České vysoké učení technické v Praze
Fakulta elektrotechnická

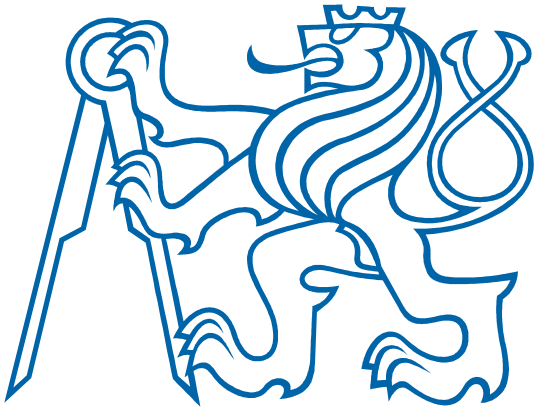
DIPLOMOVÁ PRÁCE

2009 Petr Kovács

České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra měření



Diplomová práce

Videopreprocesor s FPGA

Petr Kovács

Vedoucí práce: Ing. Jan Fischer, Csc.

Leden 2009

Prohlášení

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

Nemám závažný důvod proti použití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a změně některých zákonů (autorský zákon).

V Praze dne . . . . . .

. . . . . .

Petr Kovács

Poděkování

Za odborné vedení diplomové práce, konzultace a cenné rady děkuji Ing. Janu Fischerovi, CSc.

Stejné poděkování patří rodině a přátelům, za podporu při psaní a tvorbě diplomové práce.

Abstrakt

Cílem této práce je navrhnout a realizovat malou kompaktní kameru pro bezkontaktní měření rozměrů objektů v reálné čase. Kamera je na začátku práce nazvána VIDEOSENZOR. První kapitoly popisují návrh modulu zpracování obrazového signálu. Modul je hlavní částí kamery. Srdcem modulu je hradlové pole FPGA. V práci je vyřešen způsob připojení periferních obvodů k FPGA a způsob jeho konfigurace. Další kapitoly popisují návrh videopreprocesoru s FPGA pomocí jazyka VHDL. V práci je navrženo rozhraní videopreprocesoru pro komunikaci s nadřazeným procesorem a metody pracující v reálném čase potřebné pro bezkontaktní měření.

Abstract

The aim of this thesis is to design and construct a small compact camera for real time, contactless measuring of the size of an object. The camera will be from the beginning of this thesis called VIDEOSENZOR. The first chapters describe the design of the image processing module; this module is the main part of the camera. The heart of the module is the FPGA gate array. In the thesis, the method of connecting the peripheral circuit with the FPGA and the method of its configuration is resolved. The other chapters describe the design of an FPGA based videopreprocesor using the VHDL language. The design of the interface of the videopreprocesor for communication with a higher level processor and real time methods used for contactless measurments are resolved in the thesis.

OBSAH

1. Úvod

Pojem videometrie se objevuje v souvislosti s bezkontaktním měřením především rozměrů objektů. FISCHER(1) popisuje videometrii takto:

… v řadě případů se nejedná jen o úkoly rozpoznávání, ale přímo o měření vyhodnocením obrazu. Právě tuto situaci vystihuje pojem videometrie, to je měření viděním. Zatímco v klasickém počítačovém vidění nebyla problematika chyb měření příliš uvažována, ve videometrickém systému jde i o věrohodnost výsledku měření z hlediska metrologického.

Tedy videometrie je měření viděním a prostředek pro měření viděním je videosenzor. Jedná se o malou kompaktní kameru, která v reálném čase měří geometrické parametry objektů na základě plošného obrazu pořízeného pomocí CCD nebo CMOS senzoru.

Videosenzor je z hlediska možností měření objektů výkonnější než jen pouhá optická závora a přitom si zachovává kompaktnost na rozdíl od systémů používajících počítač. Používá se jako prostředek k dosažení stoprocentní kontroly v automatizované průmyslové výrobě.

Návrhem videometrického systému se zabývalo již několik projektů či diplomových prací. Základním problémem, se kterým se musejí tyto systémy vypořádat, je vysoký datový tok digitálního obrazového signálu, který je 10MB/s. Jak ve své práci dokazuje ŘÍHA(2), pro zpracování tohoto signálu se výborně hodí hradlové pole FPGA zejména díky možnosti běhu paralelních procesů, pomocí kterých je možno do FPGA implementovat rychlé metody zpracování obrazového signálu.

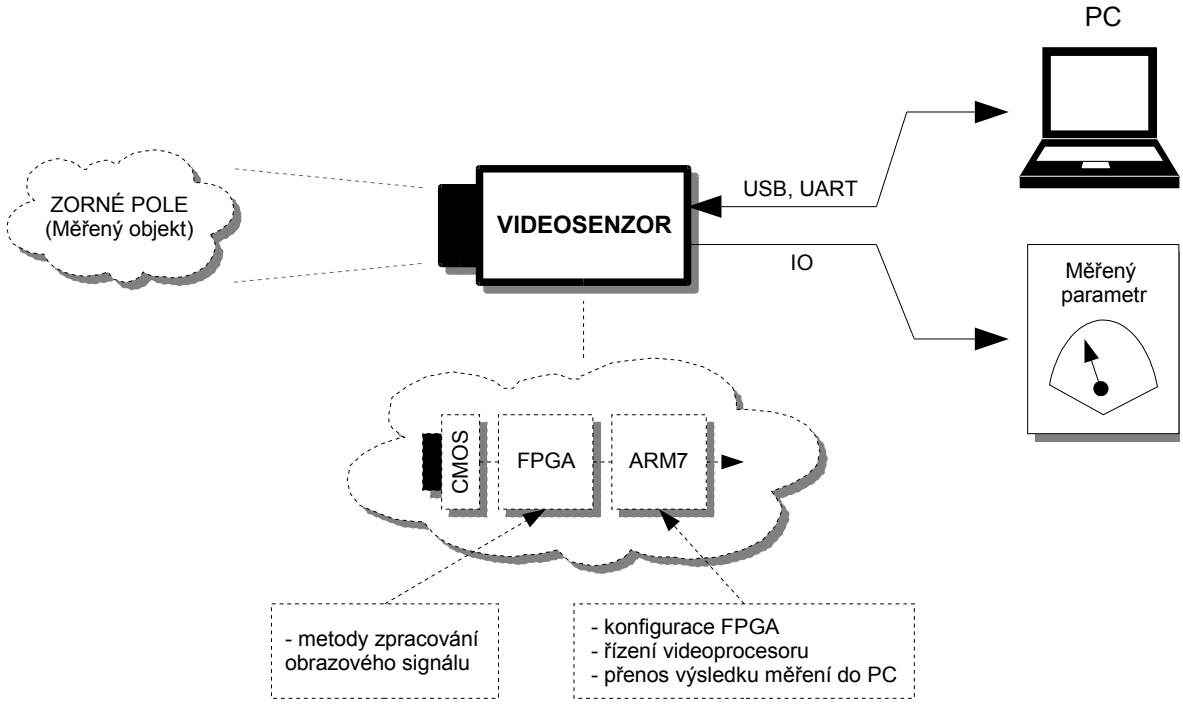
Videopreprocesor naprogramovaný v obvodu FPGA, který metody zpracování obrazového signálu pracující v reálném čase také implementuje, je základem pro tuto práci. Zde bude přínosem zejména návrh periferií videopreprocesoru, vhodných pro komunikaci s nadřazeným procesorem ARM7. Do videopreprocesoru budou implementovány metody zpracování obrazového signálu potřebné pro vyhledávání okrajů objektů, vyhledáván optických hran ve směru řádků, určování velikosti plochy, středu objektů a určování příčného a vertikálního rozměru.

Obvod FPGA není po zapnutí napájení schopen ihned samostatně fungovat a musí se nejprve zajistit jeho konfigurace do stavu, ve kterém se chová jako videopreprocesor. V této práci bude kompletně dořešen způsob konfigurace FPGA pomocí nadřazeného procesoru ARM7 a paměti FLASH. Navíc zde bude realizováno zapojení, které umožní uložení několika konfigurací FPGA

do paměti FLASH. To znamená, že videosenzor bude možno dle volby uživatele přeprogramovat na různý typ videopreprocesoru, každý s jinými metodami zpracování obrazového signálu.

Tato práce se také zabývá návrhem modulu zpracování obrazového signálu, který celý systém implementuje na jedné desce plošného spoje. Srdcem modulu bude zmíněný obvod FPGA a procesor ARM7. Modul bude také osazen USB řadičem CYPRESS FX2LP nebo též EZ-USB, který se bude používat pro přenos plynulého obrazu sledované scény do počítače za účelem nastavení a zaostření videosenzoru. Update firmware celého systému bude pomocí rozhraní UART.

Spojením modulu zpracování obrazového signálu s modulem obrazového senzoru (semestrální projekt, Jan Šedivý, 2003) vznikne kamera pro bezkontaktní měření, která bude nazvána **VIDEOSENZOR** a to je vize této práce.



Obr 1.1 Videosenzor – kamera pro bezkontaktní měření

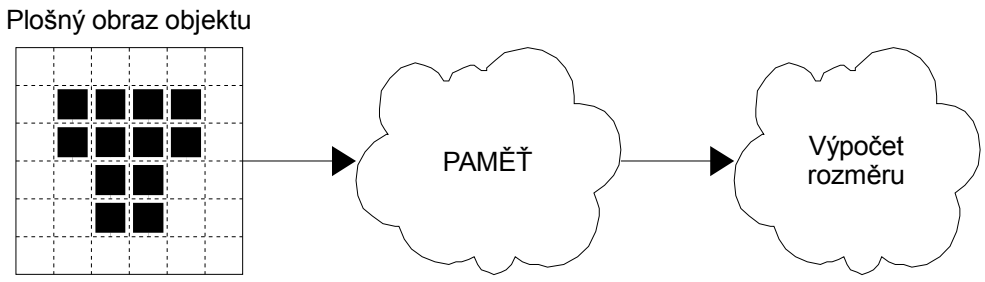
2. Rozbor problematiky

V této kapitole je popsán princip zpracování obrazového signálu videosenzorem. Jsou zde ukázány dva základní postupy, které řeší problém zpracování obrazového signálu s vysokým datovým tokem 10MB/s. Výsledkem je použití obvodu FPGA pro zpracování obrazu v reálném čase.

2.1 Postup při zpracování obrazu

Zpravidla se dají rozlišit dva základní typy zpracování obrazu a to za prvé v reálném čase a za druhé zpracování obrazu uloženého v paměti.

Zpracování obrazu uloženého v paměti je schématicky naznačeno na Obr 2.1. Plošný obraz měřeného objektu je nejprve celý načten do paměti odkud je postupně vyčítán a algoritmem zpracování obrazu je vypočítán rozměr objektu (například horizontální rozměr).



Obr 2.1 Zpracování obrazu uloženého v paměti

Takové zpracování obrazu může být realizováno pomocí procesoru a paměti schopné načítat data s frekvencí odpovídající datovému toku obrazového signálu tedy 10MB/s. Množství dat před a za pamětí je stejné a výsledek měření rozměru je zpožděn minimálně o dobu, po kterou se plošný obraz vyčítal z paměti. Rychlost vyčítání je závislá na rychlosti algoritmu pro výpočet rozměru. Vlivem tohoto zpoždění může docházet ke ztrátě dalších snímků.

Zpracování obrazu v reálném čase je naznačeno na Obr 2.2. Postup zpracování obrazu je rozdělen do dvou částí. V první části je vysoký datový tok snížen tím, že se z plošného obrazu

objektu naleznou pouze jeho užitečné informace, které objekt charakterizují a to jsou hrany. Hrany objektu představují základní informaci jeho existenci a o jeho aktuálním stavu v pozorované scéně. Z hran může být pak rovnou vypočten rozměr objektu.

Obr 2.2 Zpracování obrazu v reálném čase

To, že se datový tok nalezením hran objektu podstatně snížil, dokazuje tento příklad:

K pořízení obrazu je použit obrazový senzor LM9617, který má rozlišení 648 krát 488 a produkuje 30 snímků za sekundu. Vynásobením těchto čísel dostaneme datový tok přibližně 10MB/s (Mega Byte / sec.). Tento datový tok je na Obr 2.2 před první bublinou. Před druhou bublinou je datový tok, který odpovídá datovému toku hran. Pokud se měří rozměr musí být známy minimálně dvě hrany objektu. Pro tento případ nalezení dvou hran objektu je datový tok 2 krát 30 tedy 60 hran/s. Hrana sice nebude vyjádřena jako jeden Byte, tedy 8 bitů, ale jako 10 bitů (protože rozlišení řádku je 648 a toto číslo se získá pouze 10tibitovým kódováním), přesto je vidět, že datový tok se zmenšil asi 158 000 krát. Jedná se samozřejmě o nejjednodušší příklad, kdy jsou nalezeny pouze dvě hrany, přesto, ale bude výsledný datový tok hran vždy podstatně menší než datový tok z CMOS plošného senzoru.

Z této podkapitoly vyplývá důležitý poznatek. Algoritmus výpočtu rozměru při zpracování obrazu v reálném čase může být pomalejší než algoritmus výpočtu rozměru při zpracování obrazu uloženého v paměti. Vyplývá to z uvedeného příkladu. Pokud by byly algoritmy stejně rychlé tak v druhém případě dochází k menší ztrátě načítaných snímků než v prvním případě nebo nedochází ke ztrátě načítaných snímků vůbec a i přesto může být algoritmus pomalejší. Klíčové je však najít způsob nalezení hran bez uložení plošného obrazu do paměti.

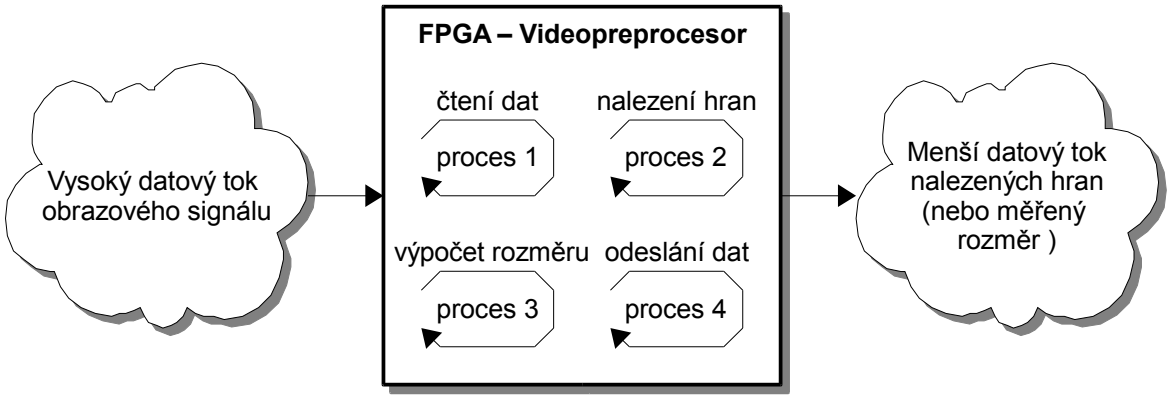
2.2 FPGA pro zpracování obrazového signálu

Pro nalezení hran objektu v reálném čase z plošného obrazu se výborně hodí programovatelné součástky jakými jsou například obvody CPLD (complex programmable logic device) nebo FPGA (field programmable gate array).

NÁDVORNÍK(5) ukazuje ve své diplomové práci způsob realizace hranového detektoru v obvodu CPLD typu XC9572XL. Zde se setkává s problémem, kdy do cílového obvodu již nebylo možné implementovat čítač pozice hrany objektu. Jednoduše v tomto obvodu nezbyl dostatek prostředků pro implementaci čítače. Tento problém však řeší použitím čítače a signálu přerušení v nadřazeném obvodu, a zcela jistě tak dokazuje, že obvod CPLD pro nalezení hran lze použít.

Větší výkon pro zpracování obrazového signálu přináší obvod FPGA. ŘÍHA(2) ve své diplomové práci dokazuje, že v tomto obvodu lze realizovat úplný hranový detektor ba dokonce lze do obvodu implementovat i další metody potřebné pro měření rozměrů objektu. Možnost implementace hranového detektoru i těchto metod pracujících v reálném čase ve své práci vysvětluje tím, že v FPGA (případně v CPLD) mohou být souběžné procesy.

Pojmem proces v hradlovém poli se rozumí logická struktura, která může být kombinační nebo sekvenční a která má nějakou funkci při zpracování digitálního signálu. Procesů v hradlovém poli může být několik, probíhají současně a mohou mezi sebou komunikovat prostřednictvím signálů. Zpracování obrazového signálu v reálném čase pomocí hradlového pole využívá procesů jak naznačuje Obr 2.3.



Obr 2.3 Procesy v hradlovém poli při zpracování obrazového signálu

2.3 Redukovaný obrazový signál

V dalším textu se bude používat pojem **redukovaný obrazový signál**. Pod tímto pojmem se bude rozumět signál na výstupu FPGA. Tedy signál, který nese pozici nalezených hran nebo například obsah objektu. Dle předchozích odstavců jde o signál s menším datovým tokem než je tok digitálního obrazového signálu, tedy je redukovaný (obsahuje podstatně menší množství dat).

2.4 Digitální obrazový signál

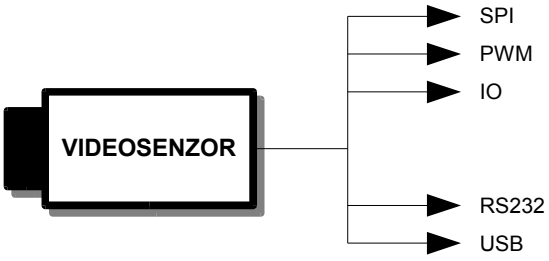
Pod pojmem digitální obrazový signál se bude rozumět signál z CMOS plošného senzoru, který nese úplnou informaci a sledované scéně (o obraze objektu). Průběh tohoto signálu bude uveden v kapitole *6.4 Vstup CMOS LM9617 – připojení plošného senzoru.*

2.5 Obvod FPGA, periférie a okolní obvody

Vizí této práce je vytvořit kompaktní systém pro bezkontaktní měření – videosenzor. Jak vyplývá z předchozí kapitoly klíčovým prvkem bude hradlové pole FPGA, pomocí kterého bude realizován videopreprocesor s metodami zpracování obrazového signálu. K FPGA budou ovšem muset být ještě připojeny další okolní obvody a to zejména ze dvou důvodů.

První důvod je, že obvod FPGA není schopen po zapnutí napájení ihned samostatně fungovat, protože nemá interní FLASH pro uložení své konfigurace. Existuje varianta použití speciální FLASH navržené pro FPGA, které si pomocí interního automatu načte samo konfiguraci. To by ovšem bylo příliš jednoúčelové (paměť nemůže být použita k ničemu jinému a udrží jen jednu konfiguraci) a navíc musí být použit konkrétní typ FLASH od výrobce FPGA. Pro konfiguraci bude lepší použít nadřazený procesor a externí paměť FLASH.

Druhý důvod je, že videosenzor musí mít periférie pro komunikaci s nadřazeným systémem. Periférie které se uvažují jsou dle Obr 2.4.



Obr 2.4 Periférie videosenzoru

Rozhraní USB by se využilo pro přenos obrazu do PC v reálném čase za účelem sledování měřeného objektu. Důvod jeho použití je zejména datový tok který postačí pro přenos obrazu a to, že se jedná o standartní rozhraní, které je v dnešní době osazeno na každém počítači.

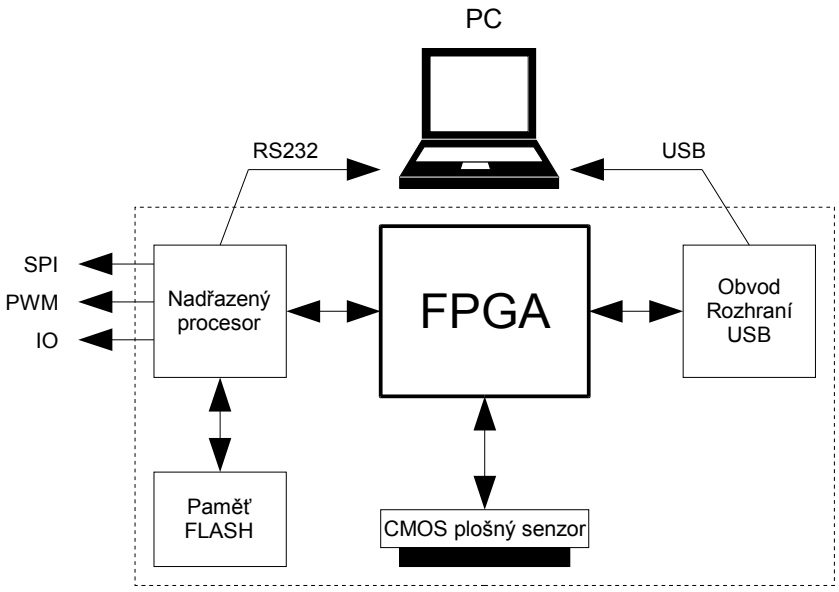
Rozhraní RS232 by se hodilo pro svou jednoduchost. Videosenzor by mohl s počítačem komunikovat pomocí obyčejného terminálu sériové linky. Pomocí tohoto rozhraní by mohly být nastavovány základní funkce videosenzoru a také by pomocí tohoto rozhraní mohl být prováděn

update firmware (naprogramování nadřazeného procesoru a následné nahrání konfiguračního souboru pro FPGA).

Rozhraní PWM a IO (universální IO piny) by se hodilo pro řízení mechanických částí v reálném čase. Například signál, který vypne výrobní linku pokud je videosenzorem změřen špatný rozměr výrobku atd.

Rozhraní SPI by se hodilo pro komunikaci s jiným nadřazeným systémem nebo procesorem.

Z prvního a druhého důvodu vyplývá připojení dalších obvodů dle obrázku Obr 2.5.



Obr 2.5 Okolní obvody pro obvod FPGA

2.6 Nadřazený procesor

Vhodným nadřazeným procesorem by mohl být některý ze zástupců rodiny procesorů ARM7. Tyto procesory disponují mnoha periferními bloky, obsahují interní FLASH atd. Výhodou je také jejich dobrá dostupnost na trhu.

Jako vhodný by mohl být **LPC2148**. Krom jiné obsahuje tyto bloky, které by se hodily pro konstrukci vizuálního senzoru:

2 x rozhraní SPI: Použilo by se jedno jako výstup z videosenzoru a jedno pro komunikaci s FLASH a FPGA.

7 x pulsně šířkový generátor PWM: Použil by se 2 x pro výstup z videosenzoru.

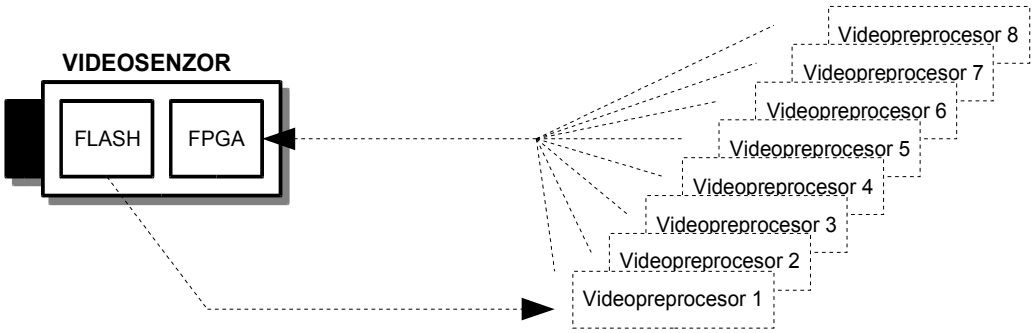
32 bitovou a 16 bitovou IO bránu: Použila by se jako výstupy IO z videopreprocesoru a také na komunikaci s FPGA.

2 x rozhraní UART: Použilo by jedno pro linku RS232.

2.7 Paměť FLASH

Vhodnou pamětí je typ **M25P80**. Jedná se o paměť s rozhraním SPI (serial peripheral interface). Její velikost je 1MB (přesně1048576 B). Paměť je organizována do 16 sektorů. Každý sektor obsahuje 256 stránek a každá stránka 256 Byte. Do paměti lze uložit až 8 konfiguračních souborů (každý zabere dva sektory, více o konfiguračním souboru je rozepsáno v kapitole *4.2 Konfigurační soubor*).

Pokud by bylo možné do paměti uložit až osm konfiguračních souborů byla by to výborná vlastnost videosenzoru. Pokud by totiž v hradlovém poli nezbyl dostatek místa pro realizaci jednoho videopreprocesoru mohlo by se pomocí nadřazeného procesoru nahrát do FPGA různý typ videopreprocesoru s různými metodami zpracování obrazového signálu dle požadavku uživatele. Každý videopreprocesor uložený v paměti by obsahoval jiné metody. Také to znamená, že může být použit levnější typ hradlového pole s menším počtem hradel a registrů. Myšlenka tohoto odstavce je naznačena na Obr 2.6.



Obr 2.6 Uložení konfigurace FPGA do paměti FLASH

2.8 Obvod rozhraní USB

Při volbě rozhraní USB se vycházelo z výsledků práce (6), kde bylo prokázáno a ověřeno na reálném zapojení, že pro přenos obrazu po USB lze použít obvod **CY7C68013A.** Jedná se o obvod rozhraní USB s vyrovnávací FIFO pamětí s řídícím procesorem řady 8051.

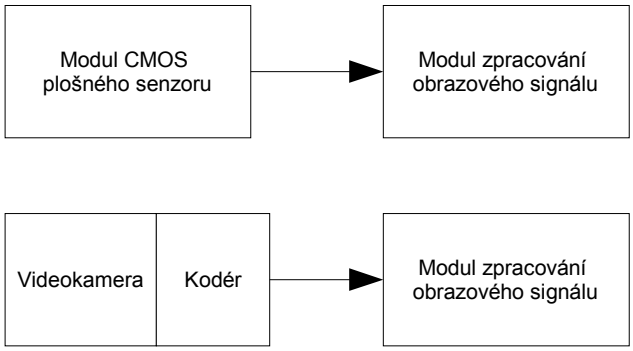
2.9 Hradlové pole FPGA

Vhodný typ hradlového pole je Spartan XC3S200 firmy XILINX viz. (4). Jedná se o FPGA za cenu přibližně 500 Kč. Mimo jiné obsahuje celkem 200 000 systémový hradel a také 288Kb

blokových RAM, které bude možno použít pro uložení vybrané části obrazu a pro realizaci výstupní FIFO paměti videopreprocesoru.

2.10 Moduly videosenzoru

Předpokládá se, že videosenzor bude složen z dvou modulů. Za prvé z modulu CMOS plošného senzoru s osazeným CMOS senzorem LM9617 a za druhé z modulu zpracování obrazového signálu. Důvod je zejména, aby celý systém nebyl omezen jen na jeden konkrétní typ obrazového senzoru. Použít půjdou různé typy CMOS plošných senzorů s digitálním výstupem nebo eventuálně videokamera a kodér s digitálním výstupem připojeným do stejného konektoru jako CMOS plošný senzor viz. Obr 2.7.



