable“ je aktivní pro dvě následující náběžné hrany a povolí provést s pamětí „Objects“ právě dvě operace. Druhý signál „om_ramSwitch“ je aktivní pouze polovinu této doby a slouží jako přepínač mezi první a druhou prováděnou operací.

V souladu s výše popsaným principem je při první vzestupné hraně provedena operace čtení, na základě vstupních signálů „isObject“, „isCollision“ a povolovacího signálu „om_ramEnable“ jsou aktivovány příslušné porty paměti a čtena data objektů na adresách svých labelů. Výstupní signály procházejí skrze multiplexory do sčítačky, kde se okamžitě vyhodnotí a jsou přivedeny zpět na port A paměti. Tím operace čtení končí, při sestupné hraně se mění signál „om_ramSwitch“ a přepíná paměť do režimu zápisu. Nyní vstupní data portu A obsahují sečtený objekt a vstupní data portu B jsou nulová. Při druhé vzestupné hraně je proveden zápis těchto dat na aktivních portech paměti „Objects“.

Nechť jsou objekty čtené z paměti označené dle svých portů jako ***A*** a ***B*** a zpracovávaný obrazový bod jako objekt ***P***. Adresy, na kterých operace probíhají, označme zkratkami *LV* pro label „LabelValue“ a *CL* jako label „CollisionLabel“. Pak lze všechny možnosti zapsat do tabulky tab. 4.1.

AuxInput			Operace č.1 (čtení) om_ramSwitch = 1		Operace č.2 (zápis) om_ramSwitch = 0	
Object	**New**	**Collision**	**Port A**	**Port B**	**Port A**	**Port B**
0	0	0	NOP	NOP*	NOP	NOP
1	0	0	A = Obj(LV)	NOP*	Obj(LV) = A+P	NOP
1	**1**	0	A = 0	NOP*	Obj(LV) = P	NOP
1	0	**1**	A = Obj(LV)	B = Obj(CL)	Obj(LV) = A+B+P	Obj(CL) = 0

*) data portu B jsou nulována externí logikou

Tab. 4.1 Operace s objekty

V předchozím textu již bylo také zmíněno uvolňování labelů a správa orientovaných cest v tabulce odkazů, této problematice patří celá spodní polovina schématu fig. 4.31. Podívejme se zblízka na druhý z portů paměti „LabelPath“ (první je zakreslen v fig. 4.27). Tento port je zapojen výhradně pro účel zápisu, který může nastat ve dvou případech.

První je situace, kdy při slučování objektů chceme vložit do tabulky odkazů další hranu. Pak musíme na adresu zanikajícího labelu zapsat hodnotu labelu, který ho nahrazuje. V našem případě tedy do adresy „CollisionLabel“ zapisujeme hodnotu „LabelValue“.

Druhý případ je při přiřazení volného labelu. Label, který byl v tomto obrazovém bodě právě obsazen se stává aktivní součástí svého vlastního a nově vzniklého objektu. To znamená, že je nepřípustné, aby z něj existovala vedoucí hrana, která by ho odkazovala na objekt jiný. Doplníme tedy obsazení volného labelu o zápis odkazu sama na sebe, tedy na adresu „LabelValue“ zapíšeme hodnotu „LabelValue“. Jistě neušlo pozornosti, že právě tuto situaci hledáme v bloku „LabelPath“ jako příznak konce orientované cesty.

Z pohledu signálů je situace jednodušší než při obsluze paměti s objekty, zde provádíme pouze jednu operaci zápisu během jedné vzestupné hrany „DcmClk“. Stačí ve správný okamžik povolit zápis na dobu jednoho taktu, k takovému účelu lze výborně použít signál „om_ramSwitch“. Také si lze všimnout, že data jsou v obou případech stejná a adresa je volena pouze mezi dvěma vstupními labely.

isCollision	isNewLabel	Enable	Adresa (start)	Data (cíl)
0	0	**0**	xxxx	xxxx
0	**1**	**1**	„LabelValue“	„LabelValue“
1	0	**1**	„CollisionLabel“	„LabelValue“

Tab. 4.2 Možnosti vložení do tabulky odkazů

Poslední nediskutovaný úkon je uvolnění labelu. Již mnohokrát bylo v textu odkázáno, že label po sloučení zanikne, ale zatím nikde není vysvětleno jakým způsobem. Podívejme nyní na zapojení fronty „FreeLabelFifo“. Obecně lze popsat tuto frontu jako soubor ukazatelů na prázdná místa v paměti, neboli jako seznam labelů, které lze použít pro uložení nově nalezených objektů.

Zápis do paměti se řídí podobným pravidlem jako zápis do tabulky odkazů. Při aktivním signálu „isCollision“ je zápis povolen na dobu jednoho taktu signálem

„om_ramSwitch“, kde zapisovanými daty je vždy kolidující label „CollisionLabel“. Provedením zápisu labelu do této fronty je považován za uvolněný. Samozřejmě v tabulce odkazů tento label stále existuje a odkazuje na objekt, se kterým byl sloučen. Otázkou zůstává, zda opětovným použitím labelu nebude tato orientovaná cesta zničena. Logickou odpovědí je „ano, bude“. Je ale podstatné si uvědomit, ze kterého bodu je tato cesta procházena. Odkažme se na popis bloku „LabelPath“, kde je uvedeno, že procházení cesty začíná z labelu „LabelHistory“.

Ve stejný okamžik, ve kterém jsme label uvolnili, jsme z něj také vytvořili hranu v tabulce odkazů. To znamená, že i pokud by se v paměti předešlého řádku vyskytoval zápis tohoto labelu, bude přečten a interpretován jinou hodnotou než svoji vlastní. Tím nenastane situace, při které by byl tento label vyhodnocen a zapsán zpět do paměti předešlého řádku. Pokud v paměti neexistuje zápis tohoto labelu, není v principu možné, aby z něj byla procházena cesta. Paměť obsahuje přesně jeden předchozí řádek, to znamená, že zápis v tabulce odkazů nebude nikdy využit po uplynutí více než doby jednoho řádku od okamžiku sloučení labelu. Princip také zajišťuje, že výskyt jakéhokoliv odkazu vedoucího k tomuto labelu zanikne dříve než poslední uvažovaný výskyt tohoto labelu samotného.

Sečteno a podtrženo, pro správnou funkci nesmí být uvolněný label použit dříve, než po uplynutí doby jednoho řádku. Fronta je typu FiFo, čili k porušení tohoto pravidla by musel být během tohoto řádku přečten celý její dosavadní obsah. S počtem 127 labelů je taková situace uměle konstruovatelná, nicméně v reálné scéně prakticky vyloučena.

Výstup fronty, tedy první volný label v pořadí, je jako signál „FreeLabel“ přiveden přímo k rozhodovacímu stromu. Jeho funkce již byla popsána v předchozím textu. Zde pouze nastíníme problematiku čtení tohoto labelu. Ve většině případů předpokládáme bezproblémové fungování paměti a její okamžité čtení, respektive platná data již po první vzestupné hraně. V takto ideálním případě by stačilo obdobně jako v předchozích případech přivést povolovací signál pouze na dobu jednoho taktu. Paměť a její výstupní logika se ale nemusí vždy chovat ideálně, data nemusí být okamžitě dostupná a výstup může mít nenulovou latenci. Pro zachycení všech případů je čtecí port fronty doplněn o zpětnou vazbu mezi výstupními daty a povolovacím signálem. Jeho chování lze nejlépe přirovnat k synchronnímu RS klopnému obvodu. Výstup tohoto obvodu se aktivuje řídicím signálem tvořeným z kombinace signálů „isNewLabel“ a „om_ramSwitch“, v tomto stavu obvod setrvává, dokud není resetován příznakem platných výstupních dat „Valid“ z výstupu „FreeLabelFifo“. Signál klopného obvodu se sečte s původním řídicím signálem, čímž vznikne puls začínající požadavkem na přečtení dalšího volného labelu a končící jeho úspěšným čtením.

Rekapitulace, spolupráce funkčních bloků

Popsali jsme činnost všech zapojených celků, pojďme zrekapitulovat zpracování obrazového bodu dle návrhu fig. 4.26 spolu se zakreslenými průběhy fig. 4.33.

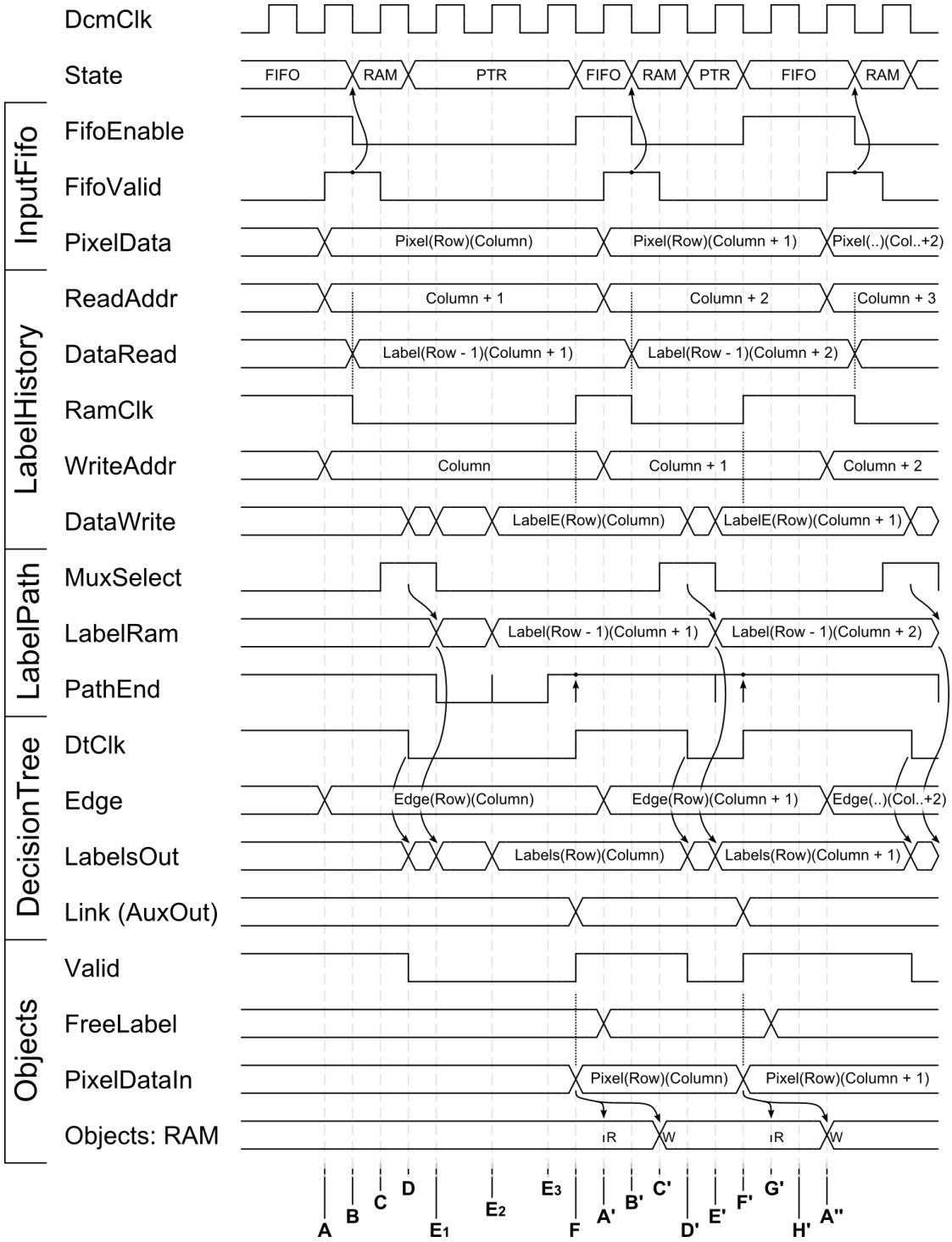


Fig. 4.33 Signály mezi bloky algoritmu

Projdeme si chování obvodů hranu po hraně a vysvětlíme, jak spolu bloky tvoří algoritmus schopný zpracovat obrazová data. V této ukázce chceme demonstrovat, že průchod stavy automatu není vždy stejný. V prvním průchodu se budeme držet situace,

která koresponduje s průběhy znázorněnými v předchozích popisech, v druhém průchodu ukážeme nejrychlejší možný průběh.

Začneme ve stavu „st_readFifo“ a náběžnou hranou hodin **A**. Při této hraně je úspěšně přečtena další položka vstupní fronty „InputFifo“. Úspěšné čtení signalizuje její aktivní signál „FifoValid“. Všechny signály přímo svázané se vstupními daty „PixelData“ se obnoví novými daty. Přejdeme k následující sestupné hraně **B**. Dle aktivního signálu „FifoValid“ dojde k přechodu do stavu „st_readRam“. Signál „FifoEnable“ je svázaný se stavem „st_readFifo“ a stává se neaktivním. Paměť předešlého řádku „LabelHistory“ je také řízena tímto signálem (duplikovaným do signálu „RamClk“), při jeho sestupné hraně proběhne operace čtení a signál „DataRead“ je obnoven z obsahu paměti. Při vzestupné hraně hodin **C** je detekován stav „st_readRam“ a aktivován signál „MuxSelect“, který restartuje procházení cesty „LabelPath“. Také s největší pravděpodobností dojde k resetu signálu „FifoValid“ (chování tohoto signálu je závislé na generovaném IP jádře). Během sestupné hrany **D** přechází automat do stavu „st_readPtr“. S tímto stavem je spojen signál „DtClk“ duplikovaný i do signálu „Valid“ pro správu objektů. Tento signál je ve stavu „st_readPtr“ neaktivní. Při sestupné hraně „DtClk“ se obnoví signály a labely rozhodovacího stromu což může vést ke změně labelů pozic C a E vedených jako „LabelsOut“ a vstupních dat pro zápis v „LabelHistory“. Následuje vzestupná hrana **E1**, zde je v bloku „LabelPath“ přečtena první hrana z orientované cesty, porovnáním dat a adres je zjištěno, že se nejedná o konec cesty a signál „PathEnd“ se stává neaktivním. Změna výstupního signálu „LabelRam“ se také může přímo promítnout do labelů rozhodovacího stromu na pozicích C a E. Při vzestupné hraně ve stavu „st_readPtr“ je také vynulován signál „MuxSelect“. Během hrany **E2** přečteme v tabulce odkazů další hranu. Ve scénáři totožném s fig. 4.27 opět není konec cesty nalezen. Pokračujeme ve vzestupné hraně **E3**, během čtení tabulky odkazů se výstupní data již nezměnila a nyní jsou shodná se svou adresou. Tím je vyhodnocen konec cesty a aktivován signál „PathEnd“. Hrana **F** je první sestupnou hranou od počátku stavu „st_readPtr“ ve které je aktivní signál „PathEnd“. To znamená přechod zpět do stavu „st_readFifo“. Konec stavu „st_readPtr“ znamená vzestupnou hranu signálu „DtClk“, při které jsou pomocné signály z rozhodovacího stromu promítnuty na výstup do signálů „Link“ (popsané též jako „AuxOutput“). Zároveň jsou vybrané signály z „PixelData“ překlopeny na vstupy bloku „Objects“, který v tomto okamžiku začíná svůj vlastní, dva takty trvající proces. Ve stavu „st_readFifo“ se signál „FifoEnable“ stává opět aktivním a jeho vzestupná hrana zapíše do paměti předešlého řádku label z pozice E rozhodovacího stromu.

Tím jsme dokončili jeden průchod algoritmem. Dovolíme si připodobnit druhý průchod k tomu prvnímu a nebudeme zbytečně opakovat všechny změny. Dostali jsme se k vzestupné hraně **A‘**. Krom všeho k čemu došlo i v prvním průchodu je zde aktivní i proces probíhající v bloku „Objects“. Během první vzestupné hrany „DcmClk“ následující po vzestupné hraně „Valid“ („DtClk“), mohou být aktivní operace čtení dat objektů z paměti, zápisu do tabulky odkazů, uvolnění sloučených labelů nebo čtení dalšího volného labelu „FreeLabel“. Objekty jsou sečteny a výsledek připraven k zápisu do paměti. Během hrany **B‘** se v bloku „Objects“ pouze přepne paměť objektů pro zápis. Dostáváme se k hraně **C‘**, během této vzestupné hrany jsou zapsána data sloučeného

objektu zpět do paměti. Teprve zde můžeme prohlásit první průchod algoritmem za úplný.

Chování hrany **D‘** je naprosto totožné s hranou **D.** Naproti tomu při hraně **E‘** chceme demonstrovat různé chování při různých situacích. Během prvního průchodu byly ukázány signály odpovídající procházení cesty o délce dvou hran. Nyní bude demonstrován případ, kdy label čtený z historie nebyl sloučen do žádného dalšího a odkazuje stále sám na sebe. Během hrany **E‘** dochází stejně jako v **E1** ke čtení první hrany z orientované cesty. Rozdílem je, že zde již po prvním čtení jsou data a adresa totožné. Signál „LabelPath“ zůstává i nadále aktivní, čímž se nacházíme v totožné situaci, jako nastala po hraně **E3.**

Další záměrně vložený rozdíl je nastíněn mezi druhým a třetím průchodem. Posuňme se za hranu **F‘.** Ještě při této hraně je situace kopií té předchozí, vracíme se do stavu „st_readFifo“ a správa objektů je na počátku svého procesu. Při vzestupné hraně **G‘** uvažujme situaci, kdy ve vstupní frontě „InputFifo“ nejsou připravena data. Všimněme si, že proces správy objektů pracuje skutečně nezávisle, než dokončí zápis dat z druhého průchodu, zatímco zbytek algoritmu se prakticky zastaví. Nejsou-li na výstupu fronty platná data s aktivním signálem „FifoValid“, stavový automat se při sestupné hraně **H‘** nemůže posunout dále a setrvává v tomto stavu až do prvního úspěšného čtení vstupní fronty (obdoba stavu před první hranou **A**).

Inicializace, výstup dat

Jádro algoritmu je kompletní a funkční, samo o sobě ale stále nestačí. Ve výchozím stavu po konfiguraci hradlového pole jsou paměti a fronty prázdné, nalezeným objektům by nebylo možné přidělit volné labely a uložit jejich data. Stejně tak na druhém konci, kdy již v paměti objektů máme shromážděna všechna data, ale nejsme schopni je použít, neboť existují pouze uvnitř této paměti.

Jako doplněk k našemu algoritmu jsou navrženy stavy inicializace, ve kterých je přebrána kontrola nad přístupem do pamětí, vytvořeny počáteční podmínky pro rozběh a zpracování obrazu a vyčtení dat objektů. Do návrhu je kladena jediná podmínka, činnost inicializace a čtení dat nesmí ovlivnit nebo zastavit jádro algoritmu. Vstupní frontu je nutno neustále číst, aby nedošlo k jejímu zahlcení a výpadku obrazových bodů, naproti tomu musí být operace prováděny v okamžiku, kdy v obrazu nejsou přítomny žádné objekty a jádro nevyžaduje přístup k pamětem.

Inicializace volných labelů a čtení dat vyžaduje více operací než zpracování jednoho obrazového bodu. Nemělo by být s podivem, že stavový automat inicializace (viz fig. 4.34) je složitější než automat jádra.

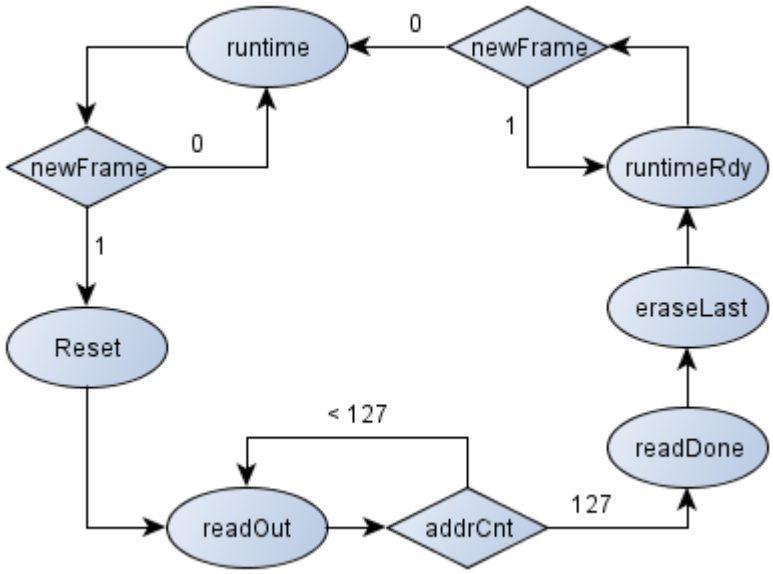
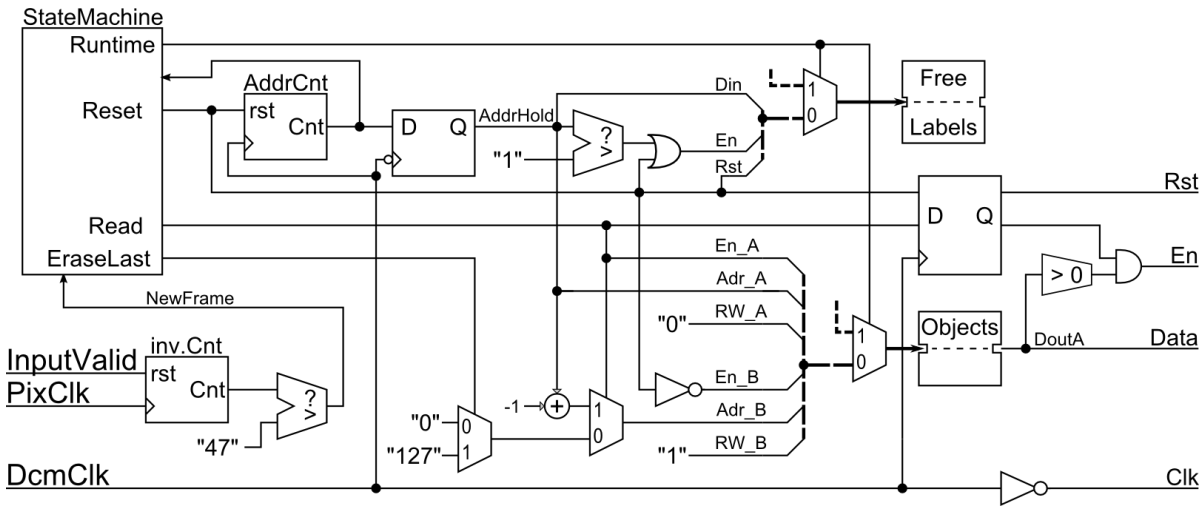