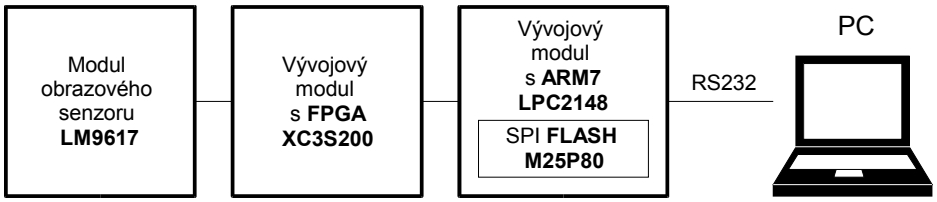
Obr 2.7 Moduly videosenzoru

V této práci se bude používat modul CMOS plošného senzoru typu LM9617, který byl navržen v rámci semestrálního projektu (Jan Šedivý, 2003).

3. Modul zpracování obrazového signálu

3.1 Testování pomocí vývojových modulů

Dříve než se pokračovalo v návrhu modulu zpracování obrazového signálu muselo se ověřit zda obvody uvedené v kapitole *2 Rozbor problematiky* budou spolu bez problémů komunikovat. Za tímto účelem se použil vývojový modul s procesorem ARM7 (s připojenou pamětí SPI FLASH pomocí kontaktů), vývojový modul s FPGA a modul CMOS plošného senzoru. Orientační propojení modulů je dle Obr 3.1



Obr 3.1 Propojení vývojových modulů

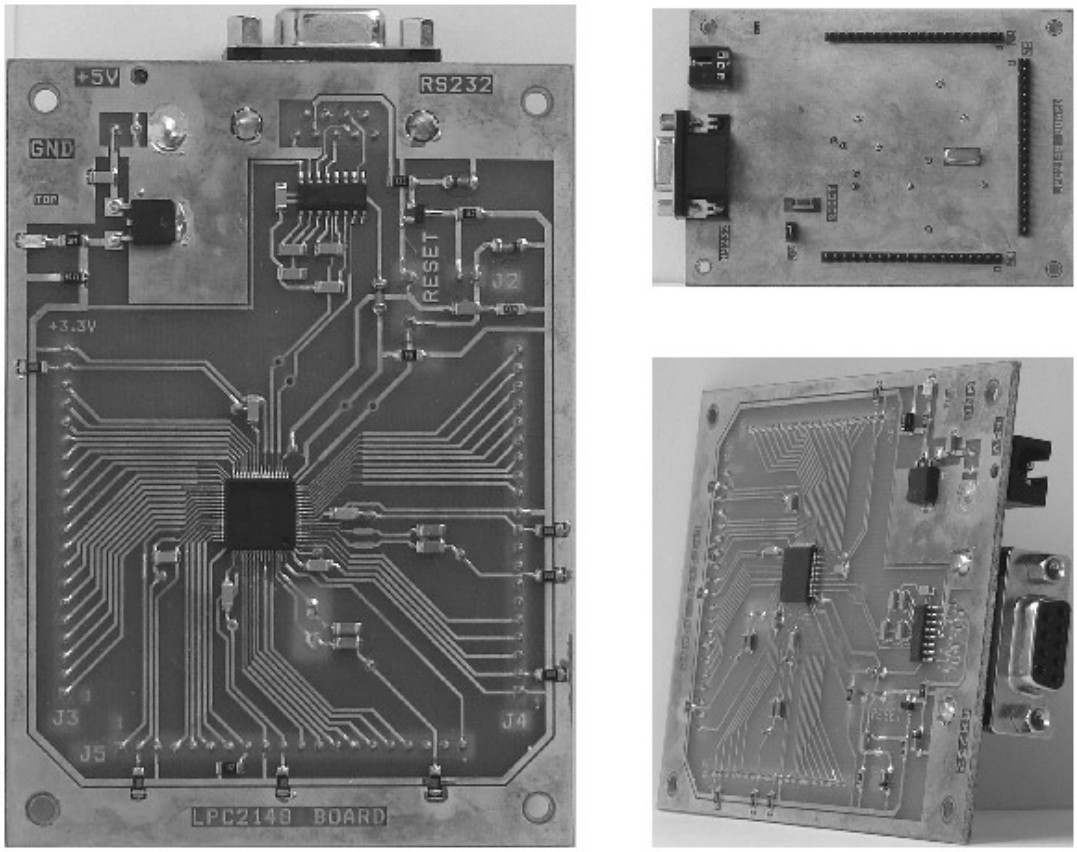
Na modulech se testovalo:

- přenos konfiguračního souboru pro FPGA po lince RS232 z PC do paměti FLASH
- konfigurace FPGA pomocí ARM7 z paměti FLASH po rozhraní SPI0
- přenos dat FPGA – ARM7 po rozhraní SPI0
- přenos dat FPGA – ARM7 paralelním rozhraním (16 pinů + příznakové signály)
- rozhraní pro čtení dat z obrazového senzoru do FPGA

Vývojový modul procesoru ARM7 byl navržen v rámci této diplomové práce, autorem této diplomové práce. Modul obsahuje tyto prvky:

- procesor ARM7 LPC2148
- stabilizátor napětí
- budiče linky RS232 s konektorem
- oscilátor 12MHz
- 3 IO konektory s vyvedenými IO piny procesoru + GND a VCC

Na obrázku Obr 3.2 je fotografie tohoto modulu. Spoj modulu je dvouvrstvý. Strana součástek obsahuje spoje a spodní strana obsahuje vrstvu GND. Modul byl vyroben levnou cestou bez prokovů, které byly zhotoveny dodatečně pomocí drátků.



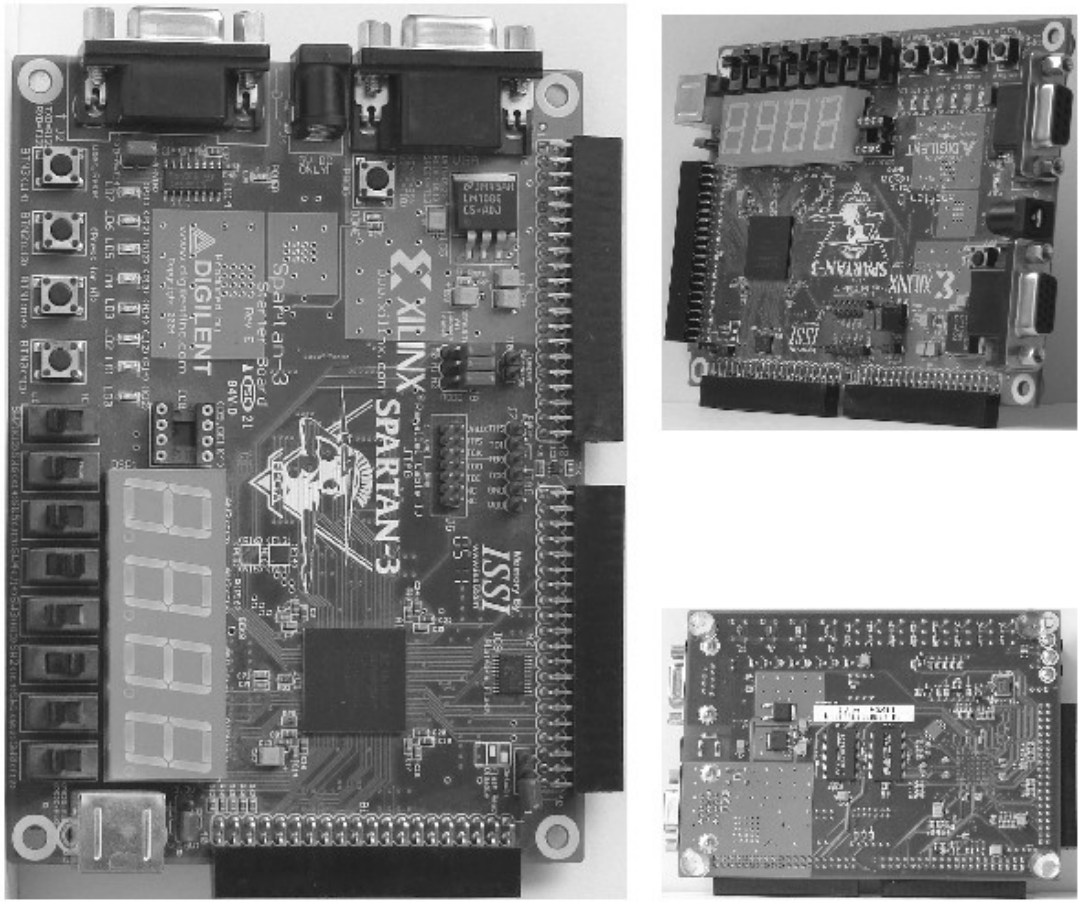
Obr 3.2 Vývojový modul s ARM7 LPC2148

Vývojový modul s FPGA byl od firmy DIGILENT. Modul obsahuje celou řadu prvků. Pro testování v rámci této práce byly použity tyto prvky:

- obvod FPGA
- IO konektory s vyvedenými piny
- uživatelská tlačítka

Fotografie tohoto modulu je na Obr 3.3.

Poslední modul, modul obrazového senzoru CMOS LM9617, byl zapůjčen z laboratoře videometrie. Jedná se o semestrální projekt (Jan Šedivý, 2003).



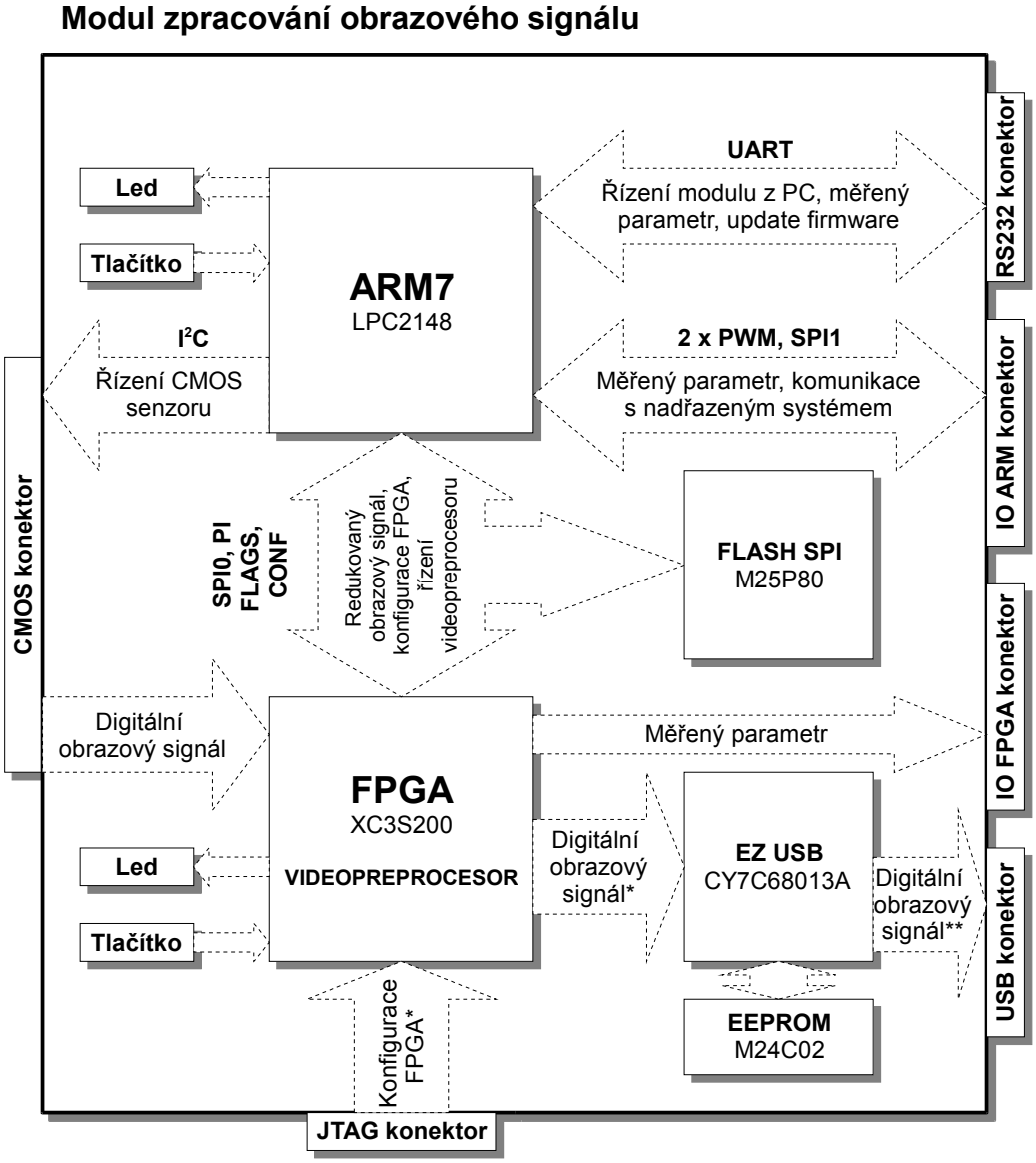
Obr 3.3 Vývojový modul s FPGA Spartan 3 XC3S200

3.2 Blokové schéma modulu

Dle předchozích úvah podle kapitoly *2 Rozbor problematiky* bylo navrženo blokové schéma modulu zpracování obrazového signálu dle Obr 3.4. Na modulu jsou vyznačeny:

- integrované obvody
- IO konektory
- komunikační rozhraní

Tok informace mezi jednotlivými obvody je vyznačen čárkovanými šipkami s příslušným popisem. Typ komunikačních rozhraní je připsán k čárkovaným šipkám. Přehledový popis významu obvodů, konektorů a komunikačních rozhraní je uveden dále.



Obr 3.4 Modul zpracování obrazového signálu – blokové schéma

INTEGROVANÉ OBVODY:

FPGA: Obvod naprogramovaný jako videopreprocesor. Srdce modulu. Zajišťuje zpracování digitálního obrazového signálu z CMOS plošného senzoru při měření rozměrů objektu.

EZ USB: Řadič sběrnice USB. Zajišťuje přenos obrazu sledované scény do počítače. Sběrnice označená hvězdičkou, kterou je tento obvod propojen s FPGA obsahuje jak datové tak synchronizační signály.

ARM7: Nadřazený procesor. Plní následující funkce: Konfigurace FPGA. Komunikace s počítačem při nastavení videosenzoru. Přenos konfiguračního souboru z PC do paměti FLASH.

FLASH SPI: Paměť s rozhraním SPI. Slouží především pro uložení konfigurace obvodu FPGA. Paměť je připojena na sběrnici SPI0 společnou pro FPGA i nadřazený procesor ARM7.

IO KONEKTORY:

JTAG konektor: Konfigurace (programování) FPGA. Použití při návrhu a testování funkcí videopreprocesoru.

IO FPGA konektor: Výstup z FPGA. Propojen je paralelní 16 bitovou sběrnicí. Jedná se o přímý výstup (například měřený rozměr objektu) nebo signály tohoto konektoru mohou být použity při návrhu videopreprocesoru (připojení testovacích LED atd.).

IO ARM konektor: Přímý výstup z ARM7. Měřený parametr vyjádřený pulsní šířkovou modulací nebo komunikace s nadřazeným systémem pomocí rozhraní SPI.

CMOS konektor: Propojení CMOS plošného senzoru a obvodu FPGA. Sběrnice obsahuje datové i synchronizační signály.

RS232 konektor: Připojení na jednoduchou sériovou linku RS232. Programování procesoru ARM7. Komunikace modulu a PC při nastavení funkcí videosenzoru. Přenos naměřených dat.

Rozhraní mezi ARM7 a FPGA lze rozdělit do čtyř skupin signálů:

SPI0: Toto rozhraní se na modulu používá pro konfiguraci FPGA pomocí procesoru ARM7 a dále pro posílání nalezených hran objektu nebo rozměru objektu (redukovaný obrazový signál).

PI: Paralelní 16ti bitové rozhraní pro posílání nalezených hran objektu nebo rozměru objektu (redukovaný obrazový signál). Rozdíl od SPI0 je ve vyšší rychlosti.

FLAGS: Pomocné příznakové signály videopreprocesoru.

CONF: Signály potřebné při konfiguraci FPGA.

Dle blokového schématu podle Obr 3.4 se bude pokračovat v návrhu propojení signálů (pinů) jednotlivých obvodů mezi sebou a mezi IO konektory. Datové přenosy budou popsány v dalších kapitolách, zejména v kapitole 6 *Videopreprocesor – vnější popis*.

3.3 Návrh propojení signálů FPGA - ARM7

Dle předchozí kapitoly bylo propojení pinů mezi FPGA a ARM7 rozděleno do čtyř skupin signálů (rozhraní) – SPI0, PI, FLAGS, CONF.

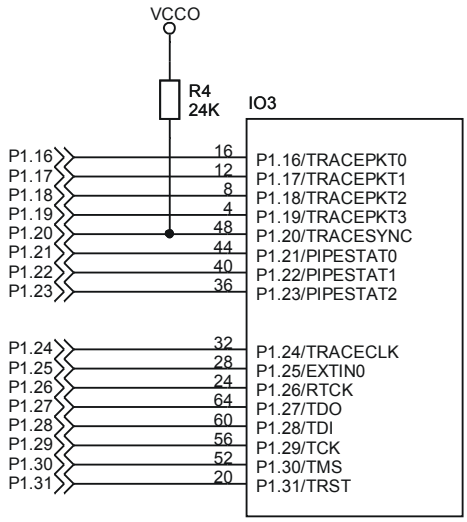
Propojení signálů pro **SPI0** a **CONF** (konfigurace FPGA + redukovaný obrazový signál) je popsáno v samostatné kapitole *4 Konfigurace FPGA pomocí ARM7*.

Dále bude popsáno propojení rozhraním **PI**. Rozhraní PI je 16ti bitové paralelní. Pro jeho

realizaci je potřeba 16 uživatelských IO pinů na straně procesoru ARM7 a 16 uživatelských pinů na straně FPGA.

Předpokládá se, že na signálech tohoto rozhraní se bude přenášet redukovaný obrazový signál, který bude vyjadřovat například pozici hrany objektu. Pozice hrany musí být procesorem přečtena jako hodnota (číslo v hexa soustavě).

Aby na straně ARM7 bylo jednoduché a rychlé načtení této hodnoty, měli by být piny rozhraní PI připojené k jedná bráně procesoru a piny by měli být v pořadí vedle sebe. K propojení se hodí brána P1 piny P1.16 až P1.31. Jedná se o uživatelské IO piny. Pro rozhraní SPI0, SPI1 a PWM by tyto piny použity být nemohly, protože to neumožňuje vnitřní architektura ARM7 a také proto se brána P1 použije pro rozhraní PI dle obrázku Obr 3.5.



Obr 3.5 Připojení signálů PI

Na straně FPGA mohou být signály P1.16 až P1.31 připojeny k libovolným uživatelským IO pinům. Seřazení pinů do požadovaného pořadí se provádí uvnitř architektury FPGA. (soubor *.vhd + soubor definice připojení výstupních pinů *.ucf). Připojení pinů bylo voleno tak, aby se signálové cesty na plošném spoji nekřížily.

Samotné paralelní rozhraní PI potřebuje také signály řízení přenosu. Nejméně dva. Signál pro platnost čtených dat a signál čtení dat. Signály pro zápis se nepředpokládají, protože rozhraní se uvažuje pouze výstupní.

Potřeba budou také signály, které informují o stavu naplnění FIFO paměti v FPGA (předpokládá se, že v FPGA bude FFO paměť realizována). Potřebné by byly signály *fifo ful*, *fifo empty*, *half fifo*.

Dále by bylo vhodné mít rezervu několika signálů s univerzálním použitím.

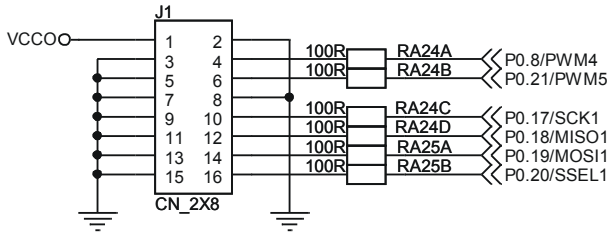
Pro propojení ARM7 a FPGA bude použito osm signálů pro řízení přenosu na rozhraní PI,

příznakové signály pro FIFO paměť a rezervní signály. Všechny jsou označeny jako **FLAGS**. Dle schématu v příloze jsou tyto signály připojeny na piny P0.9, P0.10,P0.11, P0.13, P0.15, P0.16, P0.23, P0.25 procesoru ARM7. Piny mohou být použity, protože se nejedná o signály pro rozhraní SPI0 ani SPI1 a, protože zbudou ještě dva signály pro PWM.

Všechny piny mezi obvodem FPGA a procesorem ARM7 jsou odděleny ochrannou odporovou sítí 100 Ω.

3.4 Návrh propojení signálů ARM7 – IO konektor

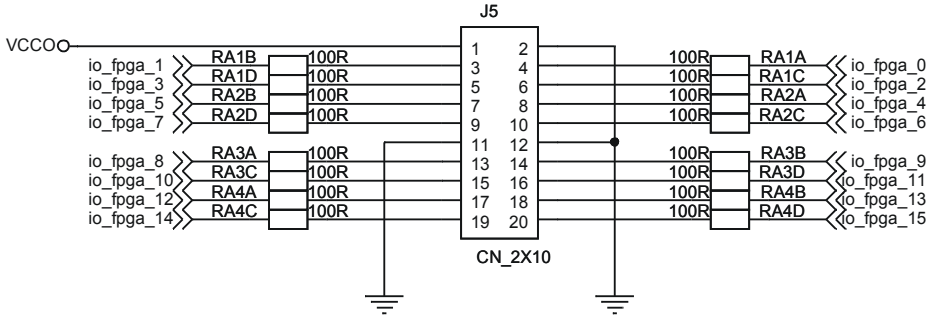
Dle předchozích úvah se pro IO ARM konektor použijí čtyři signály rozhraní SPI1 procesoru ARM7 (SCK1, MISO1, MOSI1, SSEL1) a dva signály PWM (PWM4 a PWM5) dle Obr 3.6. Na konektor jsou dále přivedeny zemnící signály a jeden signál napájení. Výstupní piny konektoru a procesoru ARM7 jsou odděleny ochrannou odporovou sítí 100 Ω.



Obr 3.6 Připojení signálů IO ARM konektoru

3.5 Návrh propojení signálů FPGA – IO konektor

Pro propojení obvodu FPGA s IO FPGA konektorem se použije 16 uživatelských IO pinů obvodu FPGA. Piny jsou voleny tak, aby se signálové cesty na plošném spoji nekřížily. Na konektor jsou dále přivedeny zemnící signály a jeden signál napájení. Piny jsou opět odděleny ochrannou odporovou sítí 100 Ω.

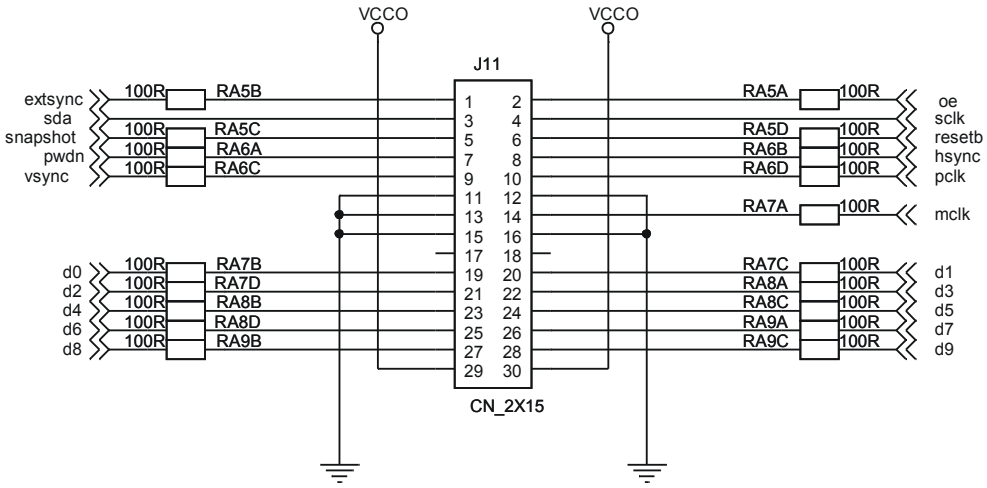


Obr 3.7 Připojení signálů IO FPGA konektoru

3.6 Návrh propojení signálů FPGA – CMOS konektor

CMOS konektor pro modul obrazového senzoru je propojen jak s obvodem FPGA, tak s procesorem ARM. S procesorem ARM7 je propojen sběrnicí I^2C. Tato sběrnice bude sloužit pro nastavení režimu práce obrazového senzoru. Ostatní piny, piny pro přenos obrazu a řídící piny obrazového senzoru jsou připojeny k obvodu FPGA. Rozmístění pinů na konektoru je podle modulu obrazového senzoru (semestrální projekt, Jan Šedivý, 2003).

Pro případ, že by videopreprocesor v FPGA pracoval na hodinovém kmitočtu signálu *pclk* (signál čtení dat z CMOS senzoru) je tento signál připojen a jeden z pinů globálního hodinového signálu FPGA. Piny jsou opět odděleny ochrannou odporovou sítí 100 Ω.



Obr 3.8 Připojení signálů CMOS konektoru

3.7 Návrh propojení signálů FPGA – EZ USB

Předpokládá se, že na signálech mezi obvodem FPGA a obvodem EZ USB se bude přenášet digitální obrazový signál. Dle blokového schématu na Obr 3.4 je označen hvězdičkou. Nejedná se o signál ve stejném formátu jako z CMOS obrazového senzoru, ale o signál ve formátu, který je obvodem FPGA upraven tak, aby mohl být načítán do vstupní FIFO paměti EZ USB.

Načítání obrazu do FPGA je možné synchronně či asynchronně. Pro oba režimy jsou potřeba signály PB0 až PB7 (datové signály) obvodu EZ USB. Budou spojeny s piny obvodu FPGA.

Při asynchronním přenosu jsou dále potřebné následující signály obvodu EZ USB: FIFOADR0, FIFOADR1, PKTEND, SLOE, SLWR . Všechny budou s FPGA propojeny.

Při synchronním přenosu jsou dále potřebné následující signály obvodu EZ USB: FIFOADR0, FIFOADR1, PKTEND, SLWR, IFCLK . Všechny budou s FPGA propojeny.

Pomocí uvedených signálů bude možný přenos obrazu v synchronním i asynchronním režimu.

Protože, na obvodu FPGA zbyl ještě dostatek volných uživatelských IO pinů jsou i další piny obvodu EZ USB s obvodem FPGA propojeny jako rezerva. Propojení signálů FPGA – EZ USB je patrné ze schématu v příloze *G*. Piny jsou opět odděleny ochrannou odporovou sítí 100 Ω.

3.8 Návrh propojení signálů ARM7 – RS232 konektor

Propojení pinů ARM7 s RS232 konektorem je pomocí čtyř signálů dle schématu v příloze *G*.