Fig. 4.34 Stavy inicializace

Nejprve bude ve stručnosti popsán jeden průchod inicializací. Nechť je počátkem stav „str_runtime“. V tomto stavu resetovací logika nijak nezasahuje do běhu algoritmu a čeká na začátek příštího snímku. Začátek snímku je detekován signálem „rst_newFrame“. Po detekci nového snímku následuje jeden takt stavu „str_reset“ a přechod do stavů „str_readOut“ a „str_readDone“, ve kterých jsou data objektů vyčtena z paměti a odeslána k výstupu. Během těchto stavů také probíhá příprava pamětí pro další snímek, respektive smazání veškerých nalezených objektů a uvolnění jejich labelů. Poslední položka je dokončena stavem „str_eraseLast“. Za tímto stavem se přístup pamětí přepíná zpět k jádru algoritmu a ve stavu „str_runtimeRdy“ se čeká na začátek snímku. První platná data snímku vrací automat do výchozího stavu „str_runtime“.



Poznámka: Nepropojené vstupy multiplexorů představují původní řídicí signály z jádra algoritmu.

Fig. 4.35 Zapojení inicializace a výstupu dat

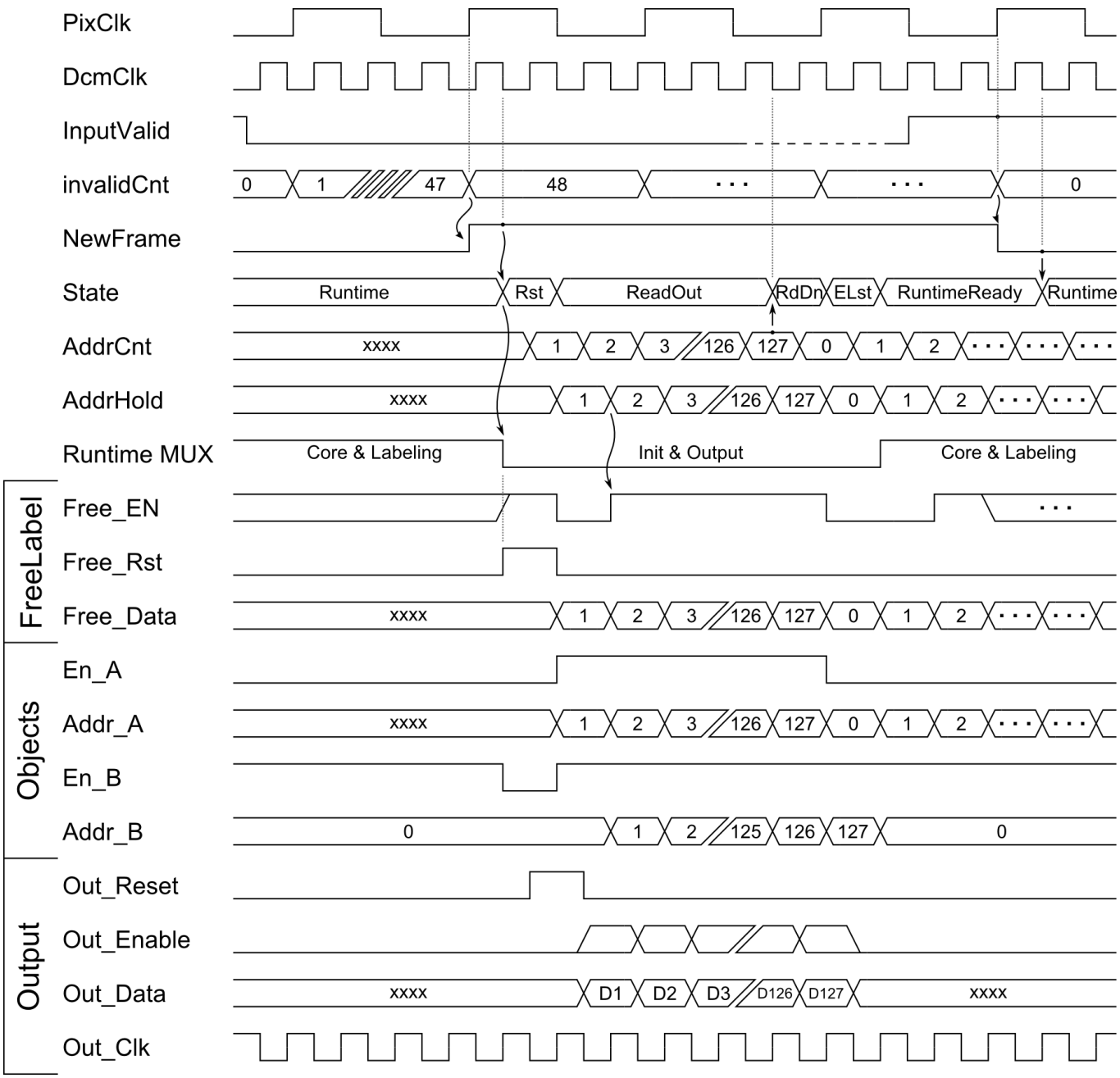


Fig. 4.36 Inicializace a výstup – průběhy signálů

S pomocí fig. 4.35 a fig. 4.36 popíšeme proces podrobněji. Inicializace začíná detekcí nového snímku ze signálu „InputValid“. Tento signál je generován během zpracovávání hran v matici obrazových bodů. Jeho významem je signalizovat platnost nalezených hran při zaplnění celé matice obrazovými body. Využijeme skutečnost, že matice je kompletní až během osmého řádku, tedy po dobu sedmi řádků je tento signál neaktivní. Obdobná situace nastane během každého řádku, kdy je potřeba osmi obrazových bodů k naplnění posledního řádku matice. Pro detekci počátku snímku hledáme výpadek signálu „InputValid“ delší než po dobu osmi obrazových bodů. Nechť tato doba odpovídá trvání 48 obrazových bodů.

Napočítáním více než 47 neplatných obrazových bodů je aktivován signál „NewFrame“ a automat přechází do stavu „str_reset“. Od této chvíle přebírá inicializace

přístup do pamětí „Objects“ a „FreeLabelFifo“. Náběžnou hranou ve stavu „str_reset“ je resetován čítač adres a smazán obsah fronty „FreeLabelFifo“, aby při jejím plnění nedošlo k duplicitám.

V dalších taktech se již dostaneme k inicializačnímu stavu „str_readOut“ a pomocným stavům „str_readDone“ a „str_eraseLast“. V náběžných hranách během těchto stavů jsou čtena data objektů z paměti na jejím portu A. Každá přečtená položka je pak během následujícího taktu vymazána zápisem nul do portu B. Přečtená data jsou opatřena řídícími signály a slouží k plnění výstupní fronty (bude popsána v dalším textu). Adresy čtených dat jsou mezitím ukládány do fronty „FreeLabelFifo“ jako nové odkazy na uvolněnou paměť.

Přečtením a uvolněním celé paměti proces inicializace končí. Přechodem do stavu „str_runtimeReady“ je navrácen přístup k pamětem zpět do jádra algoritmu a čekáme na první platná data. Inicializace trvá 129 taktů „DcmClk“, které odpovídají době nejvýše 22 taktů obrazových dat. S uvážením počátku během 48. obrazového bodu bude inicializace kompletní cca po přijetí 70 obrazových bodů. Už ve srovnání s jakoukoliv uvažovanou šířkou řádku je toto zanedbatelný počet. Tento stav je tedy nezbytnou pojistkou proti opakovanému spouštění procesu inicializace.

Po přivedení platných dat signálem „InputValid“ je vynulován čítač neplatných bodů, tím se stává signál „NewFrame“ neaktivní a automat se vrací do stavu „str_runtime“.

4.3.2.v Krok 3 – Výstupní fronta

Předchozími odstavci byl dokončen popis algoritmu, jeho jádra a podpůrné logiky. Nyní bude popsáno promítnutí dat zpracovaného obrazu na výstupní port hradlového pole.

Signály na výstupu algoritmu (viz signály „Output“ v fig. 4.36) jsou záměrně generované ve tvaru odpovídajícímu pro plnění FiFo fronty. Prakticky je tak definováno rozhraní, které přinejmenším pro hradlová pole Xilinx odděluje blok labelingu a navazující výstupní blok. Budeme-li vyžadovat připojení na různých fyzických vrstvách, lze jejich implementace velmi triviálním způsobem zaměňovat či paralelizovat.

V našem případě se předpokládá připojení běžného mikropočítače. Data budou vysílána po bytech synchronně s hodinovým signálem. Vstupními daty jsou celkem čtyři parametry, každý z nich reprezentován 32 bitovým číslem. Vstupní data jsou vedena jako jedna 128 bitů široká sběrnice, data je třeba rozdělit a převést na posloupnost slov vhodné šířky (bytů) a především zpomalit z taktu „DcmClk“ na přijatelnou mez. Kmitočet výstupních hodin si můžeme zvolit libovolně. Nechť je šířka výstupních dat jeden byte a nejvyšší snímková frekvence 100Hz. Pak je naším cílem odeslat všechna data v rámci 10ms. Jednoduchým výpočtem stanovíme žádaný kmitočet.

$$f_{DR} = \frac{D}{T} = \frac{127\ objektů\ x\ 4\ parametry\ x\ 4\ byty}{10^{-2}} = 203\ 200\ Hz$$

Nechť je za vstup považována již zmíněná FiFo fronta. K té je navržena posloupnost stavů (viz fig. 4.37), která zabezpečí korektní čtení obsahu fronty a souvislý tok výstupních dat. Funkce výstupního bloku (viz fig. 4.38) je vcelku triviální, v klidovém stavu „st_read“ čteme frontu a čekáme na platná data. Po čtení platných dat je jejich nejnižší byte přiveden na výstup ve stavu „st_output“ spolu s uložením vyšších bytů do

klopných obvodů tvořících posuvné registry. V dalších taktech probíhá v rámci stavu „st_shift“ posouvání dat, nejnižší byte je vždy promítnut na výstupní bránu. Při posledním posunutí dat ve stavu „st_shiftLast“ je zároveň provedeno čtení FiFo fronty a porovnán jeho výsledek. Jsou-li na jejím výstupu data platná, okamžitě přecházíme k jejich odeslání, v opačném případě čekáme ve výchozím stavu „st_read“.

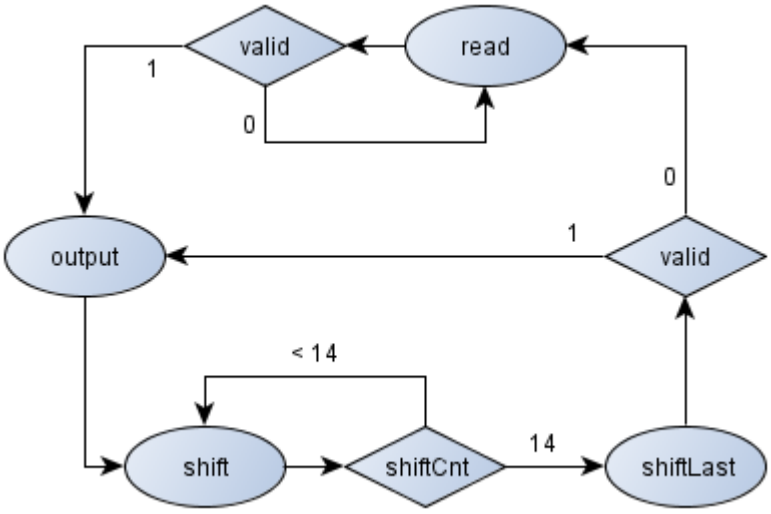


Fig. 4.37 Stavy výstupu dat

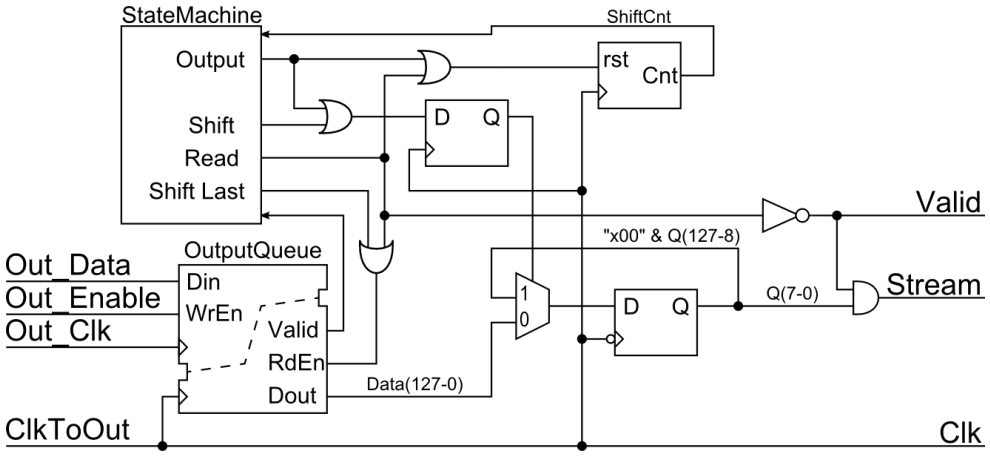


Fig. 4.38 Zapojení výstupního bloku

Výstupní signály „Valid“, „Stream“ a „Clk“ jsou dále vedeny přímo na výstupní piny hradlového pole. Jejich průběh a uspořádání dat vykresluje fig. 4.39. Data parametrů jsou vysílána od nejnižších bytů v pořadí:

1. Součet souřadnic X.
2. Součet souřadnic Y.
3. Plocha objektu – počet obrazových bodů.
4. Součet jasu.

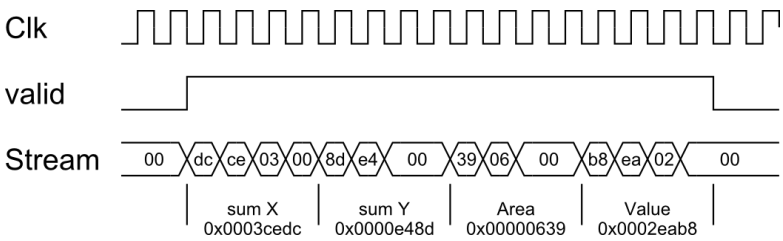


Fig. 4.39 Tvar výstupních signálů FPGA

Výpočet těžiště objektu není součástí námi navrženého zpracování, všechny potřebné hodnoty k jeho vypočtení jsou ale obsaženy ve výstupních datech. Pro úplnost uvedeme přepočet výstupních hodnot do představitelnějších čísel. Všechny obrazové body mají stejnou váhu, výpočet těžiště se pak provádí dle vzorce

$$\vec{\boldsymbol{T}} = \frac{\sum \vec{\boldsymbol{P}}}{A} = \left(\frac{\sum P_x}{A} \quad \frac{\sum P_y}{A}\right) = \left(\frac{0x3cedc}{0x639} \quad \frac{0x0e48d}{0x639}\right) = (156{,}67 \quad 36{,}73)$$

kde vektory $\boldsymbol{P}$ jsou souřadnice obrazových bodů tvořících objekt, hodnota A je plocha objektu (počet obrazových bodů) a vektor $\boldsymbol{T}$ je těžištěm celého objektu.

Ten stejný objekt lze nyní popsat těžištěm v bodě 156,67 na ose X, 36,73 v ose Y, plochou 1593 obrazových bodů a průměrnou hodnotou jasu v ploše objektu 120.

4.3.3 Data pro PC aplikaci, příjem I²C zpráv

Po zvážení možností bylo rozhodnuto využít k nahrání parametrů sběrnici I²C, která již slouží pro nahrávání parametrů do CMOS snímače. K tomuto účelu je napsán stavový automat přijímající I²C transakce na adrese 0x4D. Transakce neobsahují adresní byte registru, ihned po adrese zařízení jsou vysílána data. Transakce může být dlouhá jeden nebo dva byty, delší transakce budou ignorovány. Hodnota přijatého bytu je uložena během vzestupné hrany při potvrzení jeho příjmu (během vysílání „Ack“ bitu). Jednobytovou transakcí je možno nastavit prahovou hodnotu pro určování hran, dvoubytovou transakcí je nastavena prahová hodnota společně s doplňujícím nastavením. (násobek prahové hodnoty a volba dat pro PC aplikaci). Pro detail zprávy a jejího obsahu viz kapitolu 4.2.2.ii -b.

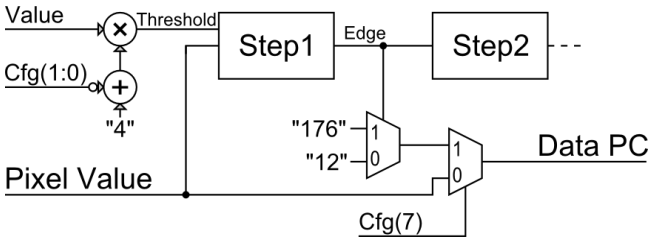
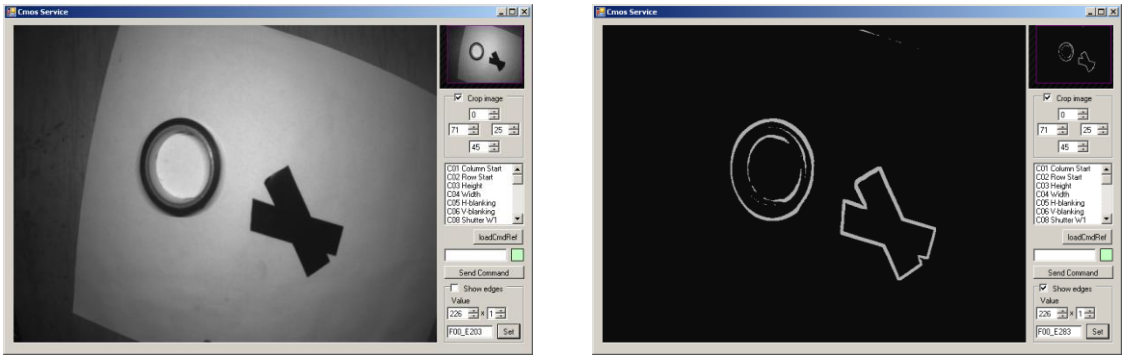


Fig. 4.40 Zapojení signálů z I²C přijímače

Dopad přijatých hodnot na zpracování obrazových dat je ideově načrtnut v fig. 4.40, první přijatý byte nastaví prahovou hodnotu „Value“, druhý přijímaný byte obsahuje doplňující nastavení „Cfg“. Výstupní data, která jsou vedena k USB periferii, je možno odebírat přímo z dat obrazového snímače, nebo jako binární obraz ze signálů určených pro algoritmus labelingu. O výběru rozhoduje stav sedmého bitu „Cfg“ bytu. V případě

výběru binárního obrazu hran je logická úroveň nejprve převedena na hodnotu jasu. Ukázka obrazu přeneseného pro obě varianty nastavení je zachycena v fig. 4.41.



Cfg(7) = 0 Cfg(7) = 1

Fig. 4.41 Přenášený obraz nalezených hran

4.4 Firmware mikropočítače, inicializace komponent

Během vývoje firmwaru nebylo zapotřebí řešit příliš zásadních problémů, nicméně je na místě uvést problém s obsazeností datových pamětí RAM. Data pro procesor jsou ukládána do 16 bytů bitově adresovatelné paměti, 80 bytů přímo i nepřímo adresovatelné paměti a 128 bytů nepřímo adresovatelné paměti. Takto omezený prostor by na ukládání proměnných, dat a struktur obsluhy USB, SD karet a souborového systému pravděpodobně nestačil, naštěstí je v čipu obsažena i 512 bytů XDATA RAM paměť. Společně s tímto prostorem již máme dostatek prostoru pro datové struktury, ale ukazuje se jako značný problém umístění haldy. Množství funkcí a jejich parametrů kladlo vyšší nároky na velikost haldy, než byl dostupný adresní prostor vnitřní RAM a přeložení haldy do XDATA paměti se nezdařilo. S vynaložením úsilí je halda redukována, data přesunuta do prostoru XDATA paměti a program úspěšně spuštěn.

4.4.1 Spuštění mikropočítače, EZ-USB periferie

Po přivedení napájení je spuštěn zavaděč a spustitelný kód je nahrán do vnitřní paměti RAM. Odtud se začne program vykonávat. Inicializace začíná nastavením hodinových signálů, přiřazením funkcí a endpointů s FIFO řadičem a nastavením vstupně výstupních pinů.

```
//set the CPU clock to 48MHz
CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1);
//set FiFo as slave, external clk, rising edge
IFCONFIG = 0x03;
FIFOPINPOLAR = 0x3B;        //SLWR active low
PINFLAGSAB = 0x8C;          //A = EP2FF ; B = EP2EF
//configure endpoints
EP1OUTCFG = 0xA0;
EP1INCFG = 0xA0;
EP2CFG = 0xE0;        //4x512 byte, IN Bulk
EP1OUTBC = 64;        //arm ep1 out
EPIE |= 0x08;         //arm ep1 out IRQ
```

```
//configure fifo
EP2FIFOCFG = 0x00; //core needs to see autoin 0->1 transition to arm
EP2FIFOCFG = 0x0C;   //autoIn, zerolen enable, 8bit wide
EP2AUTOINLENH = 0x02;//512
EP2AUTOINLENL = 0x00;
FIFORESET = 0x82;
FIFORESET = 0x00;
```

Fig. 4.42 Základní prvky inicializace EZ-USB

Nutnost nulování registru *EPxFIFOCFG* před jeho nastavením není v reference manuálu popsána, ale jedná se o velmi podstatný krok ke spuštění FIFO řadiče. Jádro periferie totiž předává kontrolu nad endpointem při náběžné hraně signálu „AutoIn“, pouhé nastavení úrovně tedy nemusí být dostačující. Tato skutečnost měla během oživování komunikace fatální důsledky. Nebyl-li signál „AutoIn“ nulován, nedošlo v drtivé většině případů k aktivaci endpointu a USB periferie nebyla schopna přenášet data. Činnost periferie je řízena externími signály, v našem případě bylo rozhodnuto připojit je staticky (viz fig. 4.1) a řídit tok dat pomocí programu.