Datové signály TXD a RXD jsou vedeny z konektoru přes převodník úrovně RS232 linky (obvod MAX3232) na piny procesoru ARM7. Piny procesoru jsou TXD - pin P0.0 a RXD - pin P0.1.

Dále je s konektorem propojen pin procesoru P0.14. Pin se bude používat při programování procesoru ARM7. Úroveň nula na tomto pinu během resetu procesoru uvede procesor do stavu naprogramování interní FLASH (spustí se „boot loader“).

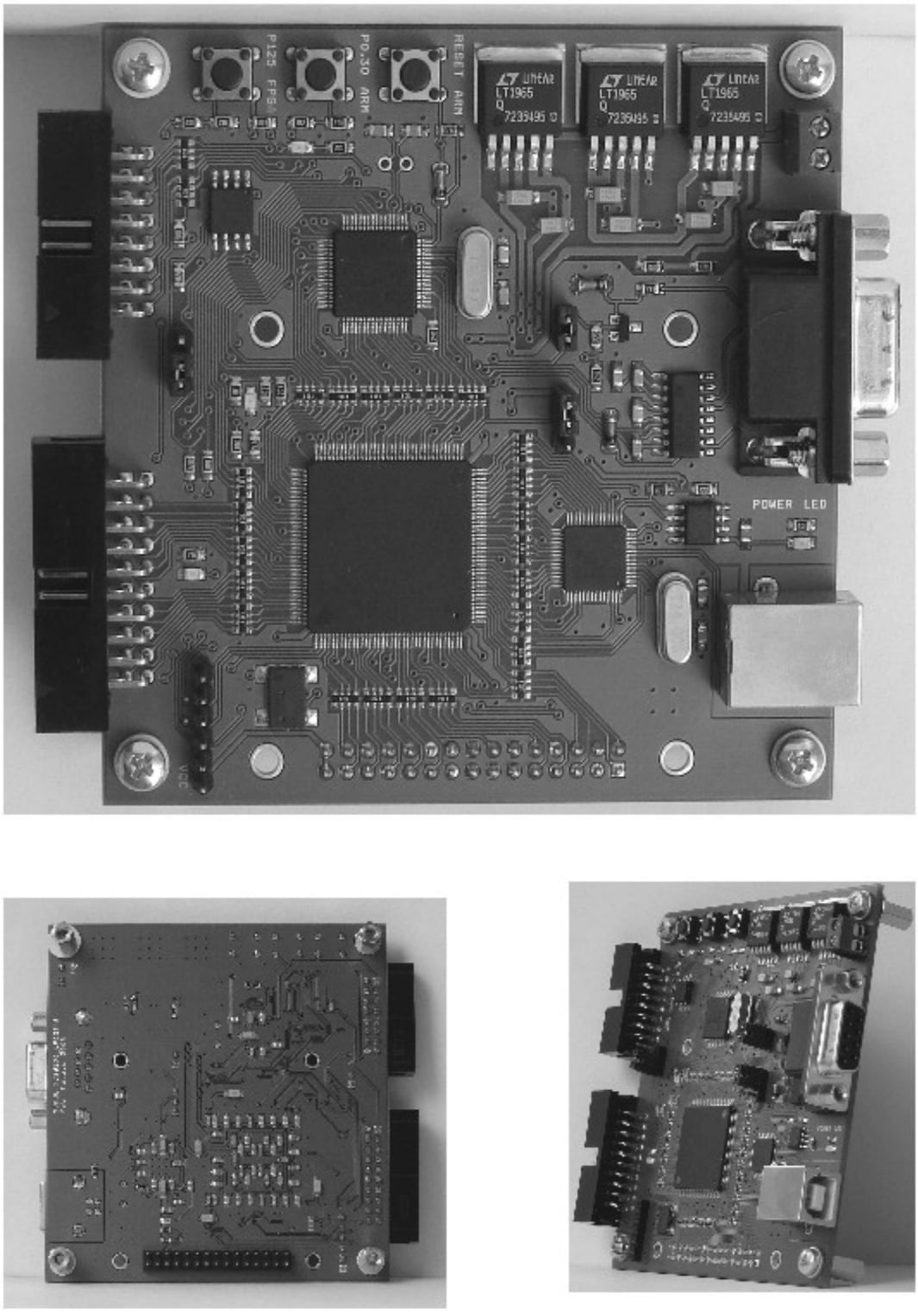
Poslední pin, který je přiveden na konektor je pin reset procesoru ARM7. Používat se bude též při programování procesoru ARM7. Signál reset procesoru ARM7 je dle schématu v příloze veden přes propojku J3 a tlačítko TL2. Propojka J3 umožňuje odpojení resetu procesoru od konektoru RS232. Tlačítko umožňuje manuální reset procesoru.

3.9 Připojení LED diod a tlačítek

K procesoru ARM7 je připojeno jedno uživatelské tlačítko (pin P0.30) a jedna LED dioda (pin P0.31) a také k obvodu FPGA je připojeno jedno uživatelské tlačítko (pin P125) a jedna LED dioda (pin P87).

3.10 Uživatelský popis modulu

Na základě uvedeného blokového schématu a na základě návrhu propojení obvodů a IO konektorů a po otestování propojení pomocí vývojových modulů se začalo pracovat na úplném elektronickém schématu modulu. Ze schématu byl následně vytvořen plošný spoj. Hotový modul (osazená deska) je na Obr 3.9. Úplné elektronické schéma modulu je v příloze *G*.

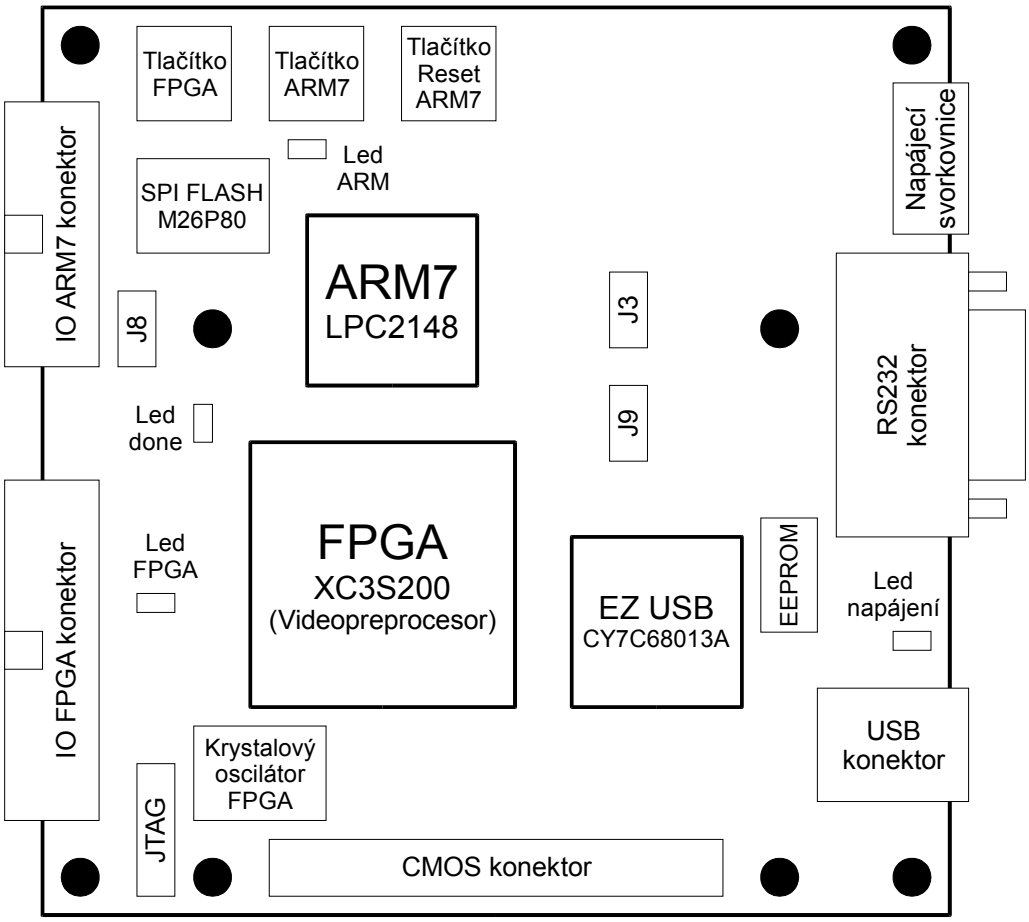


Obr 3.9 Modul zpracování obrazového signálu

Dvouvrstvý plošný spoj modulu má rozměr 9 cm krát 9 cm. Horní vrstva desky obsahuje většinu součástek a spoje. Spodní vrstva desky obsahuje převážně blokovací kondenzátory a rozlitou měď spojenou s GND. Deska má celkem 8 montážních děr průměru 3 mm, které slouží pro uchycení modulu obrazového senzoru a pro uchycení distančních sloupků v rozích modulu.

Rozmístění základních prvků modulu zpracování obrazového signálu je na Obr 3.10. V příloze tohoto dokumentu se nacházejí tabulky, které definují propojení obvodů mezi sebou, dále propojení obvodů s IO konektory, připojení tlačítek, diod a krystalového oscilátoru:

- Příloha B - IO konektory modulu
- Příloha C - tlačítka, led a oscilátor na modulu
- Příloha D - propojení pinů na modulu



Obr 3.10 Rozmístění prvků na modulu zpracování obrazového signálu

Napájecí svorkovnice (přívod napájecího kabelu) je zřetelně popsána na desce plošného spoje. Napájení je 5V DC s kladným pólem blíže nápisu 5V.

RS232 konektor je pro připojení na sériovou linku PC pomocí přímého (nekříženého) kabelu

zakončeného konektory CANNON 9. Nezbytné signály jsou TXD, RXD, RTS a DTR.

Krystalový oscilátor je zdrojem hodinového signálu pro obvod FPGA. Naprogramován je na kmitočet 50MHz.

Propojkou J3 (dvoupinová) lze odpojit signál DTR sériové linky od signálu RESET procesoru ARM7. Používá se při programování procesoru ARM7.

Propojkou J9 (třípinová) se volí jedna z variant programování obvodu FPGA. Pokud jsou propojeny dva piny blíže propojce J3 programuje se obvod FPGA pomocí procesoru ARM7. Pokud jsou propojeny dva piny dále od propojky J3 programuje se obvod FPGA pomocí rozhraní JTAG.

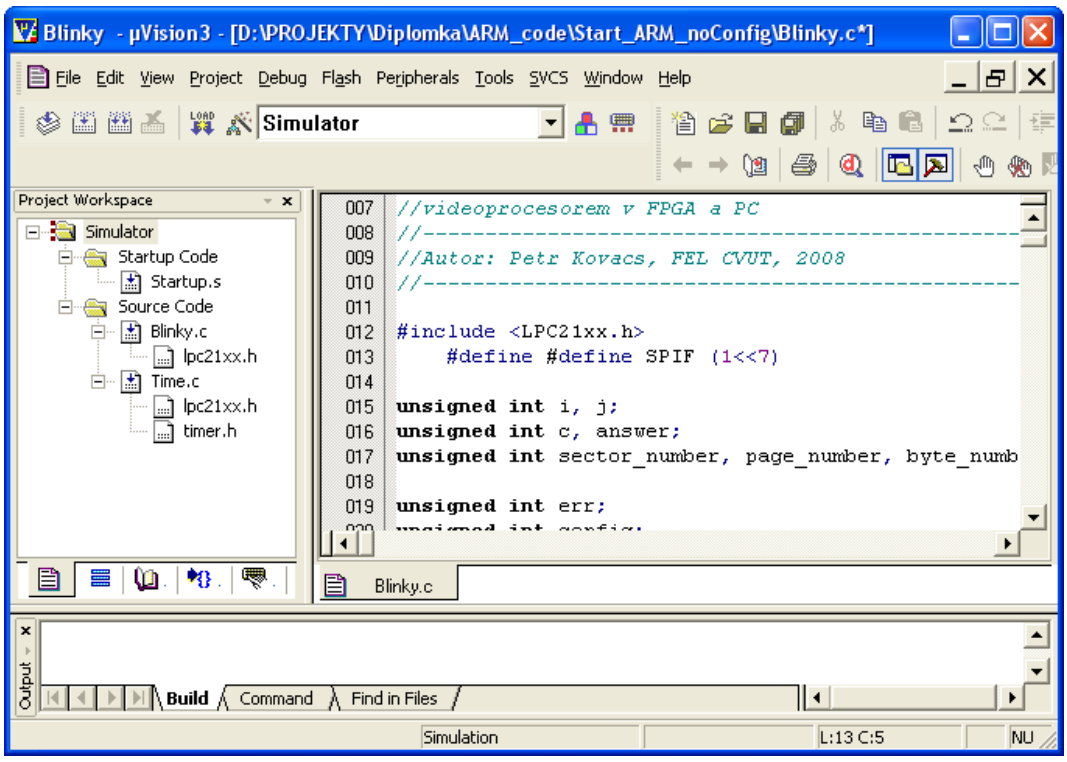
Propojkou J8 (třípinovou) se zakazuje zápis do paměti FLASH M26P80. Zápis je zakázán pokud jsou propojeny dva piny blíže konektoru JTAG. Zakázání zápisu ovšem souvisí s programovým nastavením FLASH dle (14, s. 8).

3.11 Programové vybavení pro vývoj aplikací

Aplikace pro procesor ARM7 a obvod EZ USB budou psány ve vývojovém prostředí KEIL v programovacím jazyce C. Toto vývojové prostředí se nachází na doprovodném CD:

Programy/mdk302a.exe

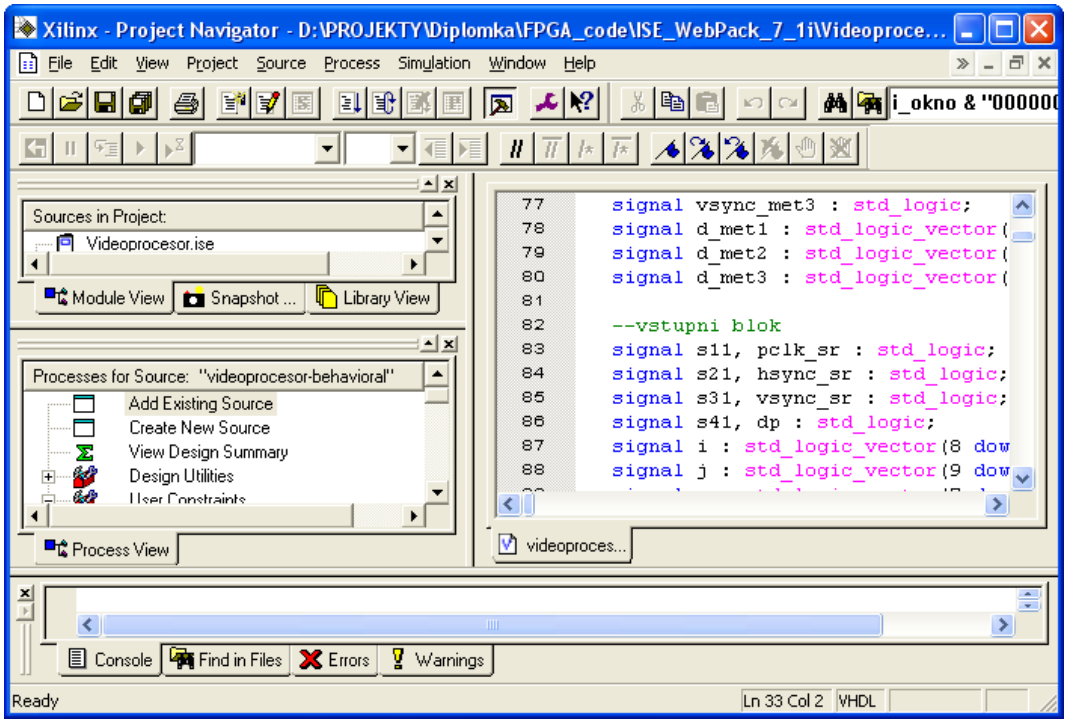
Přeložený program se do procesoru ARM7 a obvodu EZ USB nahraje dle kapitol *8.5 Nahrání programu do ARM7* a *8.7 Zobrazení naměřených dat.*



Obr 3.11 Vývojové prostředí KEIL

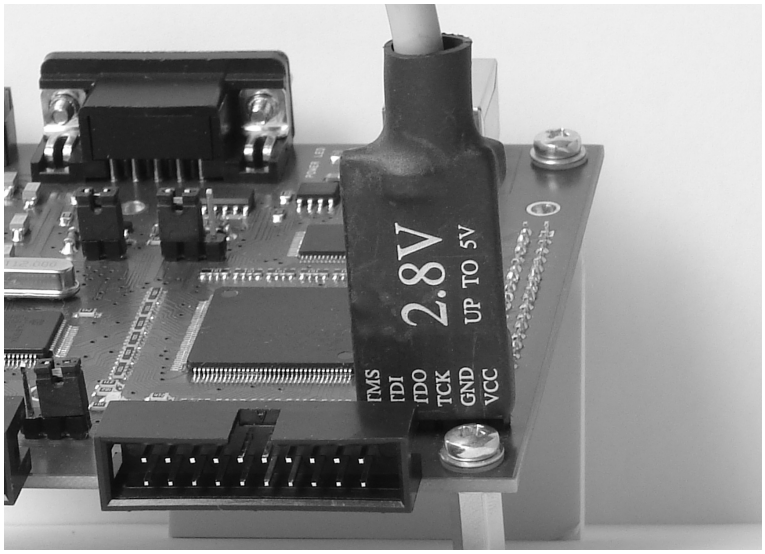
Aplikace pro procesor FPGA budou psány ve vývojovém prostředí ISE Web Pack v jazyce VHDL. Toto vývojové prostředí je za darmo ke stažení na webové adrese:

www.xilinx.com



Obr 3.12 Vývojové prostředí ISE Web Pack

Obvod FPGA se programuje z tohoto prostředí rozhraním JTAG programovacím kabelem na paralelní port PC firmy XILINX (varianta, která se používá při vývoji videopreprocesoru) nebo pomocí sériové linky dle kapitoly *8.6 Nahrání konfiguračního souboru do FPGA*.



Obr 3.13 Programovací kabel pro FPGA

4. Konfigurace FPGA pomocí ARM7

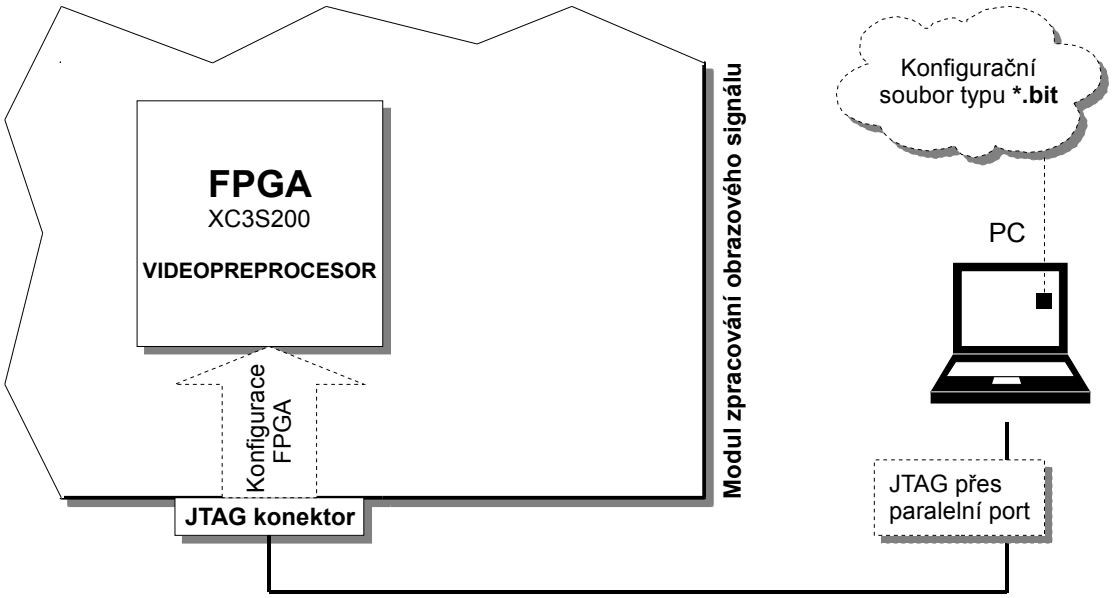
Tato kapitola popisuje způsob konfigurace FPGA v módu slave serial pomocí procesoru ARM7 a paměti FLASH SPI. Po úvodu ke konfiguraci FPGA budou popsány signály pro konfiguraci FPGA a jaký je formát a časování přenášených konfiguračních dat vůči těmto signálům. Následovat bude způsob propojení procesoru a FPGA s využitím rozhraní SPI procesoru. Závěr této kapitoly se zabývá implementací kódu pro procesor, který řídí přenos konfiguračních dat mezi počítačem, pamětí FLASH a FPGA.

4.1 Úvod ke konfiguraci FPGA

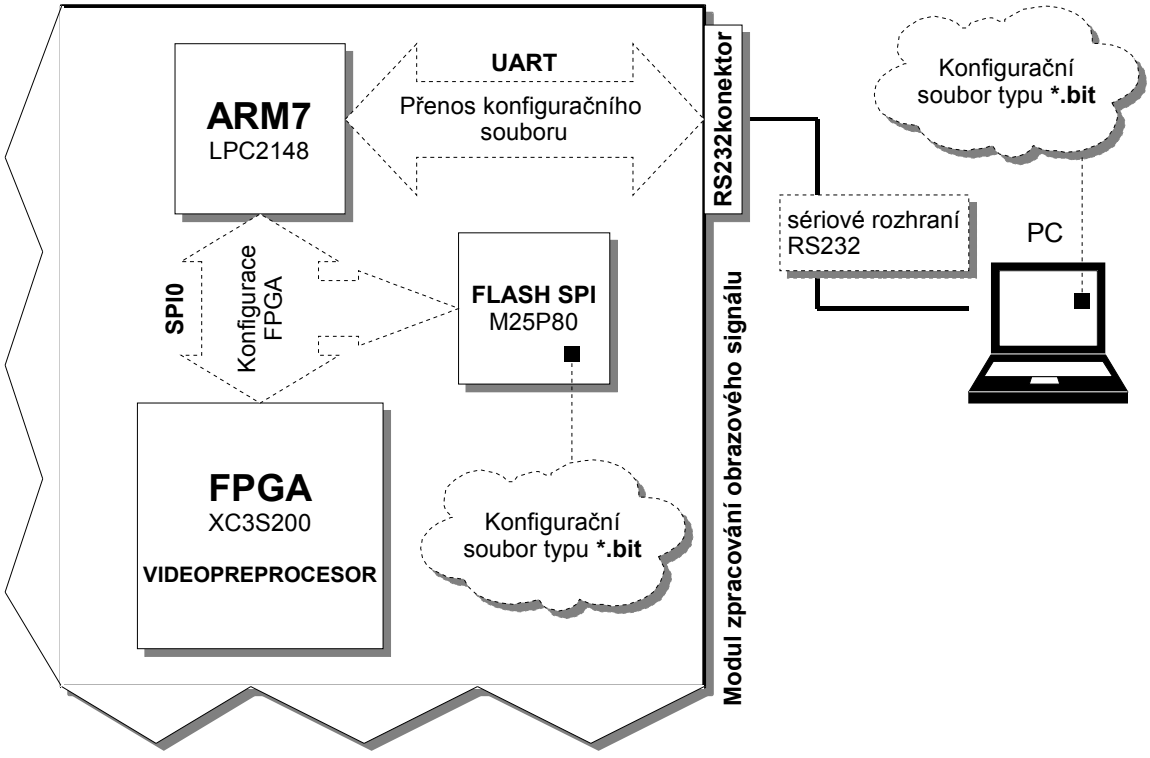
Návrh videopreprocesoru v FPGA (nebo jiného logického obvodu v této součástce) se skládá z dvou základních kroků. Prvním je popis videopreprocesoru jazykem VHDL a následná syntéza. Druhým krokem je implementace do zvoleného typu FPGA (nebo jiné programovatelné součástky). Výstupem implementace je konfigurační soubor typu ***.bit**. Problém, který se zde řeší je, jakým způsobem bude tento soubor nahráván do FPGA tedy jakým způsobem bude FPGA konfigurován. Modul zpracování obrazového signálu nabízí pro to dvě varianty.

První varianta je konfigurace FPGA pomocí rozhraní JTAG viz Obr 4.1. Tato varianta se používá při návrhu videopreprocesoru a testování jeho funkce. Pro konfiguraci FPGA pomocí tohoto rozhraní je zapotřebí JTAG adaptér na paralelní port, který je kompatibilní s programem ISE firmy XILINX. Při této variantě se konfigurační soubor nahrává přímo do paměti RAM obvodu FPGA a to znamená, že po vypnutí napájení se konfigurace maže a po opětovném zapnutí v paměti RAM již není.

Druhá varianta konfigurace FPGA je pomocí sériového rozhraní RS232 procesoru ARM7 a paměti SPI FLASH viz. Obr 4.2. Tato varianta se pro uživatele jeví jako update firmware pro modul zpracování obrazového signálu. Výhodou je, že uživatel nepotřebuje žádný speciální program ze strany počítače, ale stačí jen terminál, který umí komunikovat po sériové lince a odesílat soubor. Při této variantě je konfigurační soubor nahráván do paměti FLASH SPI (samostatný čip na modulu) a to znamená, že se konfigurační soubor uchová v této paměti i po dobu vypnutého napájení.



Obr 4.1 Konfigurace FPGA pomocí rozhraní JTAG



Obr 4.2 Konfigurace FPGA pomocí rozhraní RS232 a paměti FLASH

Celý proces konfigurace FPGA po sériovém portu je řízen procesorem ARM7. Procesor zajišťuje jak přenos konfiguračního souboru po sériovém portu z terminálu počítače, tak uložení

souboru do paměti FLASH a následnou konfiguraci FPGA rozhraním SPI při restartu modulu. Znamená to, že po zapnutí napájení začne FPGA pracovat jako videopreprocesor.

4.2 Konfigurační soubor

Konfigurační soubor je posloupnost bitů, kterými se programuje RAM obvodu FPGA a který definuje propojení jeho logických bloků. V této práci se používá konfigurační soubor typu *.bit, který bývá dle (18, s. 7) označován anglicky Binary. Existují ještě další dva typy konfiguračního souboru. Soubor typu *.hex bývá dle (18, s. 7) označován jako Hex a soubor typu *.rbt bývá dle (18, s. 7) označován jako Rawbits. Jednotlivé typy se mezi sebou liší zejména kompaktností.

Soubor typu Rawbits kóduje každý konfigurační bit jako jeden ASCII Byte. Tento typ je nejméně kompaktní a tedy zabírá nejvíce místa na disku. Pro FPGA typu XC3S200 je jeho velikost 1,06 MB. Jeho přenos po sériové lince by trval nejdelší dobu. Při rychlosti 115200Bd asi 1,5 minuty a při rychlosti 9600Bd asi 17 minut.

Soubor typu Hex representuje skupinu čtyř konfiguračních bitů jako jeden hex digit. Jeden hex digit má hodnotu od 0 do F a je dále kódován opět v ASCII. Tento typ je kompaktnější než Rawbits.