Signál	**Aktivní úroveň**	**Význam**
SLOE	HIGH	FIFO Output enable. Brána je použita pouze jako vstupní, signál trvale neaktivní
ADR0	-	Adresování zapisovaného endpointu, nulová adresa připadá endpointu č.2 .
ADR1	-	
PKTEND	HIGH	Odeslání neúplného paketu. Signál trvale neaktivní.
SLRD	HIGH	Čtení příchozích dat z FIFO fronty, trvale neaktivní.
SLWR	LOW	Zápis odchozích dat do FIFO fronty, trvale aktivní.

Tab. 4.3 Význam statických signálů EZ-USB

Nastavení polarit se provede zápisem do registru *FIFOPINPOLAR*. Polarity signálů nastavíme dle tab. 4.3. Při nastavení polarity signálu „SLWR“ na aktivní v nule, dojde k aktivaci čtení dat do endpointu 2 a jejich odesílání. Inicializace probíhá ještě před spuštěním hradlového pole a obrazového snímače, tedy na pin IFCLK není veden hodinový signál. Aktivace FIFO periferie bez hodinového signálu má za následek její zaseknutí. Provedením inicializace až po konfiguraci hradlového pole bychom sice tento problém vyřešili, ale znemožnili bychom enumerovat USB zařízení a použít endpoint 1 pro ladící výpisy. Řešení tohoto problému bude zmíněno níže.

4.4.2 Implementace souborového systému

Pro pokračování a inicializaci ostatních komponent je potřeba implementovat souborový systém. Podklady pro souborový systém jsou převzaty z volně dostupného zdroje *„Petit FAT File System Module“* [14]. Jedná se o minimalistickou verzi souborového systému s velmi malými nároky na použitou paměť a velikost kódu. V plné implementaci je s poskytnutým souborovým systémem možno číst média ve formátech FAT12, FAT16 a FAT32, pohybovat se v adresářích, otevírat a číst soubory nebo do nich s jistými omezeními zapisovat. V základu nejsou součástí implementace rutiny pro přístup na disk či jiné paměťové médium, ale z totožného zdroje lze spolu se souborovým

systémem získat i vzorové aplikace s již vytvořeným přístupem k SD/MMC kartám. Právě taková implementace byla převzata jako vzorová.

4.4.2.i Omezení vývodů mikropočítače, emulace sériových komunikací

V předcházejícím textu (kapitola 3.4.1) byla řešena fyzická vrstva a signály pro sériovou komunikaci mezi SD kartou a mikropočítačem. Pro realizaci transakcí bylo zamýšleno využít SPI periferie mikropočítače, nicméně stejně jako v případě hradlového pole je i zde tlak na redukci návrhu a použití menších a levnějších pouzder. Pro naši aplikaci to znamená, že bude-li to možné, budeme využívat pouze prostředky obvodu Cypress obsažené v 56 pinovém pouzdře. Bohužel 56 pinové pouzdro nemá piny sériové komunikace k dispozici.

V důsledku to znamená, že sériové komunikace bude nutno přivést z jiných vývodů a signály emulovat programem. Konfigurací obvodu do režimu „FIFO slave“ přicházíme o většinu dostupných vývodů a ani pro emulovanou komunikaci jich nezbývá dostatek. Existuje jeden způsob, jakým lze část zabraných pinů opět využít. Sběrnice pro čtení a zápis FIFO řadiče je široká 16 bitů, komunikace po endpointu 2 je ale nastavena pouze na 8 bitů, zbylých 8 není využitých. Nastavíme-li všechny zbývající i nepoužité endpointy do 8 bitového režimu, řadič FIFO uvolní vyšších 8 bitů sběrnice a piny PD0 - PD7 bude možné využívat jako vstupně/výstupní. Dále je možné vhodným nastavením registru *PORTACFG* uvolnit i pin PA7. Realizované zapojení SD/MMC slotu již bylo zobrazeno v předchozích kapitolách (viz fig. 4.2).

4.4.2.ii Vrstvy souborového systému

Implementace souborového systému je rozdělena do více vrstev dle úrovně abstrakce a funkcí, které nabízí. Vlastní souborový systém pracuje nezávisle na připojené paměti a její fyzické vrstvě, obsahuje funkce pro práci se soubory a pro pohyb uvnitř souborového systému. Všechny své funkce poskytuje aplikační vrstvě, na druhou stranu vyžaduje dostupnost funkcí pro čtení sektorů paměti. Vrstva „disk InOut“ představuje abstraktní třídu definující rozhraní mezi souborovým systémem a fyzickou pamětí, obsahuje prototypy funkcí pro otevření paměti a čtení jejího sektoru. Tyto funkce již jsou implementovány přímo nad fyzickou vrstvou („mmcbb.c“) a zpracovávají přístup k sektorům skrze SPI transakce.

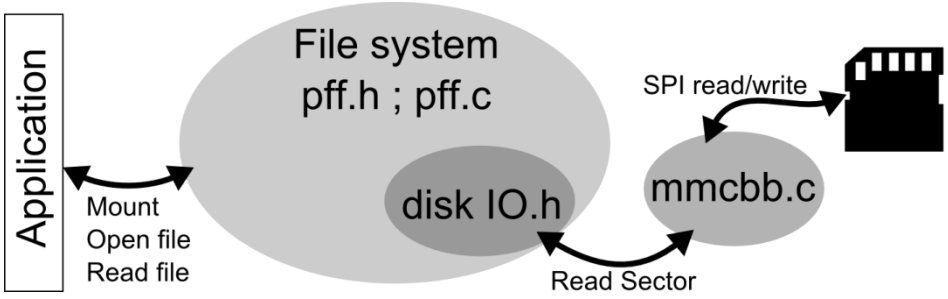


Fig. 4.43 Vrstvy souborového systému, přístup k paměti

4.4.2.iii Úpravy provedené ve zdrojovém kódu Petit FatFs

Do funkce souborového systému byly provedeny jen minimální zásahy, většina změn se týkala umístění datových struktur, minimalizace nároků na použitou paměť a umístění

dat do XDATA paměti. Funkce provádějící SPI transakce byly oproti vzorové implementaci [14] redukovány a jejich přístup přizpůsoben vstupním a výstupním pinům použitého mikropočítače. Funkce, které nebyly nezbytně nutné pro čtení souborů, byly kvůli nárokům na paměť odstraněny. Podpora formátů FAT12 a FAT32 byla prohlášena za nadbytečnou a v rámci uvolnění paměti také odstraněna.

4.4.3 Bootování hradlového pole

V předchozím kroku inicializace byl nakonfigurován FIFO řadič v USB periferii a spuštěna komunikace na endpointech 0 a 1. Obsluha konfigurace hradlového pole je časově náročný proces, během kterého nejsou zpracovávány žádné příchozí zprávy. Má-li dojít k enumeraci zařízení, musí být dokončena ještě před začátkem bootování hradlového pole. Z tohoto důvodu je volání této rutiny přesunuto do zahaleče, kde čeká na uplynutí vloženého intervalu a teprve pak je rutina spuštěna.

Po nastavení vstupních a výstupních pinů je souborovým systémem otevřena externí paměť a v paměti otevřen programovací soubor „***hdl_top.bit***“. Po otevření souboru je zahájena konfigurační sekvence, která je popsána již v kapitole 3.2.4.iii. Data z otevřeného souboru jsou ihned po jejich přečtení odesílána do hradlového pole. Bitsream konfiguračního souboru je nahráván do hradlového pole jako jedna souvislá SPI transakce.

Po nahrání bitstreamu je kontrolován signál „*DONE*“. Je-li tento signál aktivní, konfigurace hradlového pole proběhla úspěšně a inicializace může pokračovat. Pokud dojde kdykoliv během procesu k chybě, je bootování hradlového pole přerušeno a nastaven časový interval, po jehož uplynutí dojde k opakování celého procesu.

4.4.4 Nastavení obrazového snímače

Hradlové pole po svém spuštění generuje hodinový signál pro obrazový snímač. Tím je obrazový snímač spuštěn. Krom generování obrazových dat je snímač nyní aktivní i na I^2C sběrnici a je možné s ním komunikovat. Po startu zařízení nelze zaručit, že generovaný obraz odpovídá našim požadavkům. Z tohoto důvodu je v externí paměti uložen soubor, který obsahuje seznam registrů a jejich požadovaných hodnot.

4.4.4.i Soubor příkazů pro nastavení CMOS snímače

Základními požadavky na formát souboru je jeho čitelnost uživatelem, snadná editace a interpretace. Při implementaci je také vhodné zvážit, zda nelze použít či recyklovat již existující kód a implementaci tak zjednodušit. Ukládání parametrů do registrů obrazového snímače již bylo řešeno výše během vývoje počítačové aplikace, kde se příkazy zasílaly pomocí textových zpráv. Není důvod tento mechanizmus nevyužít.

Formát zpráv je převzat v nezměněném tvaru (viz kapitolu 4.2.2.ii), liší se pouze zdroj textových řetězců. Namísto příjmu z endpointu 1 jsou příkazy čteny ze souboru z externí paměti, každý řádek souboru obsahuje příkaz pro nastavení jednoho registru. Funkcemi souborového systému je otevřen textový soubor „***cfgCmds.txt***“, který je čten po 10 bytových blocích. Prvních osm bytů jsou znaky příkazu, například „C09_0284“, další dva byty jsou znaky <CR> a <LF> ukončující řádek. Přečtený příkaz je okamžitě zpracován, po jeho odeslání je ze souboru přečteno dalších 10 bytů a krok se opakuje. Využitím jednotného formátu zpráv je možné zahrnout do souboru i příkaz pro nastavení

prahové hodnoty hran v hradlovém poli. Nastavení komponent je dokončeno po přečtení celého souboru.

4.4.5 Dokončení inicializace, restart FIFO řadiče, zahaleč

Po dokončení všech kroků inicializace je zařízení připraveno zpracovávat obrazová data a přenášet obraz do aplikace v nadřazeném PC. Nyní se vracíme k výše zmíněnému problému zaseknutí FIFO řadiče, ke kterému může dojít během inicializace USB periferie, pokud není přítomen hodinový signál na pinu IFCLK. Po úspěšném startu a nastavení všech komponent je hodinový signál již generován, polaritu signálu SLWR nyní nastavíme na aktivní v log. 1, čímž signál deaktivujeme. Po uplynutí krátké časové prodlevy znovu zavoláme rutinu inicializace. Jejím provedením je start zařízení dokončen.

Hlavní smyčka programu neboli zahaleč, slouží pro obsluhu SETUP zpráv na nultém endpointu a pro zpracování příkazů na přechod USB zařízení do režimu spánku. Krom hlavní smyčky je aktivní přerušení od příjmu paketu na prvním endpointu, které slouží pro komunikaci s PC aplikací.

5. Přehled odvedené práce

Časová náročnost při realizaci této práce není dána pouze množstvím vykonaných úkonů, ale i širokým záběrem znalostí a různorodostí použitých komponent. V této kapitole je uveden stručný výčet všech provedených prací.

5.1.1 Nastudování teorie a pochopení problematiky

- Nastudování problematiky hradlových polí, seznámení se s jazykem VHDL a vývojovým prostředím ISE WebPack. [1, 9, 15, 16, 17, 22]
- Seznámení se s registry CMOS obrazového snímače, jeho konfigurací a funkcemi příslušných registrů. [7]
- Bližší seznámení s časováním a formátem I^2C transakcí. [8]
- Nastudování EZ-USB periferie, přístupu k endpointům a obsluhy paketů. Seznámení se s použitím EZ-USB ve Windows. [10, 11, 19, 21, 23, 24]
- Seznámení se s jazykem C# a vývojovými nástroji pro aplikace ve Windows.
- Nastudování předchozích prací spojených s podobnou problematikou. [2, 4, 5, 6]
- Nastudování teorie a algoritmů labelingu. [18, 20]
- Seznámení se s implementací souborového systému. [12, 13, 14]

5.1.2 Mikropočítač a EZ-USB Cypress Cy7c68013A

Pro modul EZ-USB byl napsán firmware obsluhující tyto funkce:

- Obsluha USB periferie a komunikace s nadřazeným PC. Odesílání obrazových dat do nadřazeného PC v režimu „Slave FIFO“.
- Hostitel transakcí I^2C sběrnice, odeslání příkazu do CMOS snímače nebo FPGA.
- Sériová komunikace s externí pamětí SD/MMC. Obsluha souborového systému FAT, čtení souborů z paměťové karty SD/MMC.
- Sériové nahrávání konfiguračních dat ze souboru do FPGA a jeho spuštění.
- Nastavení parametrů obrazového snímače dle souboru v externí paměti.

5.1.3 Hradlové pole Spartan-3 XC3s200

Chování logiky bylo popsáno jazykem VHDL, navrženými obvody jsou realizovány hlavní funkce:

- Zpracování signálů z CMOS obrazového snímače, výpočet souřadnic.
- Nalezení hran v obraze a vytvoření binárního obrazu.
- Zpracování algoritmu – labeling.
  - Hlavní a přidružené stavové automaty
  - Zpracovávání na vyšším kmitočtu.
  - Logika rozhodovacího stromu.
  - Tabulka ekvivalentních značek (tabulka odkazů).
  - Správa paměti – fronta ukazatelů na volnou paměť.
- Výstupní fronta pro data nalezených objektů.

- Příjem nastavení po I^2C sběrnici, výstup dat pro PC aplikaci (obraz nebo hrany).
- Simulace a testování – Napsán kód generující testovací obrazová data včetně synchronizačních signálů a konfiguračního slova na I^2C.

5.1.4 Návrh a úpravy algoritmu labelingu

Návrh algoritmu probíhal v úzké návaznosti na jeho implementaci a realizovatelnost logikou v FPGA, tím byl postup složitější a časově náročnější. Během vývoje algoritmu byly provedeny následující kroky:

- Zhodnocení stávajících metod
- Úprava algoritmu pro nasazení v hradlových polích
- Úprava a redukce rozhodovacího stromu
- Minimalizace paměťových nároků
  - Nahrazení tabulky ekvivalentních značek
  - Vytváření a procházení orientovaných cest
- Recyklace uvolněné paměti
- Simulace a testování v prostředí MATLAB, vytvoření skriptu napodobujícího zpracování obrazu hradlovým polem.

5.1.5 Aplikace pro Windows

Pro zobrazení snímaného obrazu a nastavení parametrů zařízení byla v jazyce C# napsána aplikace pro operační systém Windows. Vývoj probíhal v prostředí MS Visual Studio 2010 nad rozhraním .NET Framework 4. Aplikace provádí či umožňuje následující operace:

- Otevření přístupu k EZ-USB zařízení a jeho endpointům.
- Komunikace textovými zprávami s USB periferií.
- Příjem, zpracování a zobrazení obrazových dat. Výběr oblasti a oříznutí zobrazeného snímku.
- Zobrazení jasových extrémů v obraze.
- Zasílání zpráv pro nastavení obrazového snímače nebo hradlového pole.
- Uživatelské rozhraní napomáhající při konstrukci textových zpráv.

5.1.6 Ostatní práce spojené s realizací

Pro úplnost uvedeme i úkony, které nejsou přímou součástí cílů, ale jsou pro realizaci nutné nebo ji výrazněji usnadnily.

- Propojení modulů plochými kabely.
- Výroba přípravku pro upevnění modulu obrazového snímače
- Úprava stativu zvětšovacího přístroje, montáž modulů na stativ.
- Přisvícení snímané scény LED osvětlením.
- Ladící výpisy z EZ-USB periferie do PC aplikace.
- Testování souborového systému a bootování pomocí mikropočítače STM32F407.

6. Zhodnocení dosažených výsledků

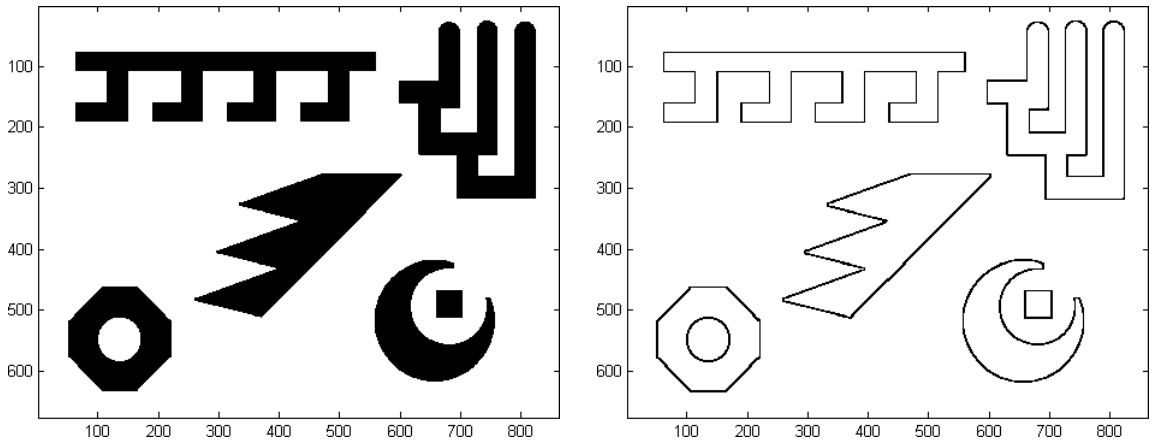
6.1.1 Zpracování obrazových dat

Využitím vyvinutého algoritmu je možno zpracovávat obrazová data a rozpoznávat objekty v obraze. Metody určování objektů vycházející z metod labelingu se podařilo na základě získaných zkušeností přepracovat do podoby realizovatelné logikou hradlových polí, s důrazem na minimální nároky.

Bylo dosaženo zpracování s vysokou mírou paralelismu, obvody příjmu a předzpracování obrazových dat vyhodnotí jeden obrazový bod v každém taktu hodinového signálu. Nalezení objektů a operace s nimi se daří zpracovávat rychlostí jednoho obrazového bodu již od tří taktů hodinového signálu. Tabulku ekvivalentních značek se podařilo redukovat na jediný sloupec, spolu s vyvinutou metodou uvolňování značek a jejich recyklace jsou paměťové nároky jádra algoritmu redukovány až na 18 bytů paměti pro každý použitý label včetně dat objektu a jeden byte paměti pro každý sloupec obrazu. Výsledkem zpracování obrazu jsou data reprezentující nalezené objekty až do počtu 127 objektů.

6.1.1.i MATLAB

Metody zpracování obrazu byly testovány s pomocí programu Matlab. Nad testovacími vzory byl spouštěn skript, který napodoboval implementaci algoritmu a zobrazoval průběžné výsledky prováděných kroků. Za testovací vzory byly voleny fotografie skutečných předmětů nebo tvary uměle vykonstruované tak, aby plně otestovaly schopnosti algoritmu. Pro výběr takových tvarů budeme diskutovat dosažený výsledek simulace.

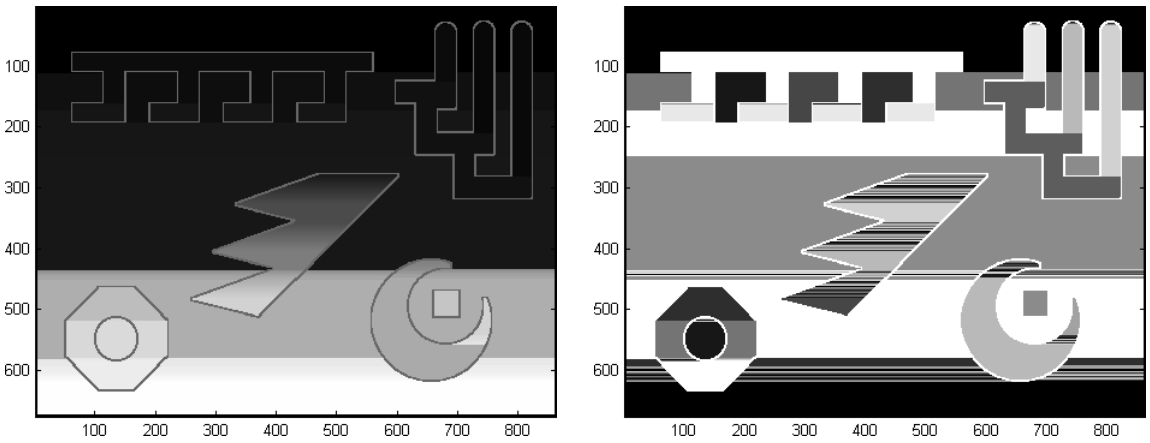


a) Vstupní obraz b) Nalezené hrany

Fig. 6.1 Simulace zpracování – vstupní data

Trochu předběhneme a prozradíme, že na zpracování tohoto snímku je zapotřebí celkem 243 labelů, ale najednou je jich obsazených nejvýše 12 (cca 5%). Zde budeme demonstrovat přínos recyklace uvolněných labelů. V ilustraci fig. 6.2-a je vykresleno přidělení labelů bez nutnosti recyklace, ve snímku je patrná monotónnost od shora