Nejkompaktnější je soubor typu Binary. Tento typ je kódován binárně, to znamená, že v souboru je uložen jeden konfigurační bit za druhým. Pro FPGA typu XC3S200 je jeho velikost 127 kB. Při rychlosti 115200Bd by jeho přenos po sériové lince trval asi 11 sekund a při rychlosti 9600Bd asi 132 sekund.

Výhodou souboru Binary je nejen jeho kompaktnost, ale i snadný způsob jeho zpracování procesorem. Soubor se po sériové lince přenáší po Bytech a protože každý Byte obsahuje pouze užitečné bity (konfigurační bity) může být každý tento Byte v nezměněném tvaru uložen do paměti FLASH kam se také data ukládají po Bytech. Jednoduché bude také jejich odesílání do obvodu FPGA po rozhraní SPI, které probíhá také po Bytech.

4.3 Paměť FLASH SPI pro konfiguraci FPGA

Konfigurační soubor se na modulu zpracování obrazového signálu ukládá do paměti FLASH typu M25P80. Jedná se o paměť s rozhraním SPI (serial peripheral interface). Její velikost je 1MB (přesně1048576 B).

Paměť je organizována do 16 sektorů. Každý sektor obsahuje 256 stránek a každá stránka 256 Byte. Do paměti lze tedy uložit až 8 konfiguračních souborů (každý zabere dva sektory). Pro detailnější informace o paměti se text odkazuje na (14).

4.4 Popis konfigurace FPGA v módu slave serial

Obecně může být obvod FPGA konfigurován procesorem v módech master serial, slave serial, master parallel, slave parallel a JTAG. Módy se mezi sebou liší tím zda jsou konfigurační data přenášena sériově či paralelně a zda synchronizační hodinový signál generuje procesor (mód slave) nebo FPGA (mód master). Nejlepší řešení pro modul zpracování obrazového signálu je použít mód slave serial (pomine-li se, že JTAG se zde používá jen při návrhu log. obvodu v FPGA) zejména díky možnosti sloučit tento mód s rozhraním SPI se kterým je kompatibilní jak použitá paměť FLASH tak i procesor ARM7.

Ke konfiguraci v módu slave serial je potřeba celkem sedm signálů FPGA. Signály spolu s jejich významem jsou uvedeny v tabulce 1 Originál tabulky v angličtině spolu s popisem módu slave serial je také uveden v (17, s. 179).

Slovní popis konfigurace FPGA pomocí signálů uvedených v tabulce je následující:

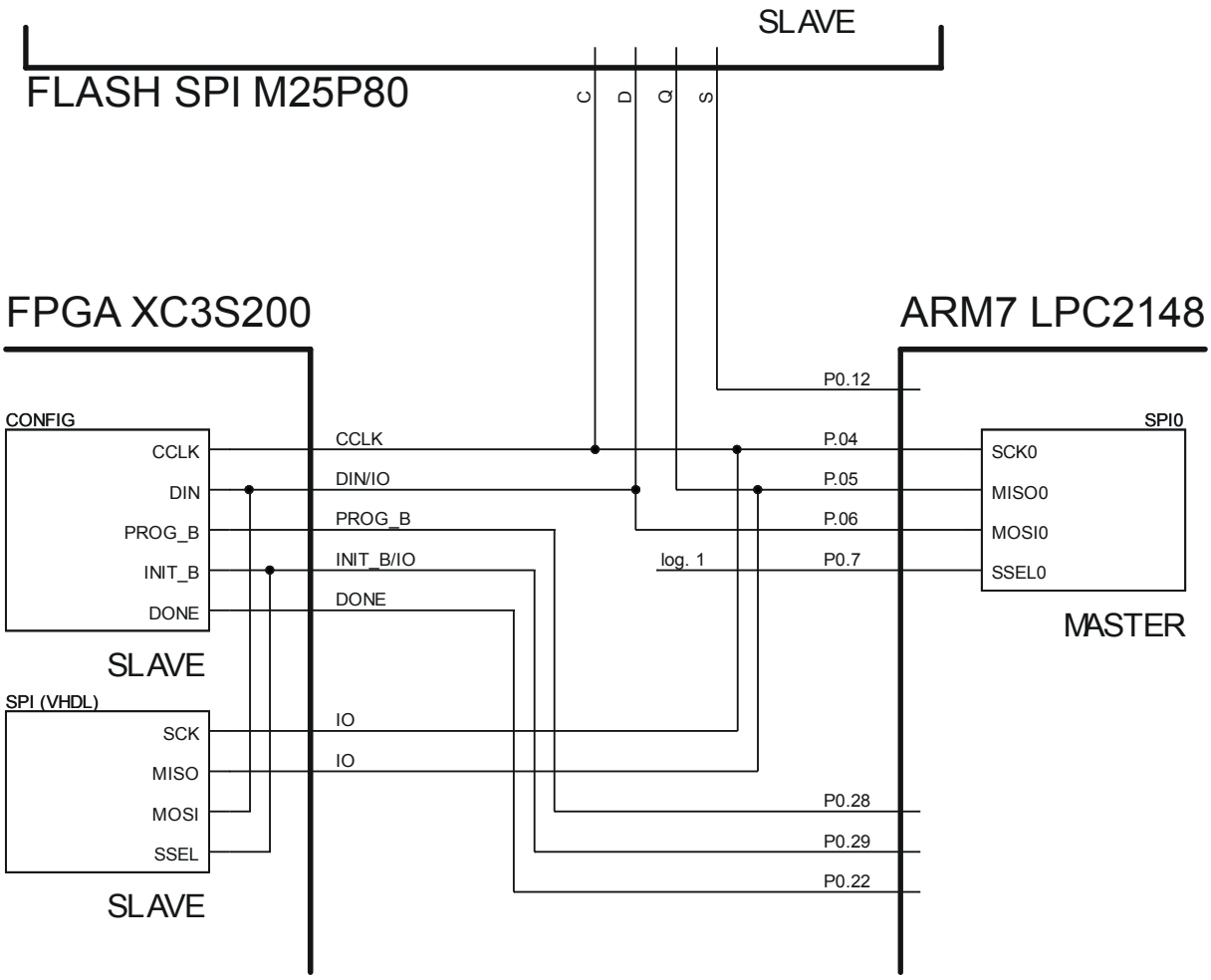
FPGA se do módu slave serial uvede nastavením signálu M2 = M1 = M0 = 1. Procesor začne konfiguraci pulsem do log. 0 na signálu PROG_B o délce minimálně 300ns a čeká až signál INIT_B přejde do log. 1. Následně započne posílání konfiguračních bitů do DIN synchronně s náběžnou hranou hodinového signálu CCLK. Během posílání dat monitoruje signály DONE a INIT_B. Pokud DONE přejde do log. 1 je FPGA úspěšně nakonfigurován. Pokud signál INIT_B přejde do log. 0 došlo k CRC chybě. Po přenosu všech konfiguračních bitů se požaduje ještě několik hodinových impulsů CLK pro rozběh FPGA.

název	směr	popis	Stav během konfigurace	Stav po konfiguraci
HSWAP	vstup	**Zapínání pull-up odporů na IO pinech**. Když je log. 0 připojí pull-up odpory na všechny IO piny. 0: pull-up odpory během konfigurace. 1: bez pull-up odporů.	V této práci je trvale připojen na log. 0.	Uživatelský IO pin.
M[2:0]	vstup	**Výběr módu konfigurace FPGA**.	Pro mód slave serial M2 = log. 1, M1 = log. 1, M0 = log. 1. Vzorkován když INIT_B přejde do log. 1.	Uživatelský IO pin.
DIN	vstup	**Datový vstup**.	Posílání konfiguračních dat na vzestupnou hranu CCLK.	Uživatelský IO pin.
CCLK	vstup	**Konfigurační hodinový signál synchronní s DIN**.	Externí hodinový signál.	Uživatelský IO pin.
INIT_B	Otevřený kolektor, obousměrný	**Indikátor inicializace**. Aktivní v log. 0. na začátku konfigurace během mazání konfigurační paměti. FPGA poté přejde do stavu log. 1.	Jestliže přejde do stavu log. 0 indikuje CRC chybu.	Uživatelský IO pin.
DONE	Otevřený kolektor, obousměrný	**Indikátor úspěšné konfigurace FPGA**. Po úspěšné konfiguraci přejde z log. 0 do log. 1.	Log. 0 indikuje, že FPGA ještě není naprogramován.	Připojen na log. 1 přes externí pull-up rezistor. Log. 1 indikuje, že FPGA je naprogramován.
PROG_B	vstup	**Start programování FPGA**. Aktivní v log. 0. Puls 300ns nebo delší restartuje konfigurační proces.	Musí být v log.1 pro začátek konfigurace.	Puls do log. 0 restartuje konfigurační proces.

Tabulka 1: Popis signálů pro konfiguraci FPGA v módu slave serial

4.5 Propojení signálů pro konfiguraci FPGA s ARM7 a FLASH

Vzájemné propojení FPGA s procesorem ARM7 a pamětí FLASH pomocí rozhraní SPI se považuje za jeden z velmi dobře vyřešených problémů v této práci. Propojení má tu dobrou vlastnost, že se využívá minima signálů (spojů na plošném spoji) a navíc některé signály přejdou po ukončení konfigurace do uživatelského módů a mohou být použity jako signály SPI popsaného jazykem VHDL. Druhá dobrá vlastnost je maximální využití přenosového kanálu na SPI a tedy vysoká rychlost konfigurace FPGA po zapnutí modulu. Propojení je uvedeno na Obr 4.3. Jedná se o logické schéma propojení a neuvádí se zde ochranné ani pull-up odpory.



Obr 4.3: Propojení signálů pro konfiguraci FPGA - logické schéma

Poznámka: Při propojení signálů se počítá s tím, že signály výběru módu konfigurace M[2:0] jsou všechny zapojeny na log. 1 pomocí jumperu J9, který je na modulu pro mód slave serial umístěn blíže stabilizátorům napětí. Signál HSWAP je trvale na log. 0.

Možnost připojit signály pro konfiguraci FPGA na signály rozhraní SPI je dána tím, že se v obou případech jedná o sériový synchronní přenos. Zde signál CCLK = SCK0 a signál

DIN = MOSI. Důležité je, aby byla správně nastavena polarita a fázový posun signálu SCK0. Nastavení se provádí v procesoru ARM7, který je pro SPI0 v režimu MASTER. V jeho registrech je nutno nastavit CPOL = 0 a CPHA = 0. S uvedeným nastavením pracuje i paměť FLASH.

V zásadě se rozhraní SPI používá pro komunikaci dvou obvodů v jednom čase. Uvedené propojení však počítá s tím, že při konfiguraci jsou současně připojeny všechny obvody tedy ARM7, FPGA i FLASH. Takovou variantu lze použít pokud pouze jeden z obvodů v režimu SLAVE je výstupní a vstupní a ostatní obvody v režimu SLAVE jsou vstupní. Nemůže tak dojít ke kolizi na sběrnici. Zde výstupní a vstupní obvod v režimu SLAVE je FLASH, vstupní je FPGA a ARM7 je v režimu master.

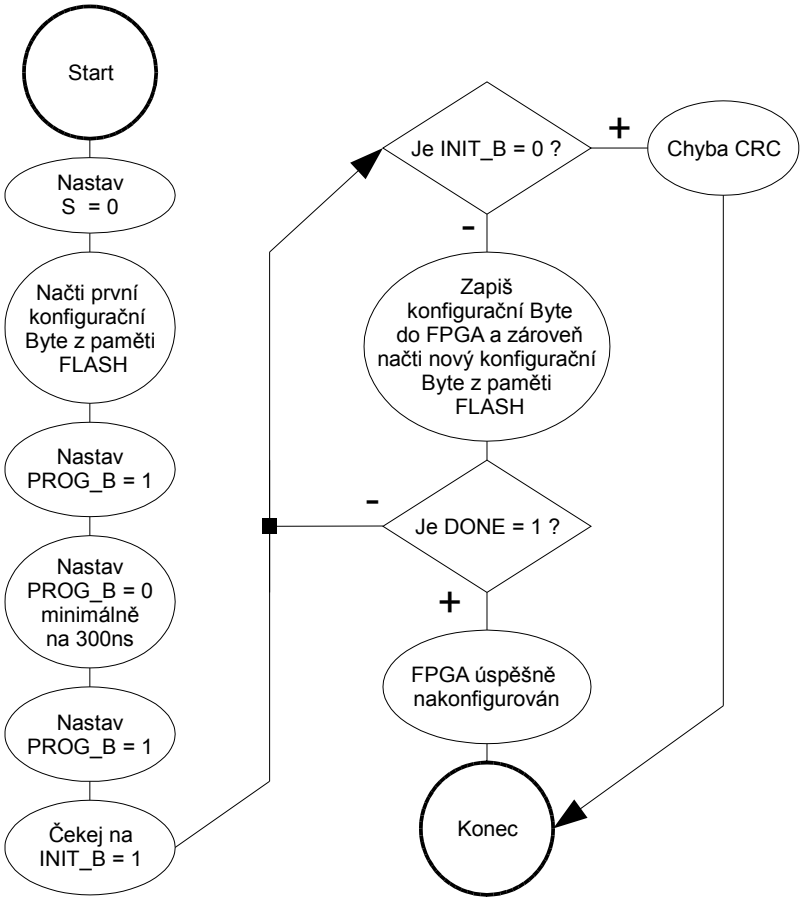
Maximální využití přenosového kanálu na sběrnici SPI je dáno právě současným připojením všech obvodů na sběrnici. Obecně se při čtení dat z paměti FLASH po SPI musí vysílat tzv. dummy Byte, které způsobí načtení Byte do vstupního registru SPI u ARM7. Zde se však místo dummy Byte vysílají konfigurační Byte a **zároveň tak dochází k načítání dat z paměti FLASH a odesílání dat do FPGA**. Musí být ovšem dodržen postup nastavování signálů FPGA a FLASH a čtení dat při procesu konfigurace přesně dle vývojového diagramu na Obr 4.4.

Konfigurace FPGA je pomocí uvedeného propojení umožněna také proto, že se data z paměti FLASH dají kontinuálně vyčíst v jednom cyklu. Instrukce pro nastavení čtení z paměti FLASH se posílají jen na začátku cyklu čtení. S ohledem na velikost konfiguračního souboru a s ohledem na velikost paměti FLASH se čtou nejednou dva sektory paměti.

Na Obr 4.3 je také uveden blok s názvem SPI (VHDL). Je tím naznačena situace, která se předpokládá, že nastane po ukončení konfigurace FPGA. Videopreprocesor popsaný jazykem VHDL využívá pro komunikaci s procesorem ARM7 jeho blok rozhraní SPI0 stejný jako při konfiguraci. Některé signály tohoto bloku použité při konfiguraci jsou následně využity jako signály ve VHDL. Šetří se tím počet signálových spojů na plošném spoji.

Dle uvedeného diagramu byl napsán program v jazyku C do procesoru ARM7. Program se nachází na doprovodném CD:

ARM_code\Start_ARM\Blinky.Uv2



Obr 4.4 Vývojový diagram - konfigurace FPGA

Při ověřování funkce uvedeného programu pro konfigurování FPGA se objevil problém s pinem P1.26. Dle (12, s. 10) úroveň log. 0 na tomto pinu při resetu procesoru (tlačítkem RESET ARM) uvede jeho piny P1.31 až P1.26 do DEBUG módu. To je ovšem nežádoucí, protože tyto piny musí být použity jako uživatelské GPIO piny pro paralelní rozhraní PI (propojené s FPGA).

Problém se podařilo vyřešit takto: Za prvé VHDL popis videopreprocesoru je napsán tak, že při resetu videopreprocesoru je na pinu P77 obvodu FPGA log. 1. Za druhé po resetu procesoru ARM proběhne konfigurace FPGA (tedy pin P1.26 spojen s P77 je v log. 1) a následně proběhne reset procesoru ARM pomocí jeho watchdog bloku. Po resetu pomocí watchdog již konfigurace neprobíhá. V uvedeném programu se jedná o řádky 82-83 a dále 229-241

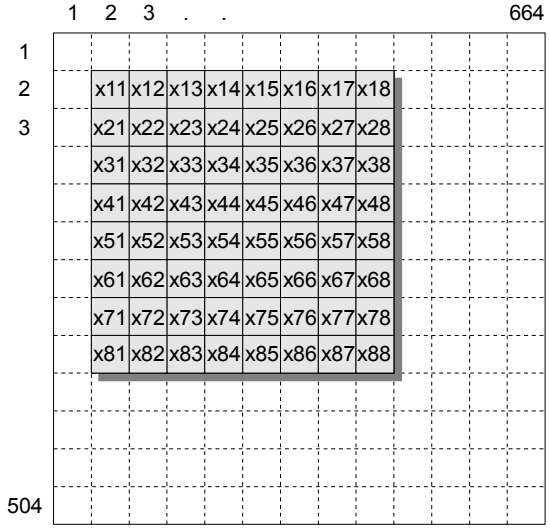
5. Metody zpracování obrazového signálu

Videopreprocesor bude implementovat metody zpracování obrazového signálu pro bezkontaktní měření objektů. Jedná se o základní metody, které budou popsány v této kapitole.

V kapitolách se uvažuje snímek složený z matice pixelů. Snímek je černobílý (kódování každého pixelu pomocí osmi bitů) a zachycuje obraz měřeného objektu.

5.1 Obrazová matice pixelů 8x8

Všechny výpočty metod zpracování obrazu se provádějí z obrazové matice pixelů 8x8 pixelů. Matice se posunuje na snímku zleva doprava po jednom sloupci a na konci každého řádku se posune o jeden řádek níže. Pixely matice jsou označeny jako *xij*, kde písmeno *i* označuje řádek a písmeno *j* označuje sloupec.

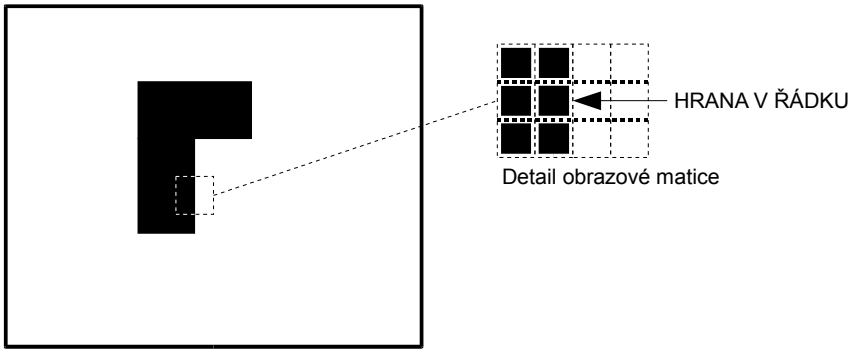


Obr 5.1 Matice pixelů 8x8

Samotná realizace získání matice pixelů z obrazového signálu bude pomocí blokových RAM obvodu FPGA.

5.2 Detekce horizontálních hran

Horizontální hrany jsou hrany ve směru řádků dle Obr 5.2.



Obr 5.2 Hledání horizontálních hran

Nalezení hrany probíhá ve dvou krocích. V prvním kroku se vypočítá konvoluce z hodnot obrazové matice. V druhém kroku se určí jestli vypočítaná hodnota konvoluce překročila prahovou hodnotu *diffk* a pokud ano je nalezena hrana. Pro výpočet konvoluce jsou použity čtyři různé vzorce a proto budou rozlišovány čtyři typy metod nalezení hran podle toho, který vzorec se použije. Výpočet konvoluce je dle těchto vzorců:

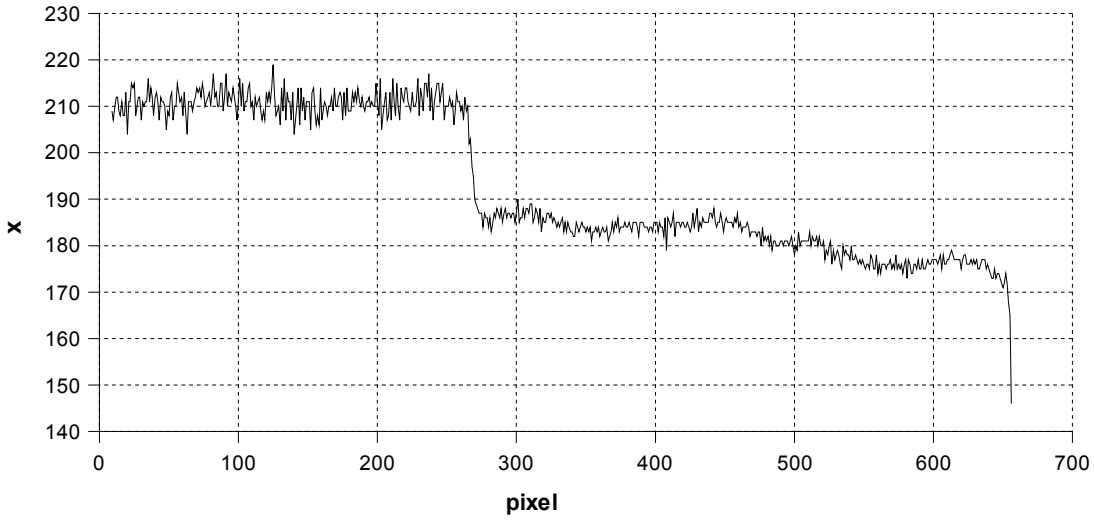
$$s0dif1 = |x54 - x55|$$
$$s0dif2 = |x54 - x55| + |x53 - x56|$$
$$s0dif3 = |x54 - x55| + |x53 - x56| + |x52 - x57|$$
$$s0dif4 = |x54 - x55| + |x53 - x56| + |x52 - x57| + |x51 - x58|$$

Těmto vzorcům odpovídají tyto konvoluční jádra:

s0dif1: |*1 -1*|
s0dif2: |*1 1 -1 -1*|
s0dif3: |*1 1 1 -1 -1 -1*|
s0dif4: |*1 1 1 1 -1 -1 -1 -1*|

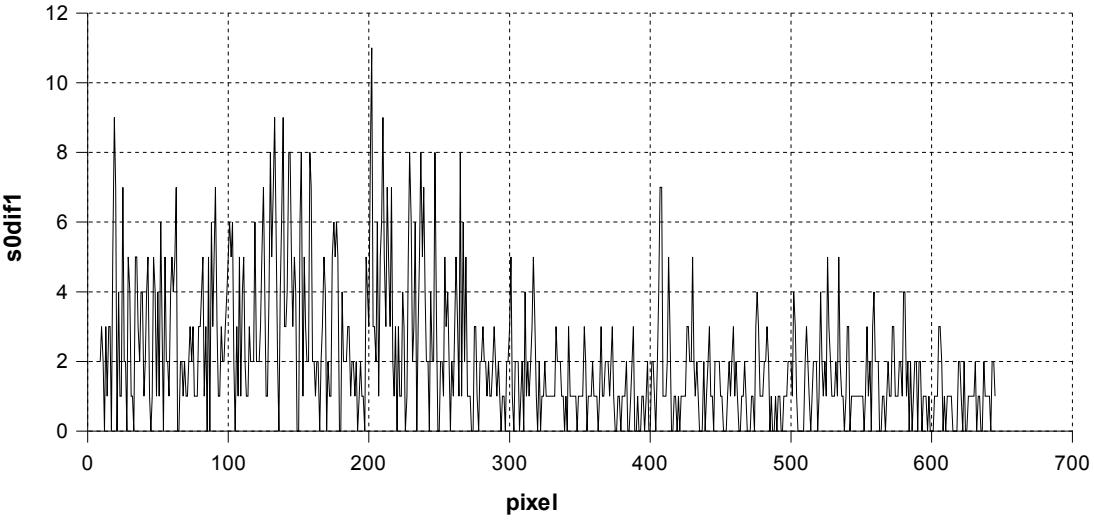
Metody byly testovány v tabulkovém procesoru. Za tímto účelem musel být k dispozici soubor skutečných hodnot z CMOS plošného senzoru. Pro detekci hran v řádku stačí hodnoty z jednoho řádku (jedná se o jasovou funkci jednoho řádku).

Za účelem získání těchto hodnot byl vzat modul CMOS obrazového senzoru, dále vývojový modul s FPGA a nakonec vývojový modul s ARM7 s výstupem sériové linky zapojené do PC. Propojeny byly v pořadí jak bylo v minulé větě napsáno. Do obvodu FPGA byla jazykem VHLD popsána FIFO paměť s výstupem na SPI rozhraní (také popsané jazykem VHDL). Procesor ARM7 vyčítal po SPI hodnoty s FIFO a následně je posílal po sériové lince do PC. V PC byly hodnoty odchyceny terminálem a zkopírovány do tabulkového procesoru. V tabulkovém procesoru byl vytvořen graf na Obr 5.3. Obrázek zachycuje přechod bílá – černá (bílý papír na černém podkladu). V obrázku se vyskytuje jedna hrana na pozici asi 275-tého pixelu.



Obr 5.3 Jasová funkce jednoho řádku z CMOS senzoru

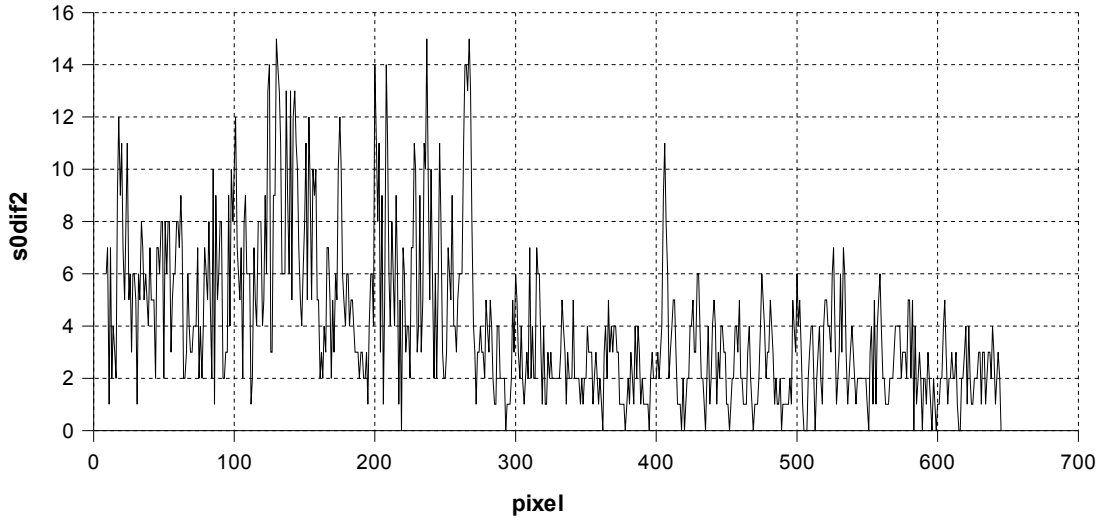
Na hodnoty jasové funkce z uvedeného obrázku byl proveden výpočet s konvolučním jádrem *[1 -1]* viz. graf na Obr 5.4.