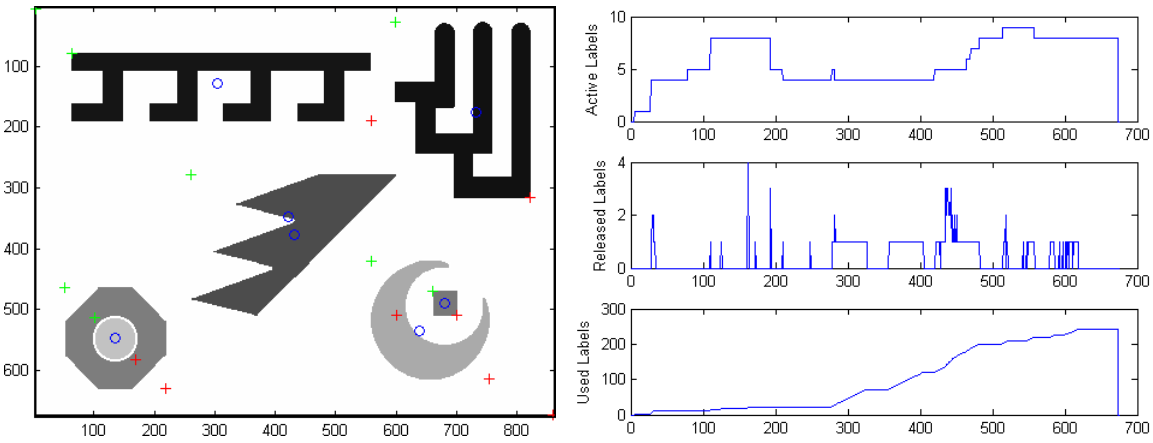
směrem dolů, žádný label není použit vícekrát. Pro druhý případ bylo dovoleno použít pouze nejnižší možný počet 12 labelů, rozdíl je více než jasný.



a) Využito 243 labelů b) Využito 12 labelů a jejich recyklace

Fig. 6.2 Simulace zpracování – ohodnocení obrazových bodů

Výsledky a parametry nalezených objektů se s rozdílným počtem použitých labelů nemění. Uveďme si nalezené objekty a grafické znázornění operací s nimi během zpracování snímku.



a) Nalezené objekty b) Využití labelů

Fig. 6.3 Simulace zpracování – nalezené objekty, využití paměti

Fig. 6.3-a zobrazuje nalezené objekty, jejich těžiště (pozice kroužků) a jejich obrysy (levý horní a pravý dolní roh vyznačený křížky). V grafech je znázorněný počet aktivních labelů na konci každého řádku, počet sloučených labelů během každého řádku a počet celkem přidělených labelů ke konci každého řádku.

Schopnost navrženého algoritmu zpracovávat obrazová data byla prokázána v plném rozsahu. Bylo dokázáno, že s použitím recyklace není zpracování omezeno celkovým počtem použitých labelů, ale nejvyšším počtem souběžně aktivních labelů.

6.1.1.ii Realizace hradlovým polem

Při návrhu aplikace hradlového pole nastal závažný problém během mapování navržených obvodů do logických bloků. I při zkrácení podporované délky řádku na 2048 obrazových bodů překonal rozsah návrhu možnosti hradlového pole Xc3s200 a nebylo

možno jej implementovat v plném rozsahu. Pro omezení nároků byl z výstupních dat vyjmut údaj o jasu plochy nalezeného objektu, čímž došlo k uvolnění blokové i distribuované paměti a snížení nároků na logické bloky. Narážíme i na vlastnost blokových RAM, ke kterým generátor IP jader nedokáže vytvořit přístup širší než 32 bitů na jeden blok. Využití paměti pro data širší než 32 bitů tak může být velmi neefektivní.

```
Návrh v plném rozsahu:
   Number of SLICEMs:          1,184 out of     960   123% (OVERMAPPED)
   (SLICEMs can only be placed in SLICEM sites.)
   Number of 4 input LUTs:     4,394 out of   3,840   114% (OVERMAPPED)
   Number of RAMB16s:             12 out of      12   100%

Návrh s omezením
   Number of occupied Slices: 1,905 out of   1,920    99%
   Number of 4 input LUTs:    3,372 out of   3,840    87%
   Number of RAMB16s:            11 out of      12    91%
```

Fig. 6.4 Využití prostředků Xc3s200 – ISE report

Redukovaný návrh byl syntetizován, přeložen, mapován a úspěšně spuštěn. Zpracování obrazových dat se ale podařilo prokázat pouze v simulacích (popsáno níže), ve skutečné aplikaci nejsou z dosud nezjištěných příčin vysílána výstupní data. Vedením vybraných signálů na výstupní piny a jejich měřením se podařilo ověřit činnost stavového automatu a resetovací logiky, příčinu problémů se bohužel vystopovat nepodařilo. Podezření se v této situaci ubírá směrem k činnosti generovaných IP jader, se kterými se potýkaly drobné nesrovnalosti již během návrhu.

6.1.1.ii -a Isim

Návrh logických obvodů byl laděn a testován v programu ISim, který je součástí vývojových nástrojů ISE WebPack. Simulace jsou prováděny s totožným VHDL kódem, ze kterého je generován konfigurační bitstream, tím by průběh simulace měl věrně napodobovat průběh signálů skutečné implementace. Průběh simulace byl pro ilustraci zachycen v rámci jednoho snímku.

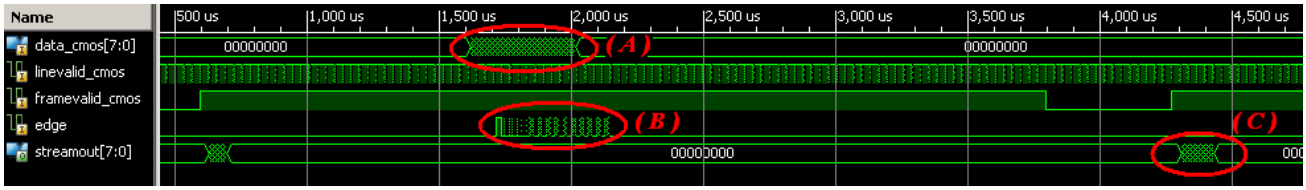


Fig. 6.5 Simulace VHDL kódu – klíčové signály

Průběhy v fig. 6.5 ukazují od shora dolů obrazová data a signály z CMOS snímače, vyhodnocené hrany a výstupní data. Do signálů jsou označeny tři význačné body. První, označený písmenem ***A*** je výsek obrazových dat obsahující testovací vzor, druhý je značen písmenem ***B*** a zvýrazňuje nalezené hrany k objektu ***A***. Posledním bodem (oblast ***C***) jsou výstupní data, která jsou přiblížena v fig. 6.6.

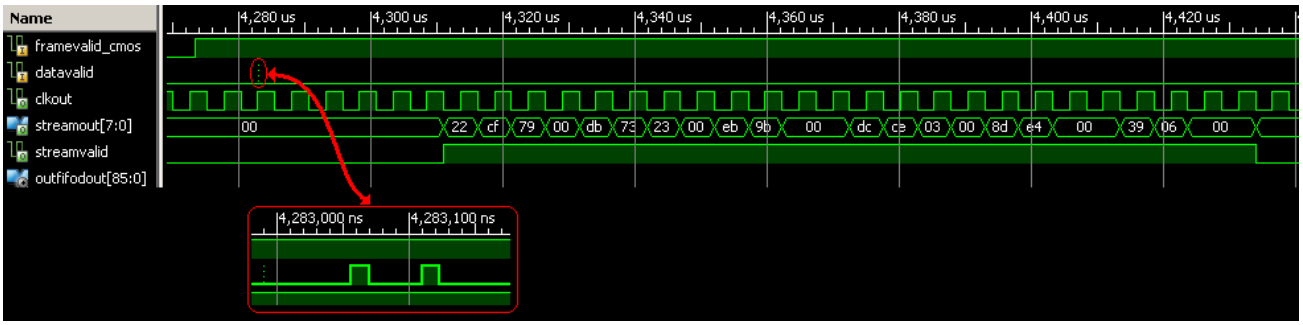


Fig. 6.6 Simulace VHDL kódu – výstupní signály

Během impulzů ve zvýrazněné oblasti jsou zapsána data do výstupní fronty, tato data jsou poté převedena na posloupnost bytů a vyvedena na výstupní piny. V rámci provedených simulací se všechny části návrhu chovají korektně, funkčnost navržených principů byla prokázána v plném rozsahu návrhu.

6.1.1.iii Zpracování dat – Shrnutí

Algoritmus labelingu byl úspěšně přepracován do podoby implementovatelné hradlovým polem, jeho principy byly ověřeny a podloženy simulacemi. Funkce se ve skutečném hardwaru v plném rozsahu prokázat nepodařilo.

6.1.2 Nadřazený systém, mikropočítač Cypress

Firmware mikropočítače konfiguruje EZ-USB periferii a zajišťuje funkce pro autonomní běh modulu. Vyvinuté rutiny pro konfiguraci hradlového pole a obrazového snímače ze souborů externí paměti byly úspěšně testovány. Přenos obrazových dat do nadřazeného PC byl ověřen spolu s funkcemi vyvinuté aplikace, nejvyšší dosažená rychlost přenosu přesáhla 42 MB/s. Správnost fungování byla ověřena v plném rozsahu implementovaných funkcí. Prostředky využívané firmwarem jsou všechny dostupné i pro nejmenší 56 pinová pouzdra obvodu Cy7c68013A. Jedinou nevýhodou použití procesoru 8051 jako řídicího je jeho rychlost, nahrání programovacího souboru do hradlového pole trvá okolo 2 minut.

6.1.3 Realizace modulu

Všechny použité komponenty byly propojeny a umístěny na stativ zvětšovacího přístroje. Provedené úpravy stativu a jeho použití se ukázalo velice přínosné pro komfort při obsluze a testování zařízení.