Obr 5.4 Konvoluce s jádrem *[1 -1]*

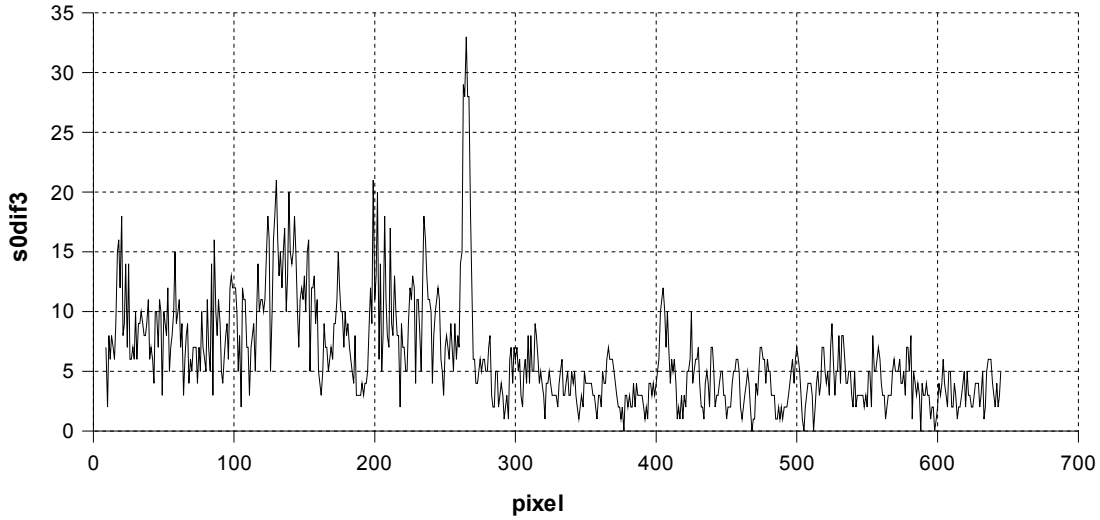
Z uvedeného grafu není poznat, kde se hrana vyskytuje. Důvodem je to, že konvoluce s jádrem *[1 -1]* (což je aproximace první derivace) je příliš náchylná na šum, který se v jasové funkci dle Obr 5.3 vyskytuje.

Při výpočtu konvoluce s jádrem *[1 1 -1 -1]* viz. Obr 5.5 ještě stále nelze rozpoznat pozici hrany.



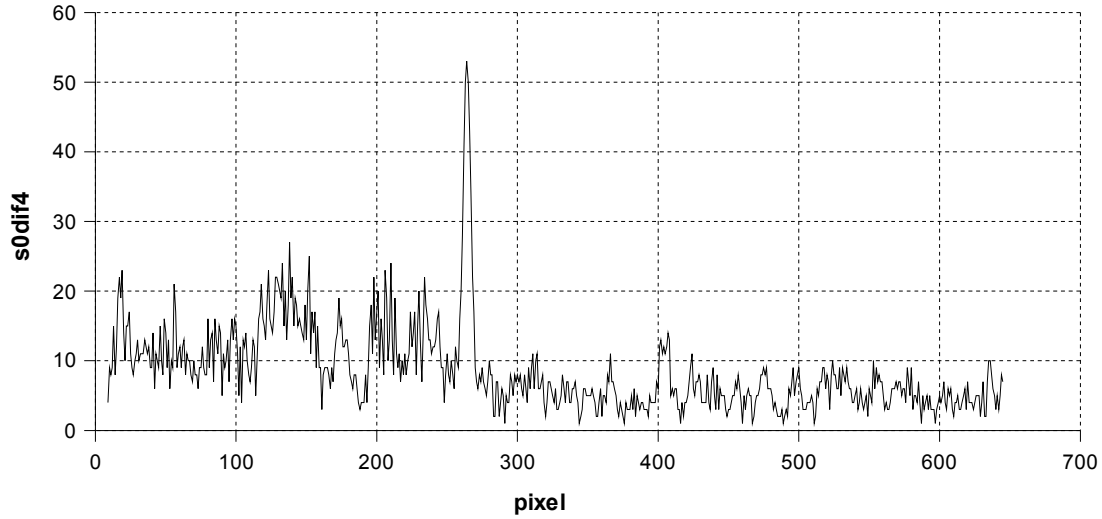
Obr 5.5 Konvoluce s jádrem *[1 1 -1 -1]*

Až při výpočtu konvoluce s jádrem *[1 1 1 -1 -1 -1]* viz.Obr 5.6 lze rozpoznat hranu na pozici 275-tého pixelu.



Obr 5.6 Konvoluce s jádrem *[1 1 1 -1 -1 -1]*

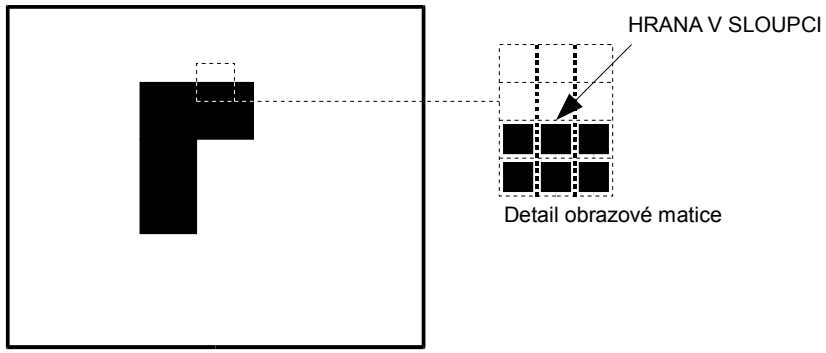
Při výpočtu konvoluce s jádrem *[1 1 1 1 -1 -1 -1 -1]* viz. Obr 5.7 je výsledek ještě o poznání lepší. Prahová konstanta *diffk* by zde mohla být nastavena například na hodnotu 40. Detekce hrany s tímto konvolučním jádrem je sice oproti detekci hrany s konvolučním jádrem *[1 -1]* méně přesná, ale odolnější na šum.



Obr 5.7 Konvoluce s jádrem *[1 1 1 1 -1 -1 -1 -1]*

5.3 Detekce vertikálních hran

Vertikální hrany jsou hrany ve směru sloupců dle Obr 5.8.



Obr 5.8 Hledání vertikálních hran

Detekce vertikálních hran je systémově stejná jako detekce horizontálních hran s dvěma rozdíly. Za prvé výpočet konvoluce se provádí sloupcově:

$$s0dif1v = |x45 - x55|$$
$$s0dif2v = |x45 - x55| + |x35 - x65|$$
$$s0dif3v = |x45 - x55| + |x35 - x65| + |x25 - x75|$$
$$s0dif3v = |x45 - x55| + |x35 - x65| + |x25 - x75| + |x15 - x85|$$

Odpovídající konvoluční jádra jsou:

s0dif1v: $|1\ -1|^T$
s0dif2v: $|1\ 1\ -1\ -1|^T$
s0dif3v: $|1\ 1\ 1\ -1\ -1\ -1|^T$
s0dif4v: $|1\ 1\ 1\ 1\ -1\ -1\ -1\ -1|^T$

Druhý rozdíl je následující. Konvoluční jádro (matice pixelů) se po obraze pohybuje kolmo k hranám. Znamená to, že pokud je v obraze jen jedna vertikální hrana potom je na každém řádku nalezena znovu dokud konvoluční jádro (matice pixelů) neopustí oblast této hrany. U signálu *0dif1v* tento problém nehrozí u signálů *0dif2v* - *0dif4v* ovšem ano.

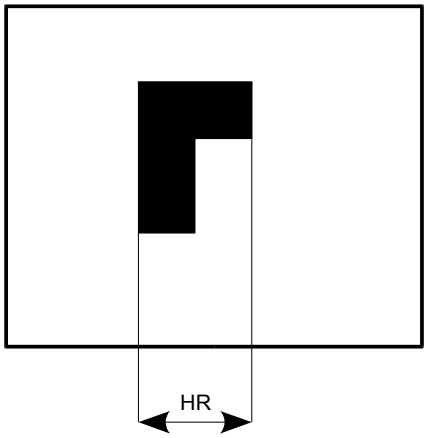
Při realizaci této metody v FPGA se na tento problém bude muset myslet. Bude muset být použita jednobitová paměť, kde bude uloženo jestli hrana již byla detekována nebo ne. Paměť musí být ovšem dlouhá jako počet pixelů v řádku jednoho snímku a její čtení musí být synchronní se čtením hodnot pixelů z CMOS plošného senzoru.

K realizaci metod detekce hran budou v FPGA muset být čítače sloupců a řádků a komparátor. Pomocí komparátoru se bude sledovat zda hodnota signálů *s0dif1–4* nebo *s0difv0–4* překročila konstantu *diffk*. Pokud ano bude aktivován signál přerušení a hodnota čítače sloupce nebo řádku se zapíše do registrů odkud bude dále vyčtena.

5.4 Měření horizontálního a vertikálního rozměru

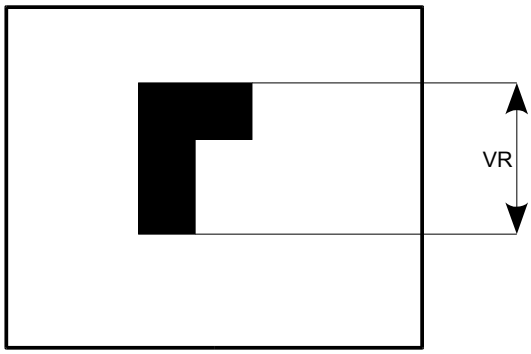
Do videopreprocesoru budou implementovány jednoduché metody pro měření horizontálního a vertikálního rozměru. Bude počítat s tím, že je na snímku jen jeden objekt.

Při měření horizontálního rozměru se pomocí metod nalezení hran detekuje první a poslední horizontální hrana objektu a následně se vypočítá její horizontální rozměr označený dle obrázku Obr 5.9 jako *HR*.



Obr 5.9 Měření horizontálního rozměru

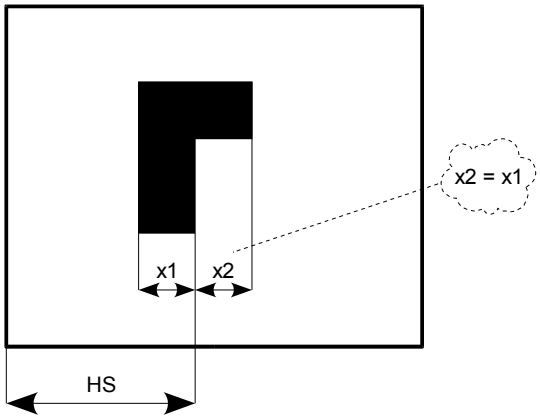
Při měření vertikálního rozměru se pomocí metod nalezení hran detekuje první a poslední vertikální hrana objektu a následně se vypočítá její vertikální rozměr označený dle obrázku Obr 5.10 jako V*R*.



Obr 5.10 Měření vertikálního rozměru

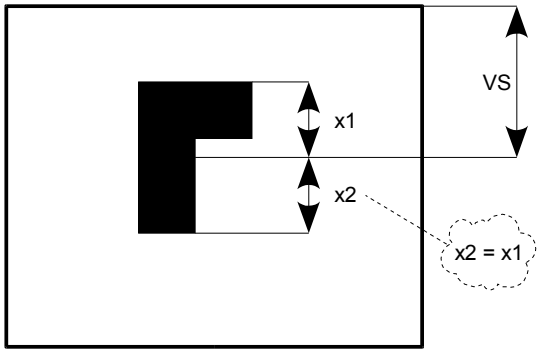
5.5 Měření horizontálního a vertikálního středu

Při měření horizontálního a vertikálního středu se opět počítá s tím, že je na snímku jeden objekt. Při měření horizontálního středu se pomocí metod nalezení hran detekuje první a poslední horizontální hrana objektu a následně se vypočítá její horizontální střed označený dle obrázku Obr 5.11 jako *HS*.



Obr 5.11 Měření horizontálního středu

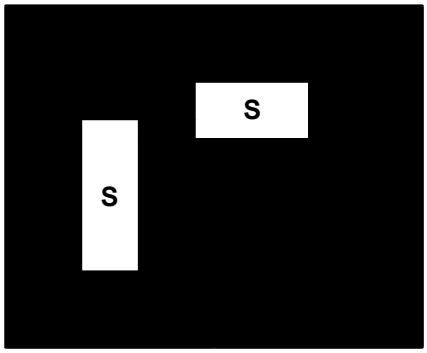
Systémově stejně bude dle Obr 5.12 fungovat metoda měření vertikálního středu, ale výpočet se provádí s detekovaných vertikálních hran. Poloha vertikálního středu bude označena jako *VS*.



Obr 5.12 Měření vertikálního středu

5.6 Měření obsahu (komparační metoda)

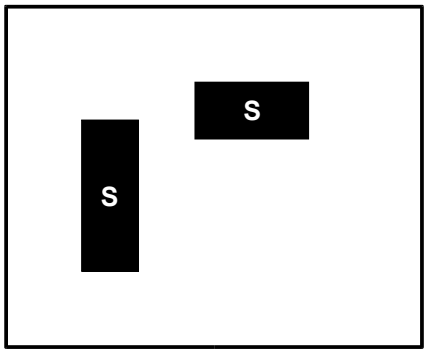
Do videopreprocesoru budou implementovány metody pro měření obsahu světlých a tmavých objektů. Bude se počítat s tím, že se v jednom snímku vyskytuje jeden nebo více objektů, ale výsledkem bude vždy součet obsahů všech objektů.



Obr 5.13 Měření obsahu světlých objektů

Měření světlých objektů viz. Obr 5.13 se provádí porovnáním jasové hodnoty pixelu s komparační konstantou. Konstanta bude označena *cmpk*. Při načtení každého pixelu z CMOS plošného senzoru se kontroluje zda hodnota překročila konstantu *cmpk* a pokud ano je do vnitřního čítače přičtena hodna +1. Na konci čtení snímku se vyčte hodnota čítače. Velikost obsahu bude dána počtem pixelů a označena jako *SB*.

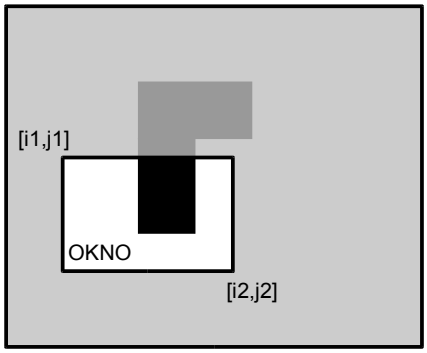
Měření obsahu tmavých objektů viz. Obr 5.14 je systematicky stejné jako měření světlých objektů, ale kontroluje se podkročení hodnoty konstanty *cmpk*.



Obr 5.14 Měření obsahu tmavých objektů

5.7 Metoda okno

Pomocí této metody se budou detekovat hrany jen ve zvolené části obrazu. Část bude vymezena obdélníkem daným souřadnicemi *[i1 j1]* a *[i2 j2]*. Uživatel bude mít možnost tyto souřadnice nastavit.



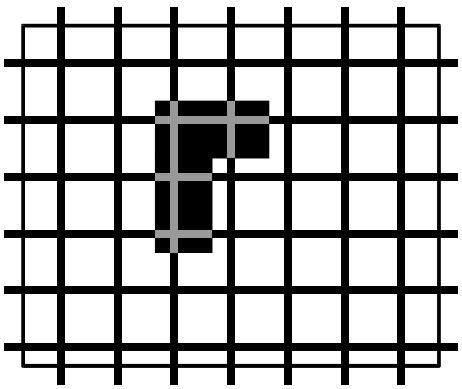
Obr 5.15 Metoda okno

K realizaci této metody budou potřeba v FPGA čítače sloupců a řádků jejichž hodnota se bude porovnávat s uloženými konstantami souřadnic.

Pomocí této metody bude mít uživatel videopreprocesoru možnost také uložit do výstupní FIFO paměti zvolenou část obrazu a nebo část obrazu zpracovanou konvolučním filtrem viz. kapitola *5.9 Konvoluční filtr*.

5.8 Metoda mříž

Pomocí této metody bude mít uživatel možnost nastavit ve kterých sloupcích a řádcích se hledají hrany. Například hledání hran v každém osmém řádku a zároveň sloupci. Výsledek bude tokový jako by se hrany hledaly v pravidelné mříži tvořené stejnými čtverci dle Obr 5.16.



Obr 5.16 Metoda mříž

5.9 Konvoluční filtr

Videopreprocesor bude obsahovat konvoluční filtr s jádrem *Mexický klobouk*, *Prewitt* a *Laplace*. Výpočet konvoluce se bude provádět z hodnot obrazové matice viz. kapitola *5.1 Obrazová matice pixelů 8x8*. Použitá konvoluční jádra jsou následující.

Mexický klobouk:

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

Laplace:

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

Previtt:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

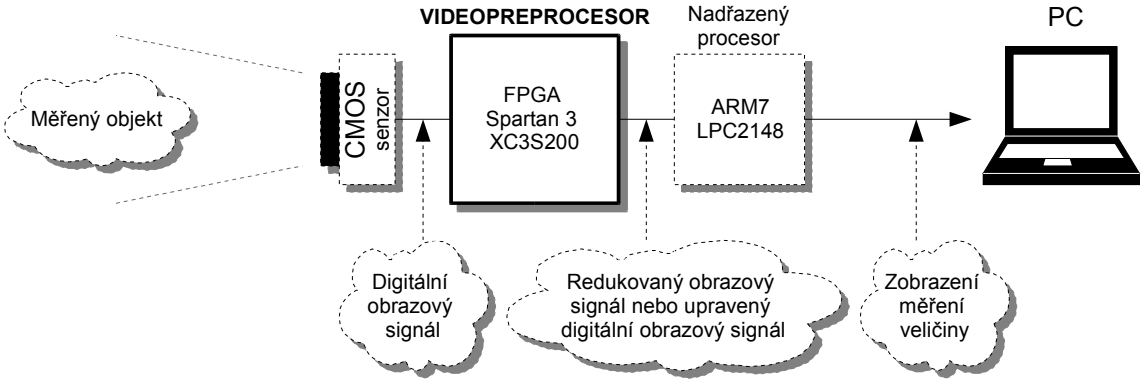
Pro realizaci konvolučního filtru se použije v obvodu FPGA kombinační logika připojená na výstupy hodnot obrazové matice.

6. Videopreprocesor – vnější popis

Tato kapitola popisuje videopreprocesor v FPGA z vnějšího uživatelského pohledu a seznamuje se všemi jeho funkcemi. Popsány jsou zde zejména periférie videopreprocesoru a jejich vstupní a výstupní signály pomocí kterých videopreprocesor komunikuje s okolními obvody.

6.1 Úvod k videopreprocesoru

Videopreprocesorem se v této práci rozumí logický obvod nahraný v hradlovém poli FPGA typu Spartan 3 XC3S200. Implementuje základní funkce potřebné při bezkontaktní měření objektů v **reálném čase**. **Vstupem** videopreprocesoru je **digitální obrazový** signál z CMOS plošného senzoru (v této práci je použit typ LM9617). **Výstupem** je za prvé **redukovaný obrazový** signál dle kapitoly *2.3 Redukovaný obrazový signál*. Za druhé videopreprocesor je možné nastavit do režimu úpravy digitálního obrazového signálu s konvoluční maskou. Místo, které zaujímá v systému systému bezkontaktního měření je dle následujícího obrázku.



Obr 6.1 Úvod k videopreprocesoru

6.2 Funkce a vlastnosti videopreprocesoru

FUNKCE:

Dle zadání diplomové práce implementuje videopreprocesor následující funkce:

- vyhledávání optických hran ve směru řádků
- určování velikosti plochy světlých a tmavých objektů
- určování středu objektu v plošném obraze
- určování horizontálního a vertikálního rozměru
- uložení vybraných částí obrazu do vnitřní paměti RAM

Navíc videopreprocesor implementuje funkce:

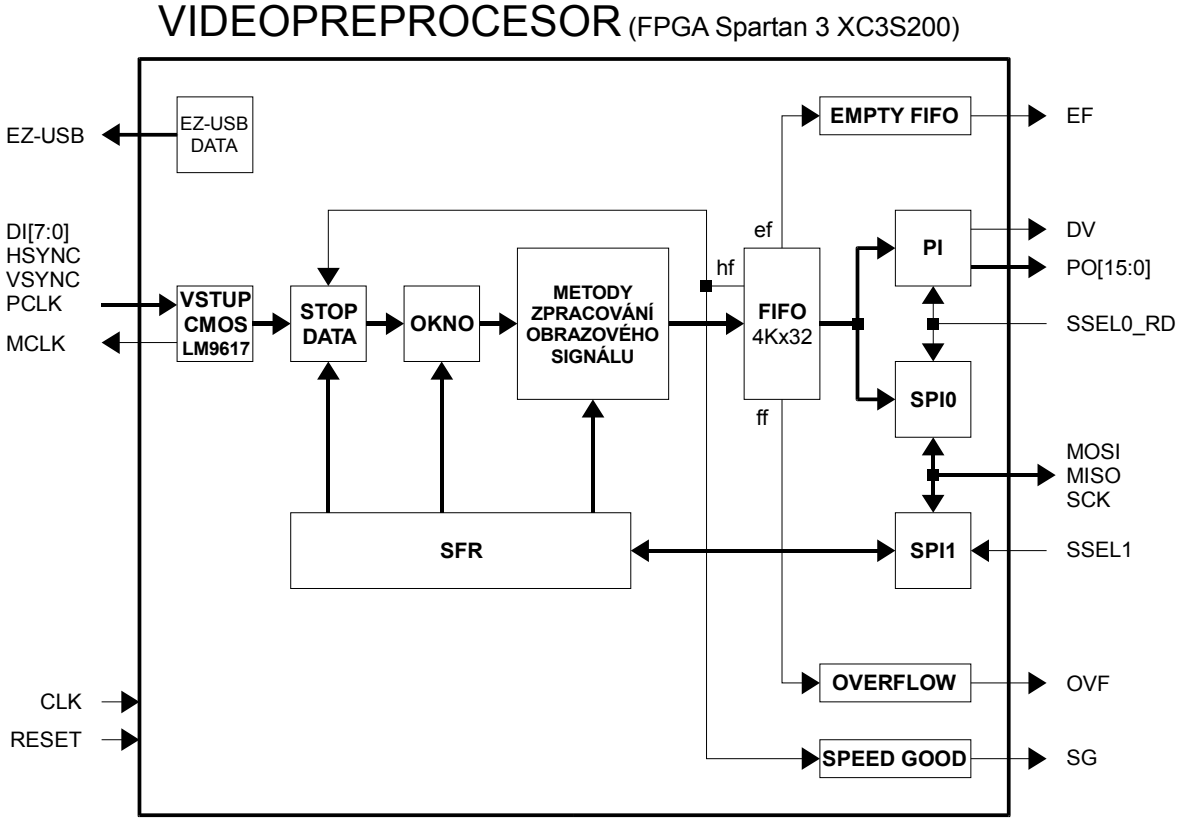
- vyhledávání optických hran ve směru sloupců
- zpracování digitálního obrazového signálu s konvoluční maskou Laplace, Prewitt, „Mexický klobouk".

VLASTNOSTI:

- nastavení funkcí videopreprocesoru rozhraním SPI
- čtení dat sériové (SPI) nebo paralelní 16 bitů (PI)
- autonomní řízení toku celistvých snímků
- připojitelnost k nadřazenému obvodu s libovolnou rychlostí čtení dat
- vyrovnávací FIFO paměť 4k32b
- pracovní frekvence 50MHz nebo větší dle možností zvoleného typu FPGA.

6.3 Blokové schéma

Videopreprocesor má blokové schéma dle Obr 6.2. Následuje základní popis významu jednotlivých bloků.



Obr 6.2 Blokové schéma videopreprocesoru

VSTUP CMOS LM9617: Vstupní rozhraní pro připojení CMOS plošného senzoru typu LM9617. Zajišťuje generování pracovního kmitočtu MCLK pro CMOS senzor a synchronizuje výstupní datové a synchronizační signály senzoru s videopreprocesorem. Lze připojit i jiný typ obrazového senzoru, ale musí být dodržen formát vstupních signálů dle kapitoly *6.4 Vstup CMOS LM9617 – připojení plošného senzoru*.

STOP DATA: Propust celistvých snímků. Tento blok dle naplnění výstupní FIFO paměti propouští nebo nepropouští snímky a čeká až nadřazený procesor vyčte zpracovaná data.

OKNO: Propouští část obrazu zvolenou uživatelem. Část obrazu se volí nastavením levého horního a pravého dolního rohu pomocí SFR.

METODY ZPRACOVÁNÍ OBRAZOVÉHO SIGNÁLU: Blok obsahuje metody pro bezkontaktní měření objektů dle kapitoly *5 Metody zpracování obrazového signálu*.

SFR: Sada speciálních funkční registrů videopreprocesoru. Jejich nastavením se volí funkce videopreprocesoru a dále tyto registry uchovávají konstanty potřebné pro metody zpracování obrazového signálu (např. komparační konstanta *cmpk* nebo diferenční konstanta *diffk*).

FIFO: Vyrovnávací paměť. Slouží k uložení vybrané části obrazu nebo nalezených hran, středu objektu a podobně.

PI: Anglicky parallel interface. Výstupní paralelní rozhraní pro přenos redukovaného obrazového signálu do nadřazenému procesoru.

SPI0: První sériové rozhraní anglicky serial peripheral interface. Výstupní rozhraní pro přenos redukovaného obrazového signálu do nadřazenému procesoru.

SPI1: Druhé sériové rozhraní. Slouží pro nastavení hodnot speciálních funkčních registrů, které určují funkci videopreprocesoru.

EMPTY FIFO: Blok příznaku prázdné FIFO paměti.

OVERFLOW: Blok příznaku přetečení FIFO paměti.

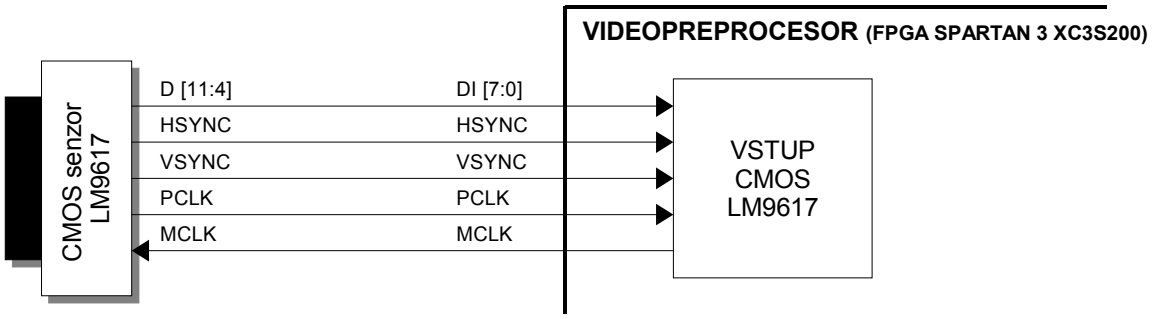
SPEED GOOD: Blok příznaku rychlého čtení výstupních dat nadřazeným procesorem.

EZ-USB DATA: Blok pro přenos obrazu do USB řadiče Cypress. Zde je naznačen orientačně. Jeho význam je popsána v kapitole *8.8 Zobrazení sledované scény pomocí rozhraní USB*.

Signály **CLK** a **RESET**: Globální hodinový signál a asynchronní reset aktivní v log. 1.

6.4 Vstup CMOS LM9617 – připojení plošného senzoru

Aby mohl videopreprocesor zpracovávat digitální obrazový signál musí se nejprve zajistit připojení CMOS plošného senzoru. Interface pro připojení senzoru zajišťuje blok VSTUP CMOS LM9617 dle následujícího obrázku.



Obr 6.3 Připojení CMOS plošného senzoru LM9617

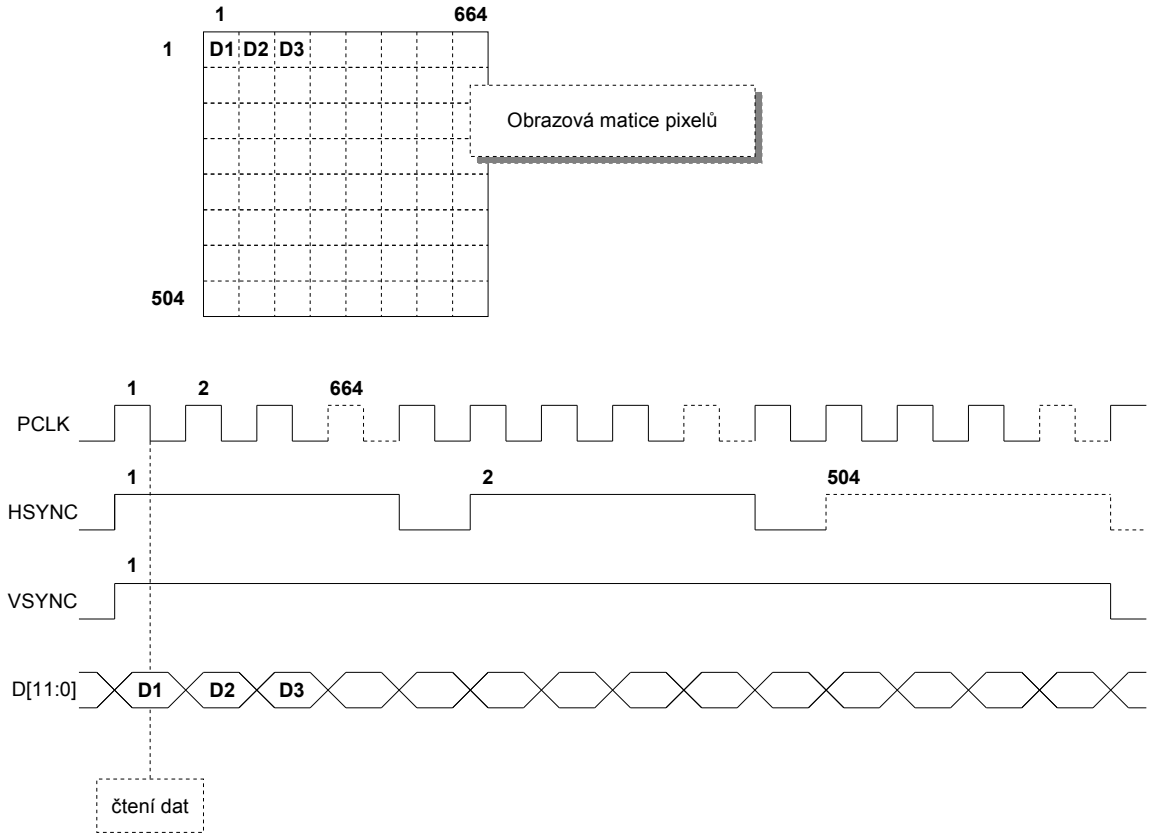
CMOS plošný senzor je elektronický obvod, který převádí jasovou obrazovou funkci na digitální obrazový signál. Tento signál je přenášen do videosenzoru pomocí signálů HSYNC, VSYNC, PCLK, D. Tyto signály lze zařadit do první skupiny – signály pro přenos obrazu. Samostatnou skupinu tvoří signál MLCK. Jde o signál pracovního kmitočtu plošného senzoru nutný pro jeho běh. Poslední skupinou jsou řídící signály pro plošný senzor. Tyto signály nastavují funkce plošného senzoru a režimy jeho práce (zde se nepoužívají).

Plošný senzor se po zapnutí napájení nastavuje automaticky sám do **výchozího režimu**. Digitální obrazový signál produkovaný senzorem v tomto režimu pro další zpracování stačí a

proto zde nebude nastavení plošného senzoru dále řešeno. Podrobnosti nastavení režimu práce plošného senzoru jsou uvedeny v (10). Na Obr 6.4 je včetně obrazové matice pixelů uveden formát digitálního obrazového signálu při výchozím režimu, na který je uzpůsoben blok VSTUP CMOS LM9617.

V zásadě lze připojit i jiné typy senzorů, ale výstupní digitální obrazový signál musí mít uvedený formát. Rozlišení se kterým vstupní blok pracuje je maximálně 1024 pixelů v řádku a 512 pixelů ve sloupci.

Signál MCLK generovaný videopreprocesorem má frekvenci **MCLK = CLK/4.** Tento signál lze pro plošný senzor generovat i jiným způsobem než videopreprocesorem. Pro senzor LM9617 musí být však dodržen rozsah pracovního kmitočtu dle katalogového listu 12MHz – 48MHz.



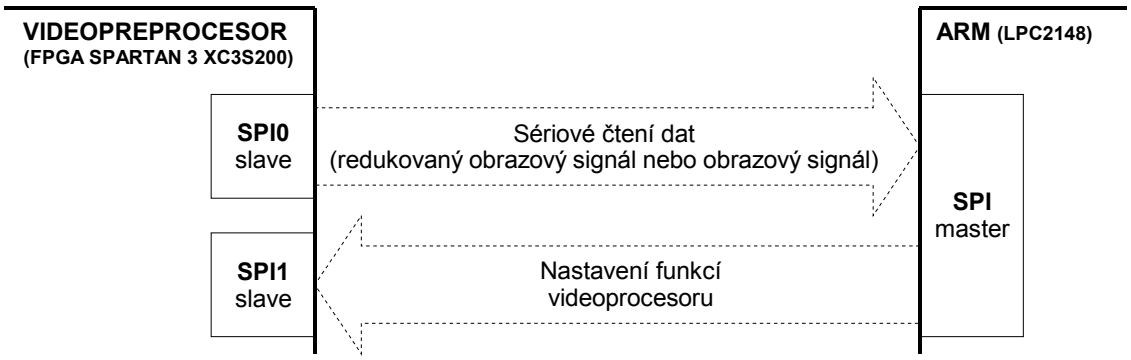
Obr 6.4 Formát digitálního obrazového signálu senzoru LM9617

6.5 Komunikace videopreprocesor – nadřazený procesor (ARM7)

Videopreprocesor umožňuje komunikovat s nadřazeným procesorem **sériově** nebo **sériově-paralelně**. Sériová komunikace je pomocí rozhraní SPI. Výhoda tohoto způsobu komunikace je minimum externích signálů a velká podpora rozhraní SPI ze strany nadřazených procesorů.

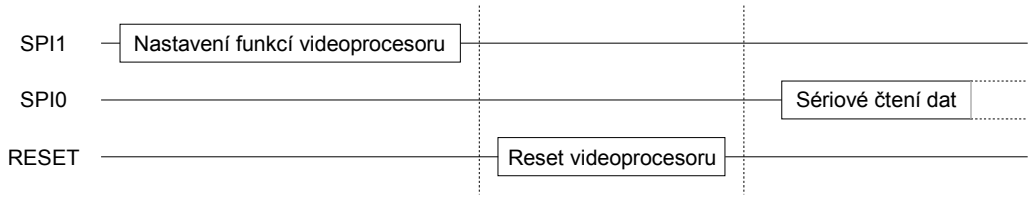
Sériově-paralelní komunikace je pomocí rozhraní SPI a PI. Výhoda tohoto způsobu komunikace je větší rychlost přenášených dat. V konečné aplikaci se používá jedna z variant komunikace.

Sériová komunikace je dle Obr 6.5. Na straně nadřazeného procesoru se používá jeden blok rozhraní SPI v režimu master. Na straně videopreprocesoru se používají dva bloky rozhraní SPI v režimu slave. Bloky mají dle Obr 6.2 společné signály MOSI, MISO, SCK a jejich výběr je pomocí signálů SSEL0_RD a SSEL1.



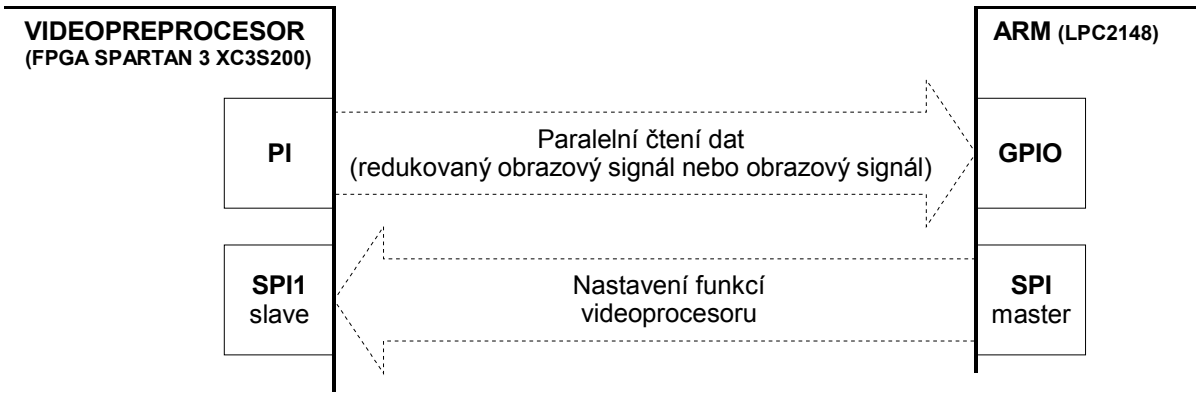
Obr 6.5 Sériová komunikace videopreprocesor - nadřazený procesor

Při sériové komunikaci musí být operace, které probíhají na sběrnici v přesném pořadí dle Obr 6.6. Z obrázku vyplývá velmi důležitá vlastnost. **Signál RESET videopreprocesoru nenuluje nastavení jeho funkcí.** Videopreprocesor se uvede v činnost nejprve nestavení jeho funkce například na hledání hran objektů, dále se provede reset videopreprocesoru a následně se pozice hran mohou z videopreprocesoru vyčítat. Tento proces se může opakovat libovolně krát za sebou.

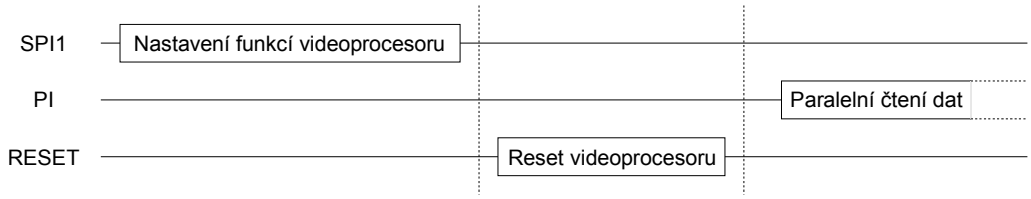


Obr 6.6 Časový diagram sériová komunikace s videopreprocesorem

Sériově-paralelní komunikace je dle Obr 6.7. Na straně nadřazeného procesoru se používá jeden blok rozhraní SPI v režimu master a pro paralelní čtení dat uživatelské GPIO piny. Na straně videopreprocesoru se používá jeden blok rozhraní SPI v režimu slave a rozhraní PI. Dále komunikace probíhá dle Obr 6.8 obdobně jako při sériové komunikaci .



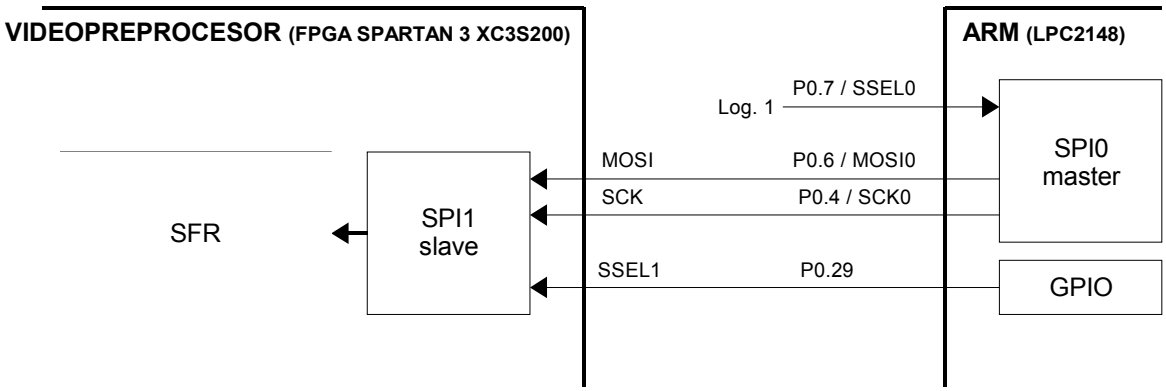
Obr 6.7 Sériově-paralelní komunikace videopreprocesor - nadřazený procesor



Obr 6.8 Časový diagram sériově-paralelní komunikace s videopreprocesorem

6.6 Nastavení funkcí videopreprocesoru

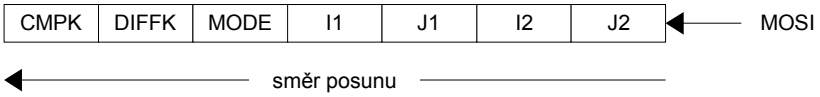
Nastavením funkcí videopreprocesoru se rozumí naprogramování hodnot speciálních funkčních registrů (SFR) pomocí rozhraní SPI1. Dle jejich hodnoty pak videopreprocesor provádí například hledání hran, měření plochy atd. Propojení všech signálů SPI1 videopreprocesoru s nadřazeným procesorem je dle následujícího obrázku.



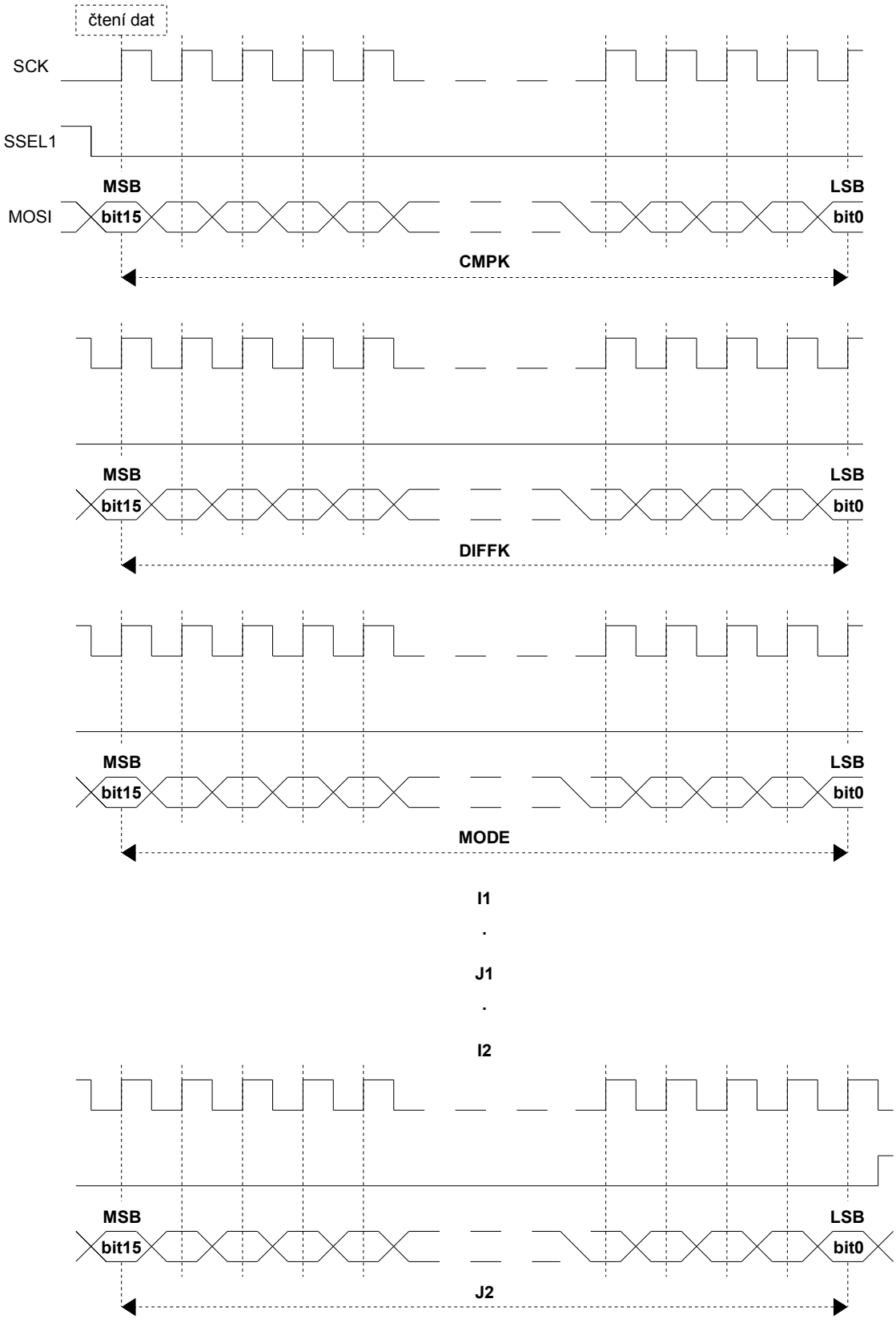
Obr 6.9 Propojení SPI1 s nadřazeným procesorem (ARM7 LPC2148)

SPI1 videopreprocesoru pracuje v režimu „nulová polarita“ a „nulový fázový posuv“ signálů SCK a MOSI (případně MISO). V nadřazených procesorech bývá tento mód nastaven bity CPHA = 0 a CPOL = 0.

SFR tvoří sadu sedmi šestnáctibitových registrů, které jsou spolu spojeny do série a celkově tvoří jeden velký posuvný registr se sériovým vstupem MOSI a paralelním přístupem k hodnotám jednotlivých registrů ze strany videopreprocesoru. Jejich pořadí a směr posuvu je dle Obr 6.10. Naprogramování všech registrů SFR se provádí v jednom cyklu během nestavení SSEL1 do log. 0. Formát přenášených dat na sběrnici SPI musí být dle Obr 6.11.



Obr 6.10 Pořadí speciálních funkčních registrů



Obr 6.11 Formát dat na sběrnici SPI při programování SFR

Tabulky, které definují význam nastavení SFR jsou uvedeny v příloze A.

Zde je ukázka kódu v jazyce C do procesoru ARM7 LPC2148, který dle předchozího popisu nastaví hodnoty všech sedmi registrů rozhraním SPI1. Kompletní kód se nachází na doprovodném CD v adresáři *ARM_code\Start_ARM\Blinky.Uv2*. Uvedená ukázka kódu nastaví videopreprocesor do módu uložení vybrané části obrazu s oknem o velikosti jedno řádku [100, 1] [100,664].

```
//nastav 0 na P0.29 (SSEL1 SPI1)
IOCLR0 |= 0x20000000;
write_spi(0x00);//cmpk
write_spi(0x00);
write_spi(0x00);//diffk
write_spi(0x00);
write_spi(0x00);//mode
write_spi(0x01);
write_spi(0x00);//i1
write_spi(0x64);
write_spi(0x00);//j1
write_spi(0x01);
write_spi(0x00);//i2
write_spi(0x64);
write_spi(0x02);//j2
write_spi(0x98);
//nastav 1 na P0.29 (SSEL1 SPI1)
IOSET0 |= 0x20000000;
```

Kód 6.1: Nastavení funkcí videopreprocesoru rozhraním SPI1

Funkce pro inicializaci SPI a funkce pro čtení a zápis na SPI, která se používá v předešlém kódu jsou ukázány v následujícím kódu.

```
// SPI rozhrani
void init_spi (void){            //inicializece bloku SPI
    PINSEL0 |= 0x5500;           //povoleni SCK, MISO, MOSI, SSEL
    S0SPCR = 0x20;               //nastaveni MASTER MODE SELECT

    S0SPCCR = 0x1E;              //nastaveni frekvence SCK
}

void write_spi (unsigned int ch){//zapis byte na SPI
    S0SPDR = ch;                 //byte do registru S0SPDR
    while(!(S0SPSR & SPIF));     //cekani na odeslani byte
}

unsigned int read_spi (void){          //cteni byte z SPI
    while(!(S0SPSR & SPIF));     //cekani na prijem byte
    return(S0SPDR);
}
```

Kód 6.2: Inicializace SPI a funkce pro zápis a čtení z SPI

6.7 Formát výstupních dat

Výstupní data tedy hodnoty zpracovaného obrazového signálu se z videopreprocesoru vyčítají ve třiceti dvou bitových rámcích. Z tohoto počtu bitů je odvozen i rozměr výstupní FIFO paměti která je 4k32 (přesně 4096 krát 32 bitů).

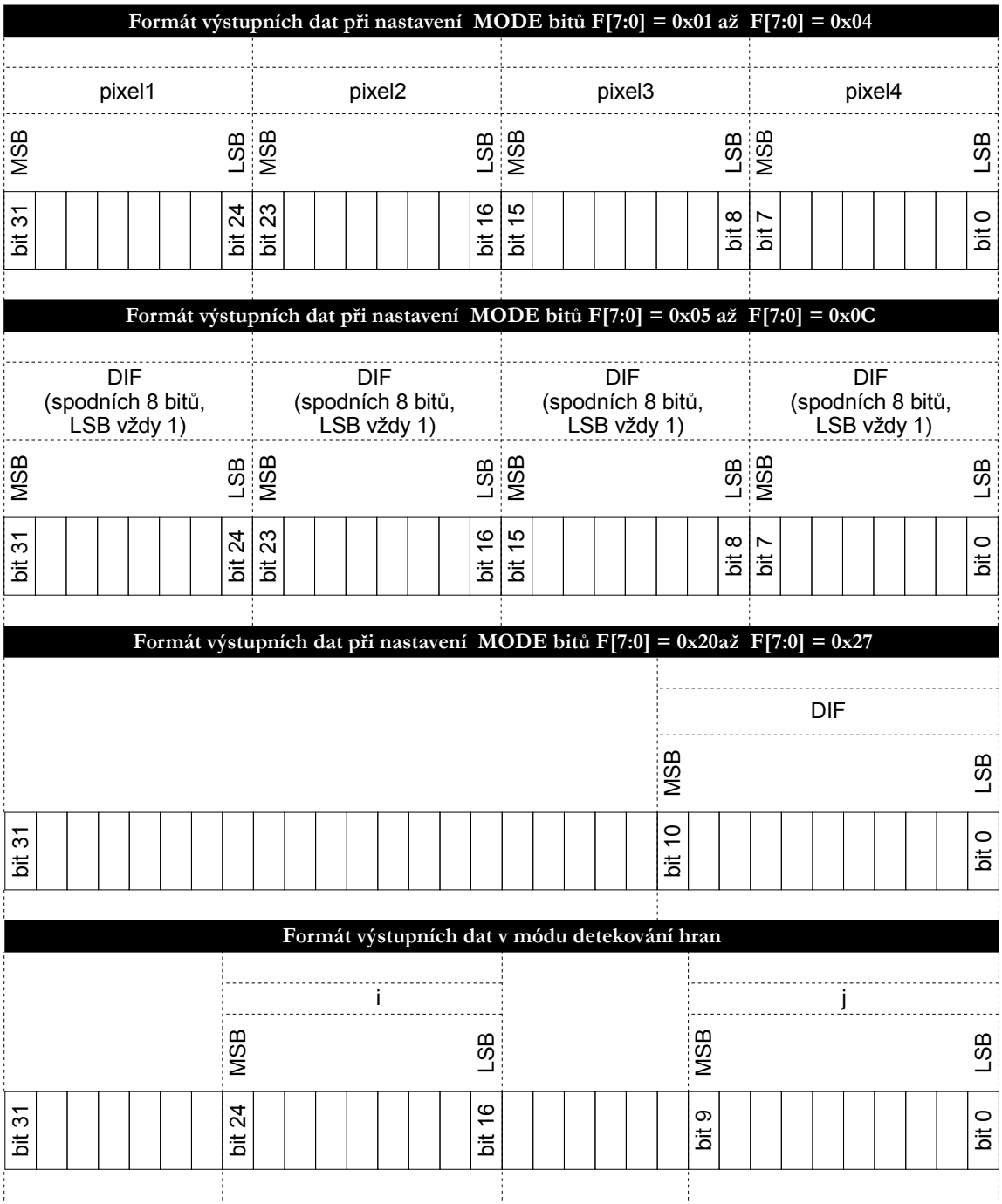
Počet bitů 32 byl zvolen na základě požadavku kódovat pozici hrany dané souřadnicí [i, j]. Při rozlišení obrazu 664 krát 504 je teoreticky maximální velikost souřadnice [504, 664]. Číslo 504 lze zakódovat do devíti bitů a číslo 664 do desíti bitů.

Protože přenos hran se bude po sběrnici SPI přenášet po Bytech (8 bitů) a po sériové lince do PC taky, bude pro zpracování dat nadřazeným procesorem lepší, když počet bitů pro kódování výstupních dat bude násobkem osmi. Protože je zde však ještě šestnácti bitová paralelní sběrnice PI je nejlepší kódovat výstupní data počtem bitů který je násobek šestnácti.