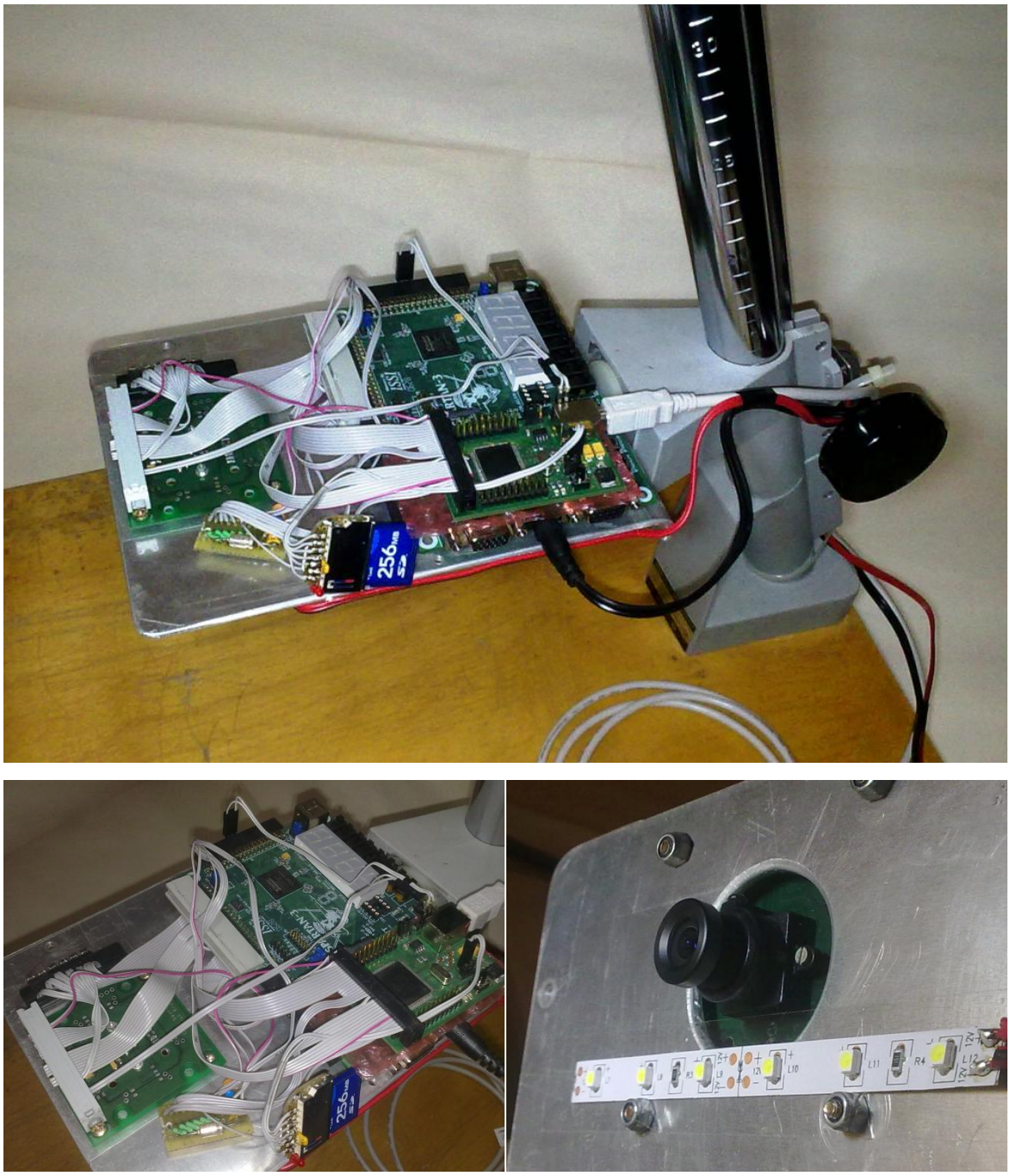


Fig. 6.7 Realizovaný obrazový modul

7. Závěr

Tato práce se zabývala vývojem metody pro zpracování obrazu z plošného obrazového CMOS snímače Aptina MT9V032 s využitím prostředků hradlového pole a její následné implementaci do hradlového pole Xilinx Spartan3 Xc3s200. S využitím této metody jsou v obraze rozlišeny objekty od pozadí scény a převedeny na soubor parametrů reprezentujících jejich vlastnosti a polohu. Dále byl řešen přenos obrazových dat z hradlového pole do nadřazeného PC s využitím EZ-USB periferie v obvodu Cypress Cy7c68013A a využití jeho jádra 8051 jako řídicího mikropočítače.

V textu práce byl popsán vývoj algoritmu zakládající se na znalostech získaných studiem již existujících algoritmů labelingu a zkušenostech získaných během práce s hradlovým polem. Část textu byla věnována firmwaru a činnostem mikropočítače Cypress, přenosu obrazových dat a jejich zobrazení ve vytvořené PC aplikaci. Také bylo uvedeno i znalostní minimum pro práci s obrazovými snímači.

Při realizaci obrazového modulu bylo vytvořeno programové vybavení pro FPGA zpracovávající obrazová data a firmware mikropočítače pro jejich přenášení skrze Hi - Speed USB periferii. Nejvyšší dosažená rychlost přenosu přesáhla 42MB/s, rychlost využitá při provozu je řádově čtvrtinová. Vytvořený firmware také při startu modulu konfiguruje hradlové pole a obrazový snímač. Pro autonomní start modulu byl do firmwaru mikropočítače implementován souborový systém a přístup k paměťové kartě formátu Secure Digital. Soubory uložené v paměťové kartě obsahují všechna konfigurační data potřebná ke startu zařízení.

Pro nadřazený PC byla v jazyce C# vytvořena aplikace, která komunikuje s EZ-USB periferií, přijímá obrazová data a umožňuje nastavovat parametry modulu. Obrazová data jsou aplikací zpracována a vykreslena do hlavního okna aplikace.

Ze stanovených cílů se podařilo splnit realizaci modulu s připojením CMOS obrazového snímače, jeho autonomní start s využitím SD paměťových karet se souborovým systémem FAT a spolupráci s nadřazeným PC. Navržená metoda zpracování obrazu nezpomaluje jeho snímání, pro bezpečnou funkci ale vyžaduje kmitočet šestkrát vyšší než je kmitočet čtení obrazových dat. Zpracování obrazu je nezávislé na rozměrech snímaného obrazu, výstupem mohou být parametry až 127 objektů.

V této práci bylo usilováno o využití malých a levných hradlových polí, rozsah návrhu ale převyšuje jejich možnosti a nepodařilo se ho implementovat v plném rozsahu.

8. Seznam použité literatury

[1] *Spartan-3 Generation FPGA User Guide, UG331*. [online].
Inc. Xilinx, Ver 1.8, June 13, 2011.
http://www.xilinx.com/support/documentation/user_guides/ug331.pdf

[2] KOVÁCS, Petr. Videoprocesor s FPGA. Praha: ČVUT 2009.
Diplomová práce, ČVUT, Fakulta Elektrotechnická, Katedra měření.

[3] FISCHER, J. *Optoelektronické senzory a videometrie.*
Skripta, Praha, ČVUT, Fakulta Elektrotechnická.

[4] NÁDVORNÍK, Vojtěch. Mnohořádkový videosenzor. Praha: ČVUT 2007.
Diplomová práce, ČVUT, Fakulta Elektrotechnická, Katedra měření.

[5] SOUČEK, Pavel. *Použití rozhraní USB2.0 pro rychlý přenos obrazu.* Praha: ČVUT 2007. Bakalářská práce, ČVUT, Fakulta Elektrotechnická, Katedra měření.

[6] ZIMA, Jiří. *Rychlý přenos obrazu a dat s využitím EZ-USB.* Praha: ČVUT 2009.
Bakalářská práce, ČVUT, Fakulta Elektrotechnická, Katedra měření.

[7] *MT9V032 Data Sheet: 1/3-inch Wide-VGA Digital Image Sensor.* [online].
Aptina Imaging Corporation. Rev.D, 5/9/2011.
http://www.aptina.com/assets/downloadDocument.do?id=844

[8] *I^2C-bus specification and user manual, UM10204*.[online].
NXP Semiconductors. Rev. 6, April 4, 2014.
http://www.nxp.com/documents/user_manual/UM10204.pdf

[9] *Spartan-3 Generation Configuration User Guide, UG332.* [online].
Inc. Xilinx, Ver 1.6, October 26, 2009.
http://www.xilinx.com/support/documentation/user_guides/ug332.pdf

[10] *EZ-USB® FX2LP™ USBMicrocontroller, High-Speed USB Peripheral Controller. CY7C68013A, CY7C68014A, CY7C68015A, CY7C68016A Data Sheet.* [online].
Cypress Semiconductor Corporation. Rev. *W, July 19, 2013. Doc. No. 38-08032
http://www.cypress.com/?docID=45142

[11] *EZ-USB® Technical Reference Manual.* [online].
Cypress Semiconductor Corporation. Rev. *E, April 24, 2014. Doc. # 001-13670
http://www.cypress.com/?docID=48811

[12] *SD Specifications part1: Physical Layer Simplified Specifications.* [online].
SD Group. Ver 4.10, January 22, 2013.
https://www.sdcard.org/downloads/pls/simplified_specs/part1_410.pdf

[13] *SanDisk Secure Digital Card: Product Manual.* [online].
SanDisk Corporation. Ver 1.9, December 2003. Doc. No. 80-13-00169
http://alumni.cs.ucr.edu/~amitra/sdcard/ProdManualSDCardv1.9.pdf

[14] The Electronic Lives Manufacturing – ChaN. *Petit FAT File System Module* [firmware]. June 10, 2014. R0.03. [online]. http://elm-chan.org/fsw/ff/00index_p.html

[15] *Using Block RAM in Spartan-3 Generation FPGAs, App.Note XAPP463.* [online]. Inc. Xilinx, Ver 2.0, March 1, 2005. http://www.eng.utah.edu/~cs3710/xilinx-docs/xapp463.pdf

[16] *Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs, App.Note XAPP462.* [online]. Inc. Xilinx, Ver 1.1, January 5, 2006. http://www.xilinx.com/support/documentation/application_notes/xapp462.pdf

[17] *LogiCORE IP FIFO Generator v9.3, Product Guide PG057.* [online]. Inc. Xilinx, Rev 3.0, December 18, 2012. http://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v9_3/pg057-fifo-generator.pdf

[18] TOMS, Daniel. *Zpracování obrazu pro sledování optické stopy.* Praha: ČVUT 2014. Bakalářská práce, ČVUT, Fakulta Elektrotechnická, Katedra měření.

[19] GREPL, Lukáš. *Modul USB cy7c68013a.* [schéma][CD]. Praha: ČVUT, Fakulta Elektrotechnická, Katedra měření.

[20] Costantino GRANA, Daniele BORGHESANI and Rita CUCCHIARA. *Optimized Block-Based Connected Components Labeling With Decision Trees*. IEEE Transactions on image processing [online]. June 2010, 19(6), ISSN 1057-7149. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5428863

[21] Rama Sai Krishna Vakkantula: *Designing With EZ-USB® FX2LP™ Slave FIFO Interface, Application Note AN61345*. [online]. Cypress Semiconductor Corporation, Rev.*J, 03/19/2014, Doc. No. 001-61345. http://www.cypress.com/?docID=49468

[22] Prajith Cheerakkoda: *Configuring x Xilinx Spartan-3E FPGA Over USB Using EZ-USB FX2LP™, Application Note AN63620.* [online]. Cypress Semiconductor Corporation, Rev.*B, Sept. 10, 2013, Doc. No. 001-63620. http://www.cypress.com/?docID=45758

[23] Gayathri Vasudevan: *Designing a Bulk Transfer Host Application for EZ-USB® FX2LP™ / FX3™, Application Note AN70983.* [online]. Cypress Semiconductor Corporation, Rev.*C, 12/12/2013, Doc. No. 001-70983. http://www.cypress.com/?docID=47153

[24] Rama Sai Krishna Vakkantula: *Streaming Data Through Isochronous or Bulk Endpoints on EZ-USB® FX2™ and FX2LP™, Application Note AN4053.* [online]. Cypress Semiconductor Corporation, Rev.*G, 08/28/2014, Doc. No. 001-15289. http://www.cypress.com/?docID=50815