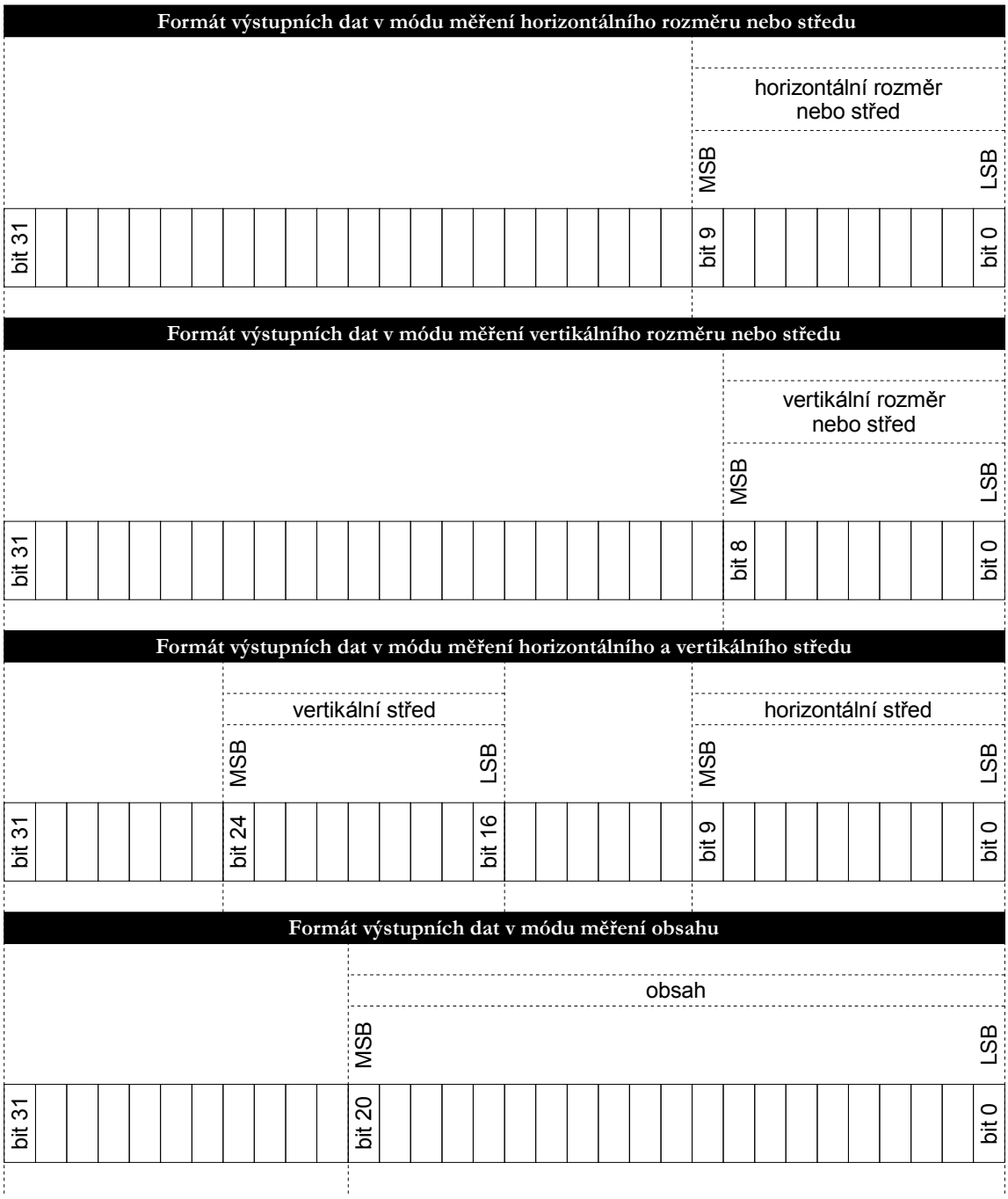
Čísla i a j jsou videopreprocesorem držena pospolu jako jeden paralelní signál až do doby vyčtení nadřazeným procesorem. S dodržením pravidla z minulého odstavce vyplývá třiceti dvou bitové kódování souřadnice [i, j].

V případě, že je videopreprocesor nastaven do režimu ukládání obrazových osmi bitových dat, jsou tyto data rozložena do třiceti dvou bitového rámce po čtveřicích.

Následují obrázky s formátem výstupních dat pro příslušný výběr metody zpracování obrazového signálu registrem MODE bity F[7:0].



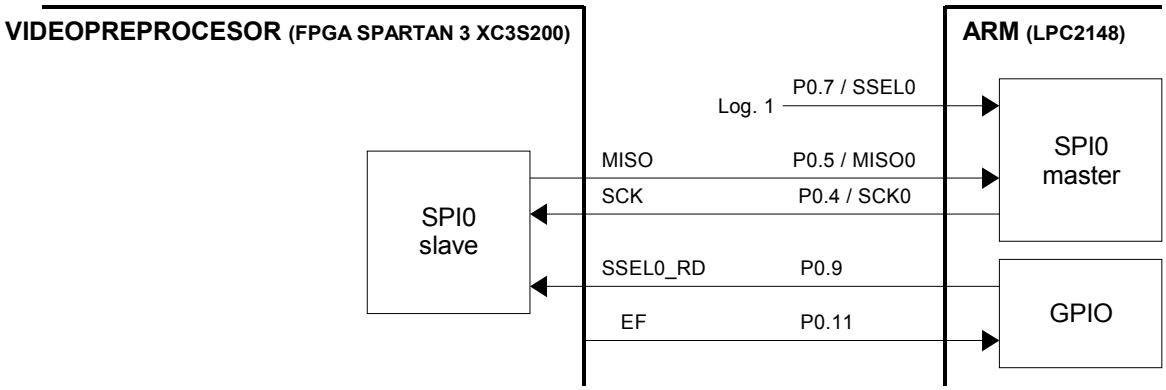
Obr 6.12 Formát výstupních dat – první část



Obr 6.13 Formát výstupních dat – druhá část

6.8 Sériové čtení dat (SPI0)

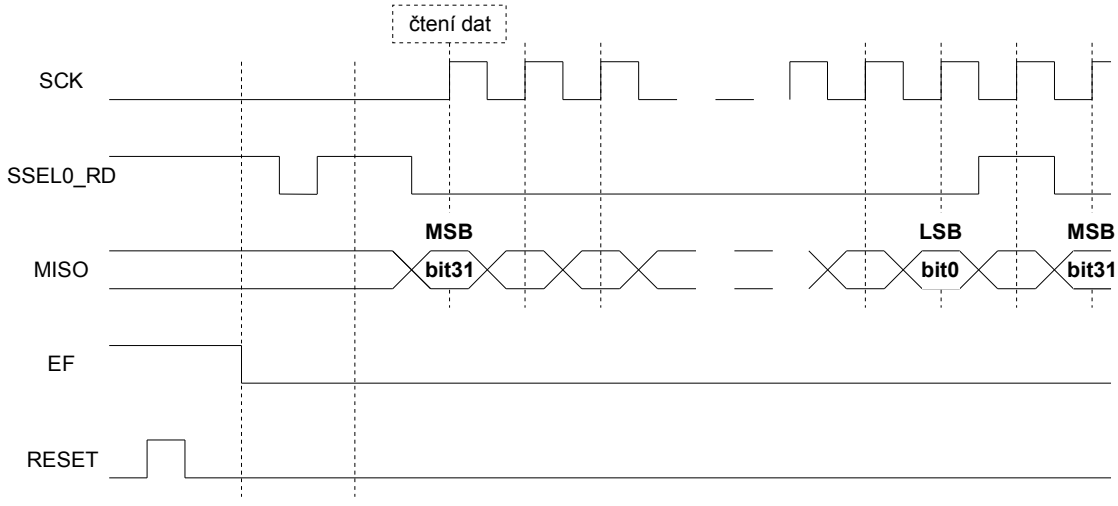
Sériové čtení dat z videopreprocesoru je pomocí rozhraní SPI0. Připojení k nadřazenému procesoru je v režimu sériového čtení dle následujícího obrázku.



Obr 6.14 Propojení SPI0 s nadřazeným procesorem (ARM7 LPC2148)

Stejně jako SPI1 také SPI0 videopreprocesoru pracuje v režimu „nulová polarita“ a „nulový fázový posuv“ signálů SCK a MOSI (případně MISO). V nadřazených procesorech bývá tento mód nastaven bity CPHA = 0 a CPOL = 0.

Poté, co proběhne nastavení funkcí videopreprocesoru a jeho reset, kontroluje se nadřazeným procesorem signál EF (empty fifo). Přechod EF z '1' do '0' značí, že ve výstupní paměti FIFO jsou již připravená data. Nyní musí proběhnou jeden impuls do '0' o periodě větší ne 1/CLK na signálu SSEL0_RD a následně mohou být kontinuálně čteny třiceti dvou bitové rámce již bez tohoto impulsu. Během čtení jednoho rámce je signál SSEL0_RD v '0' a po přečtení rámce se musí vracet do '1'. Časový diagram sériového čtení dat přes SPI0 je dle Obr 6.15.



Obr 6.15 Sériové čtení dat (SPI0)

Následuje ukázka kódu do procesoru AMR7 LPC2148 sériového čtení dat rozhraním SPI0. Kompletní kód se nachází na doprovodném CD v adresáři *ARM_code\Start_ARM\Blinky.Uv2*. Uvedený kód nejprve provede reset videopreprocesoru a následuje impuls na SSEL0_RD dle předchozího popisu. Poté, co je po sériové lince přijat znak 'r' (ready), uvedený kód čte sto šedesátkrát šestkrát za sebou třiceti dvou bitový rámec a zároveň ho odesílá po sériové lince do počítače. Toto se opakuje v nekonečné smyčce.

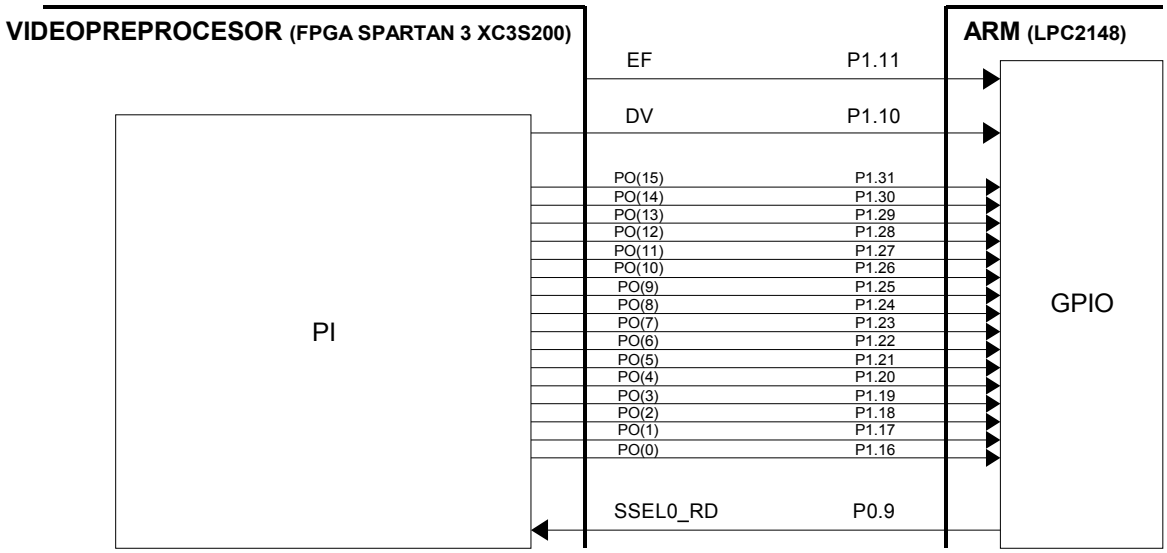
Uvedenému kódu 6.3 předcházelo nastavení funkcí videopreprocesoru dle ukázky kódu 6.1. Jedná se o část programu, která odesílá do počítače obrazová data z jednoho řádku, kde jsou následně zobrazeny.

```
//nastav 1 na P0.13 (RESET)
IOSET0 |= 0x00002000;
//nastav 0 na P0.13 (RESET)
IOCLR0 |= 0x00002000;

//cekej na P0.11 = 0 (EF)
while ((IOPIN0 & 0x00000800)){};
//nastav 0 na P0.9 (SSEL0_RD)
IOCLR0 |= 0x00000200;
//nastav 1 na P0.9 (SSEL0_RD)
IOSET0 |= 0x00000200;

//cteni radku pres SPI
if (answer == 'z'){
    while(1){
          count = 1;
          while(getchar()!='r');
          while(count <= 166){
                count ++;

                //cekej na P0.11 = 0 (EF)
                while ((IOPIN0 & 0x00000800)){};
                //nastav 0 na P0.9 (SSEL0_RD)
                IOCLR0 |= 0x00000200;
                write_spi(0x00);//dummy byte
                putchar(read_spi());
                write_spi(0x00);//dummy byte
                putchar(read_spi());
                write_spi(0x00);//dummy byte
                putchar(read_spi());
                write_spi(0x00);//dummy byte
                putchar(read_spi());
                //nastav 1 na P0.9 (SSEL0_RD)
                IOSET0 |= 0x00000200;
          }
    }
}
```

Kód 6.3: Sériové čtení třiceti dvou bitového rámce z SPI0

6.9 Paralelní čtení dat (PI)

Paralelní čtení dat z videopreprocesoru je pomocí rozhraní PI (parallel interface). Připojení k nadřazenému procesoru je v režimu paralelního čtení dle následujícího obrázku.

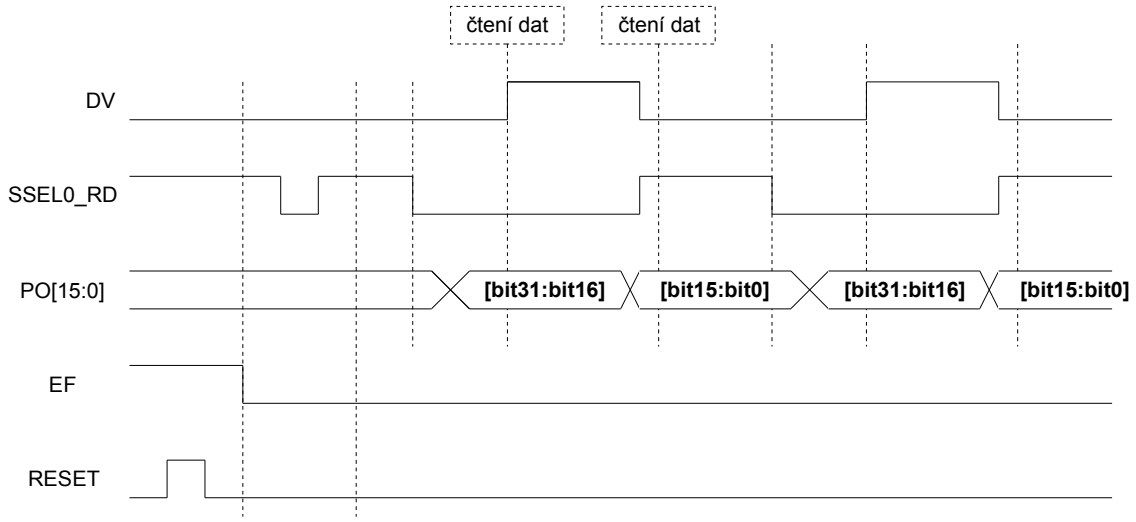


Obr 6.16 Propojení PI s nadřazeným procesorem (ARM7 LPC2148)

Způsob paralelního čtení dat je na začátku podobný sériovému čtení. Tedy reset videopreprocesoru, čekání na signál EF (přechod do '0'), impuls na SSEL0_RD a následné čtení třiceti dvou bitových rámců. Rámce se čtou ve dvou krocích. Nejprve horních 16 bitů [31:16] a následně dolních 16 bitů [15:0].

Na začátku čtení rámce nadřazený procesor nastaví signál SSEL0_RD do '0' a čeká na DV (data valid) až přejde do '1'. Poté se může přečíst horních 16 bitů. Přechodem signálu DV z '1' do '0' se může přečíst dolních 16 bitů, nyní bez kontroly signálu DV. Časový diagram paralelního čtení dat je dle obrázku Obr 6.17.

Důvod, proč nemusí být v druhém kroku kontrolován signál DV, je následující. Třiceti dvou bitový rámec je multiplexován na šestnáctibitový výstup bloku PI. Multiplexor je řízen signálem SSEL0_RD. V prvním kroku se rámec musel načíst do výstupního registru, který je před multiplexorem. Načtení do registru je proces závislý na signálu CLK a tedy muselo se čekat na signál DV. Změnou úrovně na SSEL0_RD se na výstupu multiplexoru objeví dolních šestnáct bitů. Tento proces není již na CLK závislý a proto může být dolních šestnáct bitů rovnou čteno. Nepředpokládá se přitom, že zpoždění dolních šestnáct bitů je větší než perioda, s kterou vykonává instrukce nadřazený procesor.



Obr 6.17 Paralelní čtení dat (PI)

Následuje ukázka kódu do procesoru AMR7 LPC2148 paralelního čtení dat rozhraním PI. Kompletní kód se nachází na doprovodném CD v adresáři *ARM_code\Start_ARM\Blinky.Uv2*. Funkce programu je obdobná jako již popsaná při sériovém čtení.

```
//nastav 1 na P0.13 (RESET)
IOSET0 |= 0x00002000;
//nastav 0 na P0.13 (RESET)
IOCLR0 |= 0x00002000;

//cekej na P0.11 = 0 (EF)
while ((IOPIN0 & 0x00000800)){};
//nastav 0 na P0.9 (SSEL0_RD)
IOCLR0 |= 0x00000200;
//nastav 1 na P0.9 (SSEL0_RD)
IOSET0 |= 0x00000200;

//cteni radku pres paralelni rozhrani
if (answer == 'y'){
    while(1){
          count = 1;
          while(getchar()!='r');
          while(count <= 166){
                count ++;

                //cekej na P0.11 = 0 (EF)
                while ((IOPIN0 & 0x00000800)){};
                //nastav 0 na P0.9 (SSEL0_RD)
                IOCLR0 |= 0x00000200;
                //cekej na P0.10 = 1 (DV)
                while (!(IOPIN0 & 0x00000400)){};
                dpi32 = IOPIN1;
                dpi32 = dpi32 >> 24;
                dpi8  = dpi32;
                putchar(dpi8);
                dpi32 = IOPIN1;
                dpi32 = dpi32 >> 16;
                dpi32 =     dpi32 & 0x000000FF;
                dpi8  = dpi32;
                putchar(dpi8);
                //nastav 1 na P0.9 (SSEL0_RD)
                IOSET0 |= 0x00000200;
                dpi32 = IOPIN1;
                dpi32 = dpi32 >> 24;
                dpi8  = dpi32;
                putchar(dpi8);
                dpi32 = IOPIN1;
                dpi32 = dpi32 >> 16;
                dpi32 =     dpi32 & 0x000000FF;
                dpi8  = dpi32;
                putchar(dpi8);
          }
    }
}
```

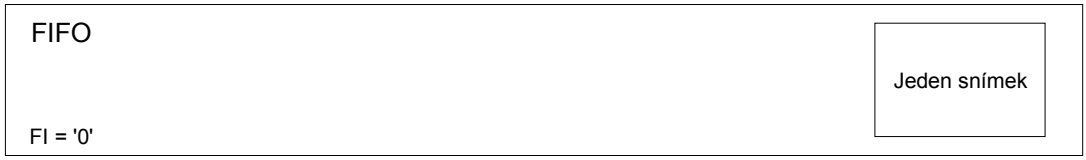
Kód 6.4: Paralelní čtení třiceti dvou bitového rámce z PI

6.10 Blok STOP DATA

Blok STOP DATA je propust celistvých snímků. Rozhodování o tom zda nový snímek projde

tímto blokem nebo ne se děje těsně před začátkem jeho čtení v době, kdy signál HSYNC = '0' a VSYNC = '0' viz. Obr 6.4. Podmínka průchodu snímku je dána stavem naplnění FIFO paměti a režimem, ve kterým tento blok pracuje. Režimy jsou dva.

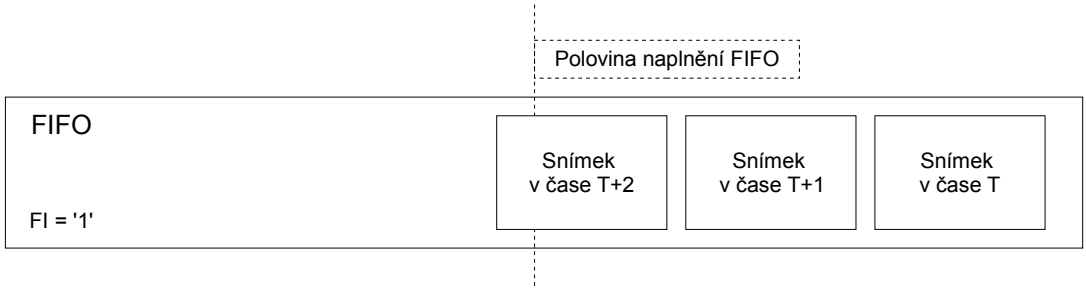
První režim se volí nastavením bitu FI = '0' registru MODE. Při tomto nastavení se do FIFO paměti načte vždy jeden snímek a čeká se na jeho úplně vyčtení nadřazeným procesorem. Snímek může být tvořen obrazovými daty nebo pouze hranami nebo jen středem nebo rozměrem objektu. Výhodou tohoto režimu je, že vyčítaný snímek je nejvíce aktuální. Tento režim je znázorně na následujícím obrázku.



Obr 6.18 Režim načítání jednoho snímku do FIFO paměti

Druhý režim se volí nastavením bitu FI = '1' registru MODE. Při tomto nastavení se do FIFO paměti načte několik snímků vyčtených z CMOS senzoru jdoucích za sebou. Načítání do FIFO probíhá do té doby než se naplní do jedné poloviny své kapacity. Přitom se vždy nechá doběhnou načítání celistvého snímku.

Protože se nechává doběhnout načítání celistvého snímku znamená to, že kdyby byla vybrána část obrazu tak, že zabírá celou kapacitu FIFO, může být bez problémů načtena do FIFO celá. Tento režim je znázorně na Obr 6.19.



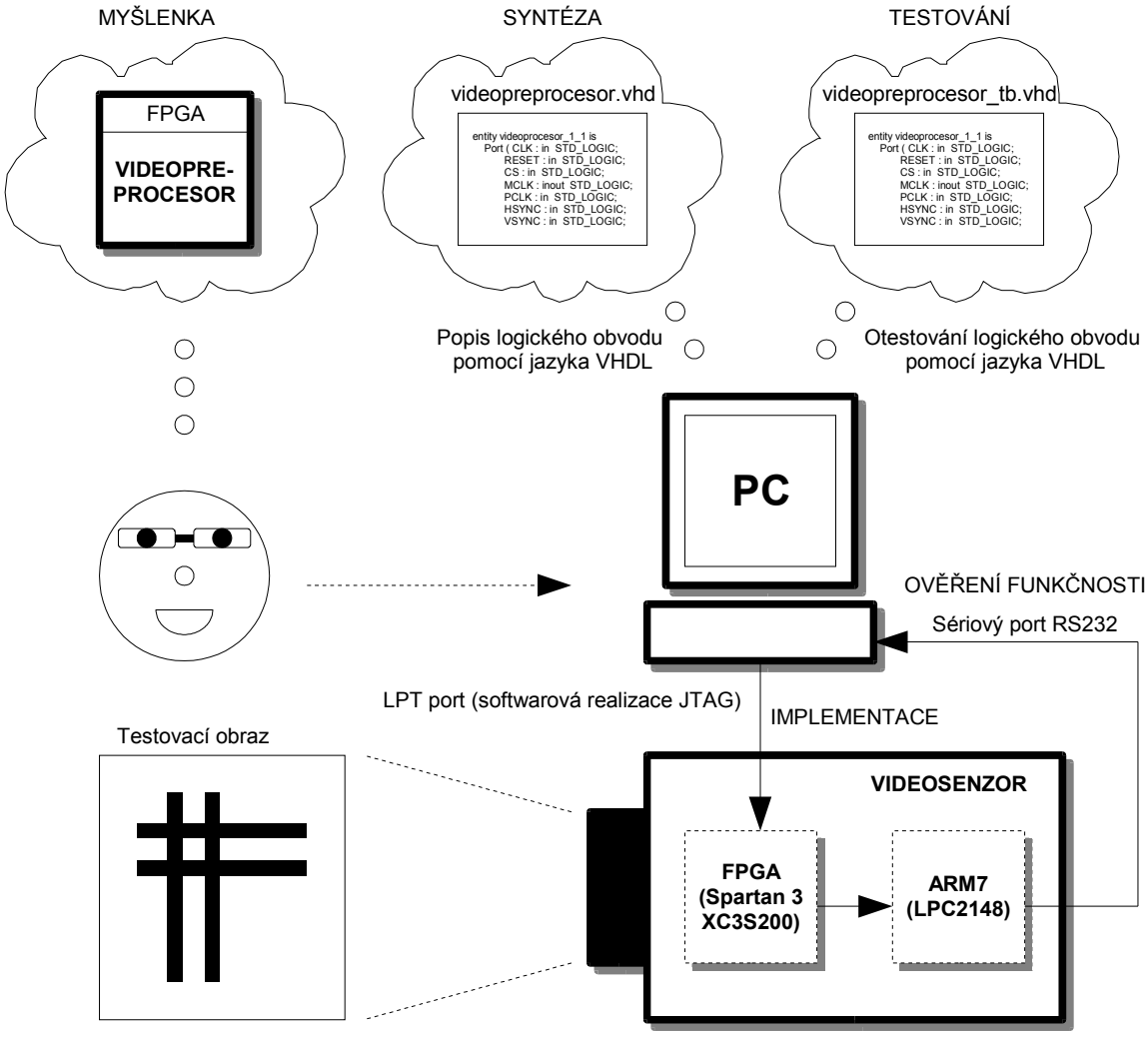
Obr 6.19 Režim načítání několika po sobě jdoucích snímku do FIFO paměti

6.11 Blok OKNO

Blok okno plní funkci dle kapitoly *5.7 Metoda okno.* jeho nastavení je dle přílohy *A.*

7. Videopreprocesor – vnitřní popis (VHDL)

Tato kapitola popisuje videopreprocesor z hlediska vnitřní struktury (propojení registrů, hradel, blokových RAM atd.). Vnitřní struktura videopreprocesoru v hradlovém poli byla popsána jazykem VHDL (VHSIC Hardware Description language). Návrh probíhal v pěti základních krocích podle následujícího obrázku. MYŠLENKA – SYNTÉZA – TESTOVÁNÍ – IMPLEMENTACE – OVĚŘENÍ FUNKČNOSTI.



Obr 7.1 Postup při návrhu videopreprocesoru

MYŠLENKA: Tento krok probíhal za pomocí tužky a papíru. Nejprve se struktura videopreprocesoru rozkreslila na jednotlivé menší bloky (např. rozhraní SPI, blok vstupu CMOS senzoru atd.) a řešilo se jaké vstupní a výstupní signály musí tyto bloky mít, aby spolu mohly být propojeny. Následovalo podrobnější rozkreslení na jednotlivé registry a hradla.

SYNTÉZA: V tomto kroku se nakreslená schémata převedla na text do jazyka VHDL. V prostředí WebPack (viz. kapitola *3.11 Programové vybavení pro vývoj aplikací*) se spustila syntéza napsaného kódu a ověřilo se tím, že je možné z napsaného VHDL textu sestavit strukturu z hradel a registrů.

TESTOVÁNÍ: Kód VHDL, který prošel úspěšně syntézou byl otestován pomocí souboru *TestBench* napsaném také v tomto jazyku. Zde se testovala logická funkčnost navržené struktury.

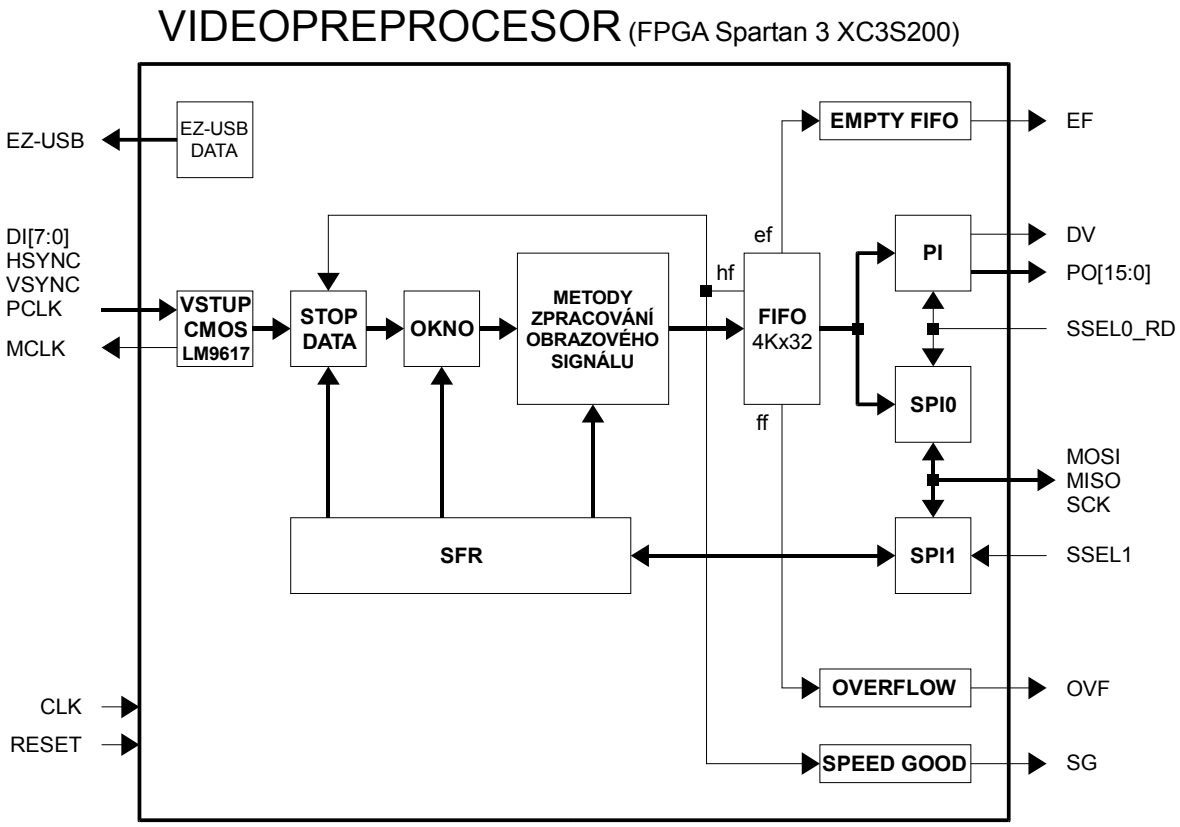
IMPLEMENTACE: Při tomto kroku se navržená struktura videopreprocesoru implementovala do hradlového pole Spartan 3 XC3S200. Implementací se rozumí sestavení logické struktury pomocí hradel registrů a blokových RAM obsažený ve cílovém obvodu FPGA a nahrání do tohoto obvodu.

OVĚŘENÍ FUNKČNOSTI: V poslední fázi se ověřovala funkčnost navrženého videopreprocesoru. Funkčnost metod zpracování obrazového signálu (detekce hran atd.) byla ověřeny na testovacím černobílém obrazci, na který byl zaostřen videosenzor. Funkčnost rozhraní videosenzoru (SPI, PI atd.) byla ověřena pomocí procesoru ARM7, ke kterému je videopreprocesor připojen (nejprve se odesílaly přednastavené konstanty později naměřené hodnoty pozic hran atd.). Naměřené hodnoty byly posílány po sériové lince do PC, kde byly zobrazeny na terminálu sériové linky nebo v MATLABU viz. kapitola *8.7 Zobrazení naměřených dat*.

VHDL kód videopreprocesoru byl rozdělen do dvou souborů. Hlavní soubor má název ***Videoprocesor.vhd*** obsahuje popis všech metod a periférií. Zvlášť byl uložen popis výstupní FIFO paměti. Její VHDL kód se nachází v souboru ***fifo_4kx32.vhd.*** Soubory jsou uloženy na doprovodném CD:

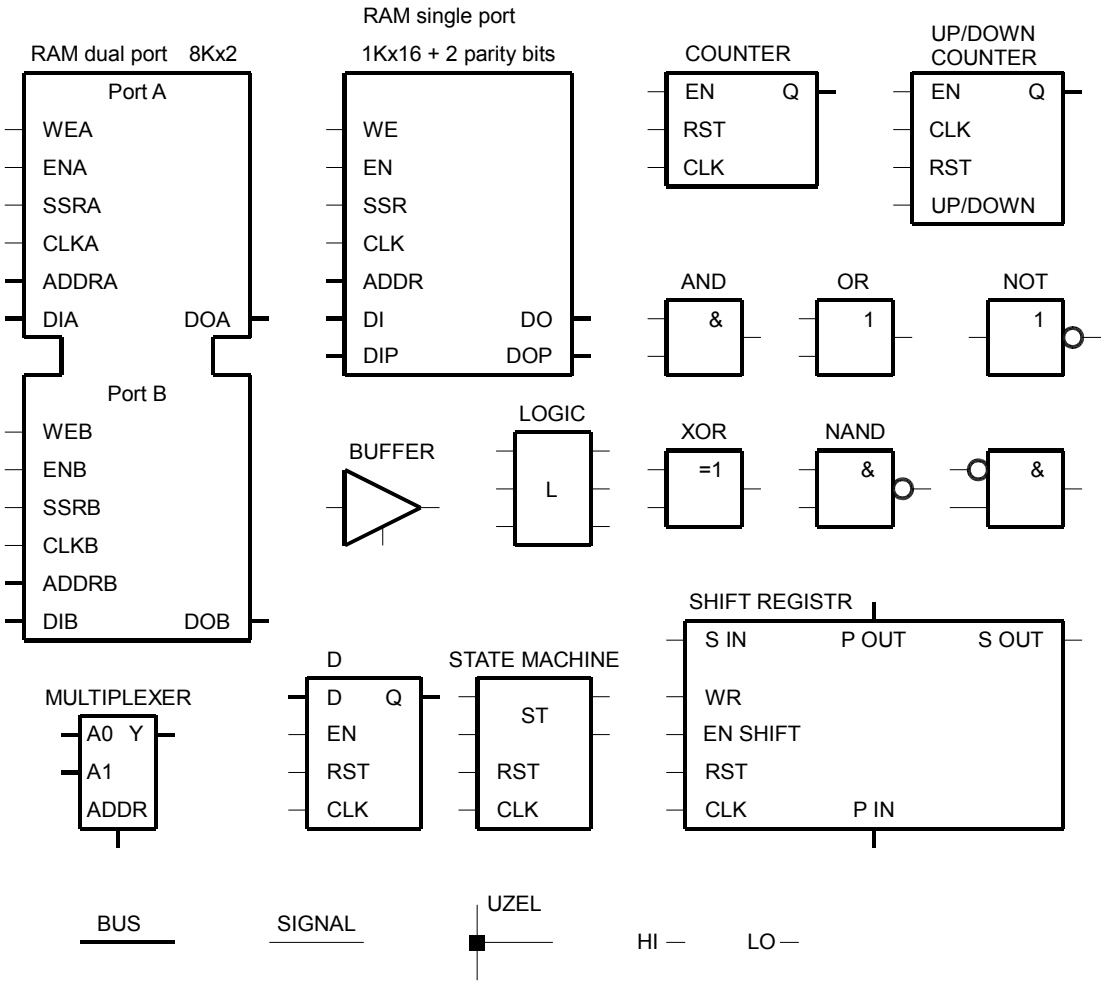
FPGA_code\ISE_WebPack_7_1i\Videoprocesor

Struktura videopreprocesoru byla rozdělena do několika samostatných funkčních bloků dle Obr 7.2. Každý blok je většinou zapsán v samostatném procesu. Například jeden proces pro rozhraní SPI0, jeden pro SPI1 jeden pro paralelní rozhraní PI atd. Jednotlivé bloky jsou vyznačeny jejich názvem zapsaném do VHDL textu jako komentář.



Obr 7.2 Videopreprocesor – blokové schéma

Následující kapitoly popisují videopreprocesor pomocí blokových schémat poskládaných z logických obvodů. Všechny značky logických obvodů použitých ve schématech jsou uvedeny na následujícím obrázku.



Obr 7.3 Schématické značky logických obvodů

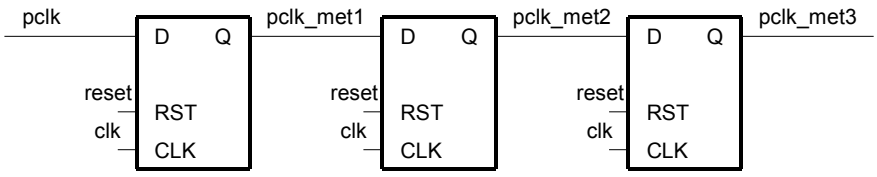
7.1 Metastabilita

Videopreprocesor v FPGA pracuje na jednom hodinovém kmitočtu *clk*, se kterým jsou synchronní všechny vnitřní signály videopreprocesoru. Signály, které do videopreprocesoru přicházejí z vnějšku (například signály z CMOS senzoru) synchronní s *clk* nejsou a musí se nejprve zajistit jejich synchronizace. Ta se provádí zařazením registrů, které vzorkují vstupní signály s frekvencí *clk*. Hrozí zde ale problém s metastabilitou těchto registrů.

Dle (15, s. 2) je možno říci, že metastabilita registru je jejich neschopnost ustálit se na definované logické úrovni v přesně definovaném čase. Příčinou metastabilního chování registru je porušení jeho časových parametrů zejména předstihu (setup time) a přesahu (hold time).

Aby se snížilo riziko vzniku metastability, řadí se na vstupní signály více registrů za sebe. Teoreticky by takových registrů muselo být pro vyloučení metastability nekonečno. V praxi se ukazuje, že je postačující zařazení tří registrů za sebe.

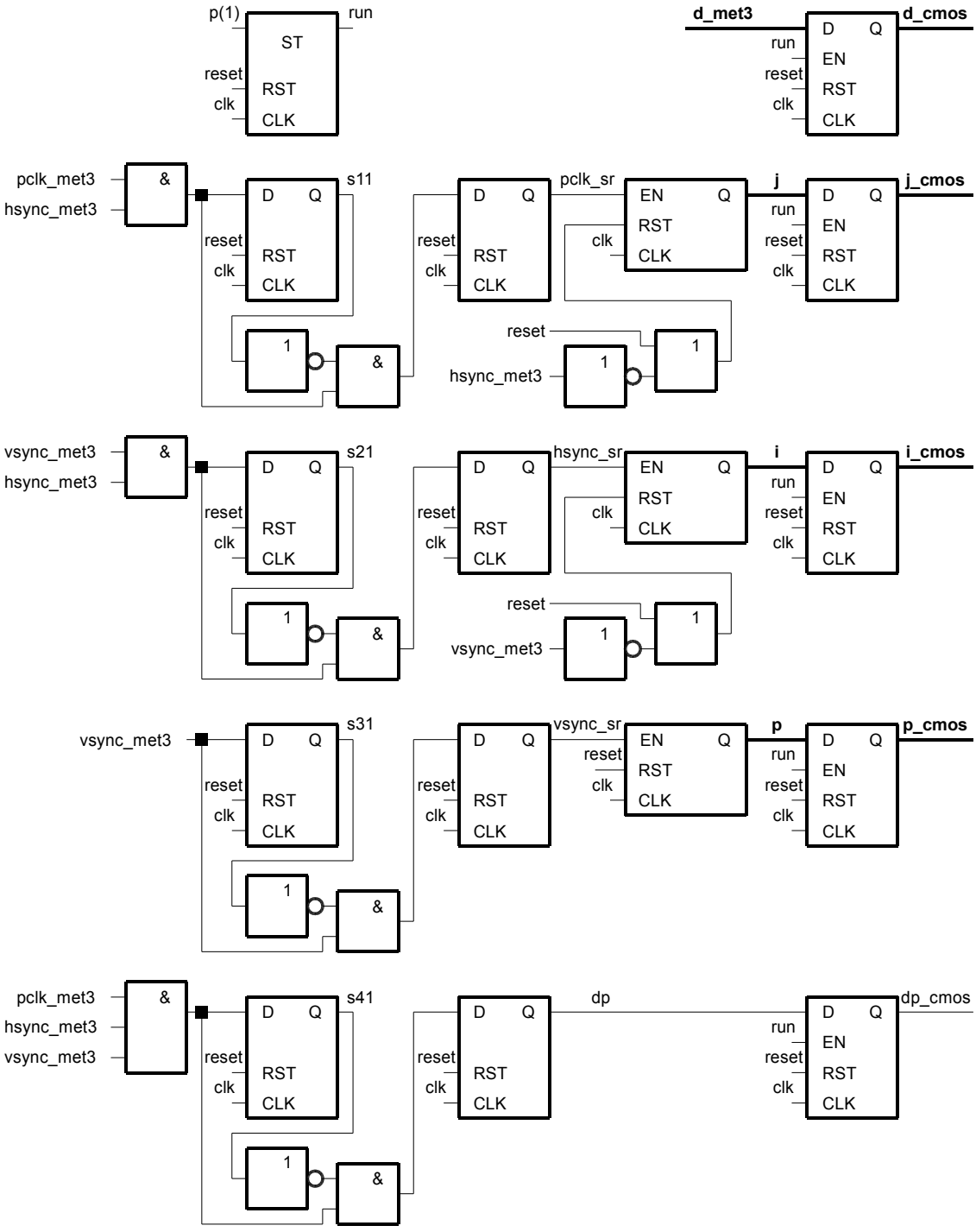
U videopreprocesoru byly při potlačení metastability pro vstupní signály z CMOS plošného senzoru použity tři registry za sebou kvůli jejich nejvyšší frekvenci a pro rozhraní SPI použit jeden registr. Ukázka zapojení registrů pro vstupní signál z CMOS senzoru je na dle následujícího obrázku



Obr 7.4 Synchronizace vstupních signálů

7.2 Vstup CMOS LM9617

Poté co signály z CMOS plošného senzoru projdou synchronizačními registry jsou dále přivedeny na vstup bloku s názvem vstup CMOS LM9617. Jeho hlavní funkcí je převést digitální obrazový signál na signály, které vyjadřují souřadnice čtených pixelů. Výstupní signály jsou bitové vektory jejichž rozsah je odvozen z rozlišení plošného senzoru.



Obr 7.5 Vstup CMOS LM9617 – blokové schéma

V jazyku VHDL (ukázka ze souboru *videoprocesor.vhd*) jsou výstupní signály definováno takto:

```
signal i_cmos: std_logic_vector(8 downto 0);
signal j_cmos: std_logic_vector(9 downto 0);
signal p_cmos: std_logic_vector(7 downto 0);
signal d_cmos: std_logic_vector(7 downto 0);
signal dp_cmos: std_logic;
```

Kód 7.1: Definice výstupních signálů bloku vstup CMOS LM9617

i_cmos: Tento signál představuje hodnotu i – tého řádku čteného pixelu.

j_cmos: Hodnota j – tého sloupce čteného pixelu.

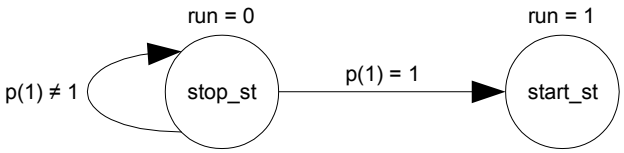
p_cmos: Hodnota p – tého snímku čteného pixelu.

d_cmos: Hodnota stupně šedi čteného pixelu.

dp_cmos: Signál data platná. S tímto signále se čtou hodnoty výše uvedených signálů. Jeho délka je jako jedna parioda *clk*.

Celý blok funguje tak, že se nejprve synchronizují vstupní signály z CMOS senzoru na jednu periodu *clk*. Například signál *pclk_met3* je synchronizován na signál *pclk_sr*, který je dlouhý jako jedna perioda *clk*. Synchronizované signály jsou dále použity pro výstupní čítače sloupců, řádků a snímků. Na výstupy jsou nakonec řazeny registry.

Na Obr 7.5 vlevo nahoře je stavový automat., který zajišťuje, aby nedošlo k tomu, že vstupním blokem projde například jen polovina snímku z CMOS senzoru po restartu videopreprocesoru. Stavový automat čeká po restartu na začátek třetího snímku a poté natrvalo otevře výstupní registry až do příchodu dalšího restartu videopreprocesoru. Jeho stavový diagram je na Obr 7.6. Signál *p(1)* je druhý bit vedený z čítače snímků viz. Obr 7.5 .



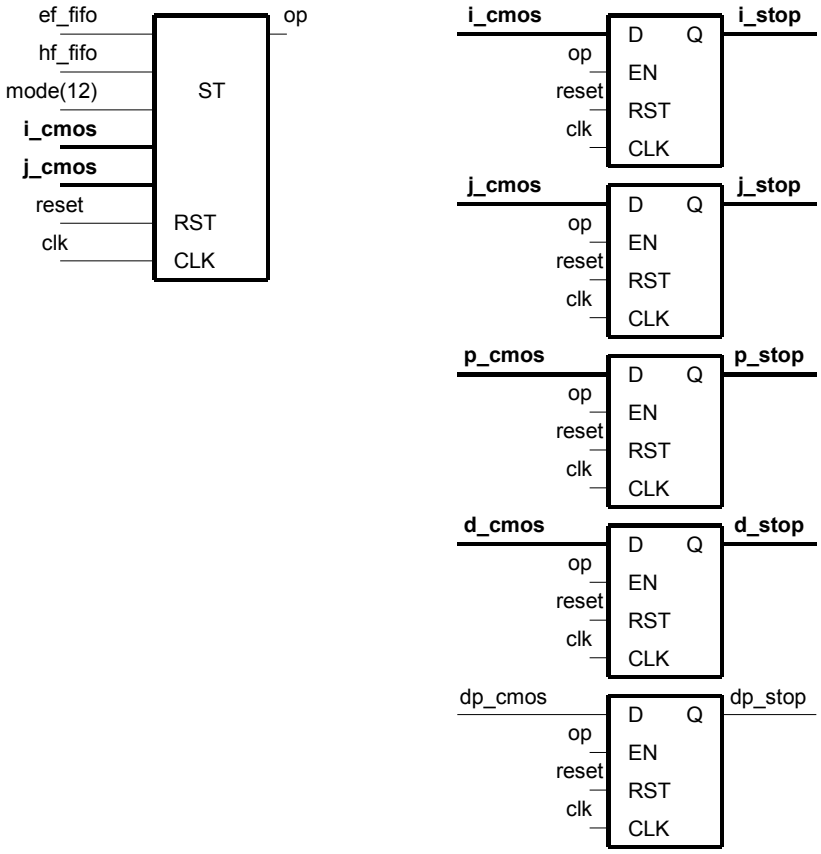
Obr 7.6 Stavový diagram automatu vstupního bloku

7.3 Blok stop data

Tento blok propouští celistvé snímky v závislosti na naplnění výstupní FIFO paměti, aby nedošlo k jejímu přetečení (viz. také kapitola *6.10 Blok STOP DATA*). Blok funguje ve dvou režimech v závislosti na nastavení bitu *mode (12)* SFR registru označeném FI.

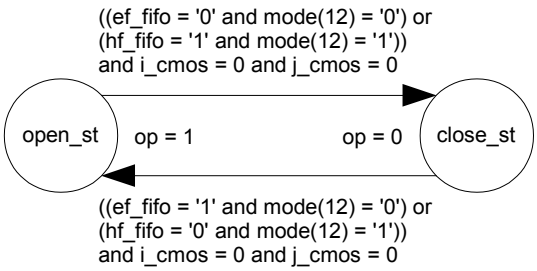
První režim („jeden snímek do FIFO a čeká se až je FIFO celá vyčtena „): Blokem projde celý jeden snímek. Následně je použita některá z metod zpracování obrazového signálu, například nalezení hran. Všechny hrany z jednoho snímku jsou uloženy do FIFO a poté se blok uzavře. Znovu se otevře až pokud je FIFO prázdná.

Druhý režim („polovina FIFO plná a poté stop“): Jako první režim, ale výstupní registry se uzavřou poté co je naplněna polovina FIFO.



Obr 7.7 Stop data – blokové schéma

Stavový diagram automatu, který řídí otevírání výstupních registrů signálem *op* je na Obr 7.8. Automat se překlápí pouze na začátku čtení celého obrázku (vyhodnocují se signály *i_cmos* a *j_cmos*) a přitom se vyhodnocuje stav naplnění FIFO pomocí signálů *ef_fifo* nebo *hf_fifo*.



Obr 7.8 Stavový diagram bloku stop data

7.4 Blok okno

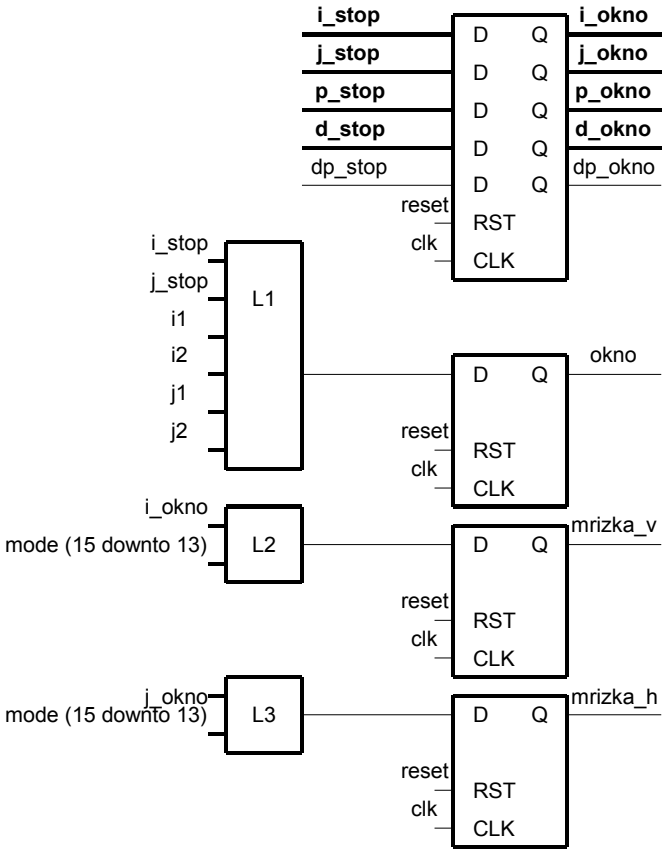
Tento blok přidává k signálům řádků, sloupců, snímků a dat ještě další tři signály:

okno: Logika L1 vyhodnocuje jestli čtený pixel daný souřadnicí *i_stop* a *j_stop* leží uvnitř okna daným souřadnicemi *i1, j1, i2, j2* (hodnoty souřadnic jsou uloženy v SFR registrech I1, J1, I2, J2) a pokud ano nastaví se signál okno do log. 1.

mrizka_v: Vertikální mřížka. Logika L2 vyhodnocuje jestli čtený pixel leží ve zvoleném řádku nastaveném v SFR registru *mode(15 downto 13)* označeném M a pokud ano nastaví se signál *mrizka_v* do log. 1.

mrizka_h: Horizontální mřížka. Logika L3 vyhodnocuje jestli čtený pixel leží ve zvoleném sloupci nastaveném v SFR registru *mode(15 downto 13)* označeném M a pokud ano nastaví se signál *mrizka_h* do log. 1.

Tyto tři signály dále vstupují spolu se signály obrazu do metod zpracování obrazového signálu a podle jejich stavu jsou výsledky metod (nalezené hrany atd.) bud uloženy nebo neuloženy do výstupní FIFO paměti.

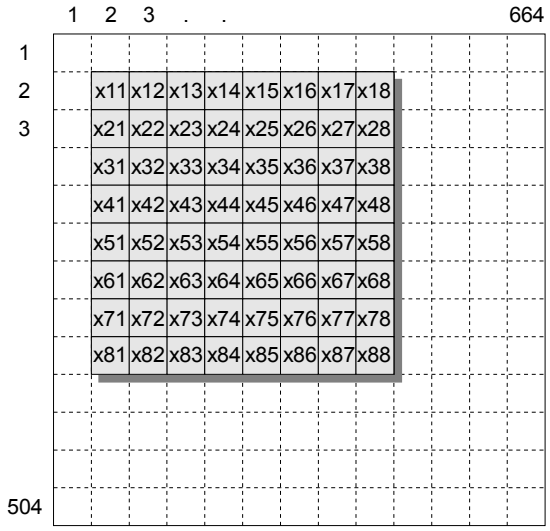


Obr 7.9 Okno – blokové schéma

7.5 Blok metod zpracování obrazového signálu

Rozkreslit blok metod zpracování obrazového signálu do blokových schémat složených z hradel a registrů by bylo příliš obsáhlé a nad rámec možností tohoto textu. Blok videopreprocesoru s názvem metody zpracování obrazového signálu zde bude rozkreslen pomocí menších bloků popsaných textem. Pouze klíčová část tohoto modulu – obrazová matice 8x8 pixelů – bude rozkreslena na jednotlivé logické obvody.

OBRAZOVÁ MATICE 8x8 PIXELŮ: Získat z načítaného snímku obrazovou matici o velikosti 8x8 pixelů dle Obr 7.10 s paralelním (souběžným) přístupem k hodnotám této matice je klíčové pro všechny metody zpracování obrazového signálu. Všechny výpočty se provádí z hodnot této matice.



Obr 7.10 Obrazová matice 8x8 pixelů

Při realizaci se musí myslet na to, že matice musí měnit všechny své hodnoty najednou s frekvencí signálu *pclk*, musí se posunovat po obraze nejprve zleva doprava a na konci každého řádku se posunout o jeden řádek níže.

Dle kapitoly *6.4 Vstup CMOS LM9617 – připojení plošného senzoru* se obrazový signál z CMOS plošné senzoru načítá po pixelech postupně po řádcích (každý o délce 664 pixelů). Pro získání obrazové matice o velikosti 8x8 bude muset existovat paměť o velikosti 7 x 664 + 8 B = 4656 B (1B = 8 bit odpovídá kódování jedné hodnoty pixelu). Použití bitových registrů (jejich počet by musel být 8 * 4656 = 37248) není možné. Použité FPGA typu Spartan 3 XC3S200 jich obsahuje 3840 a už by nezbyli žádné registry na realizaci ostatní logiky videopreprocesoru.

V úvahu připadá použití blokových RAM obvodu FPGA uvedeného typu. K dispozici je

celkem 12 blokových RAM, každá o velikosti 18432 bitů (celkem 221184 bitů) což dle předchozího odstavce stačí. Musí se však pamatovat na to, že hodnoty obrazové matice musí být možné paralelně číst a to velice rychle (s kmitočtem signálu *pclk*), aby na ně mohl být proveden v reálném čase výpočet například konvoluce s maskou mexický klobouk.

Problém s realizací masky pixelů 8x8 byl po dlouhých úvahách a po prostudování použití blokových RAM dle (16) vyřešen velice jednoduše a efektivně. Klíčové je použití blokové RAM zapojené jako circular buffer.

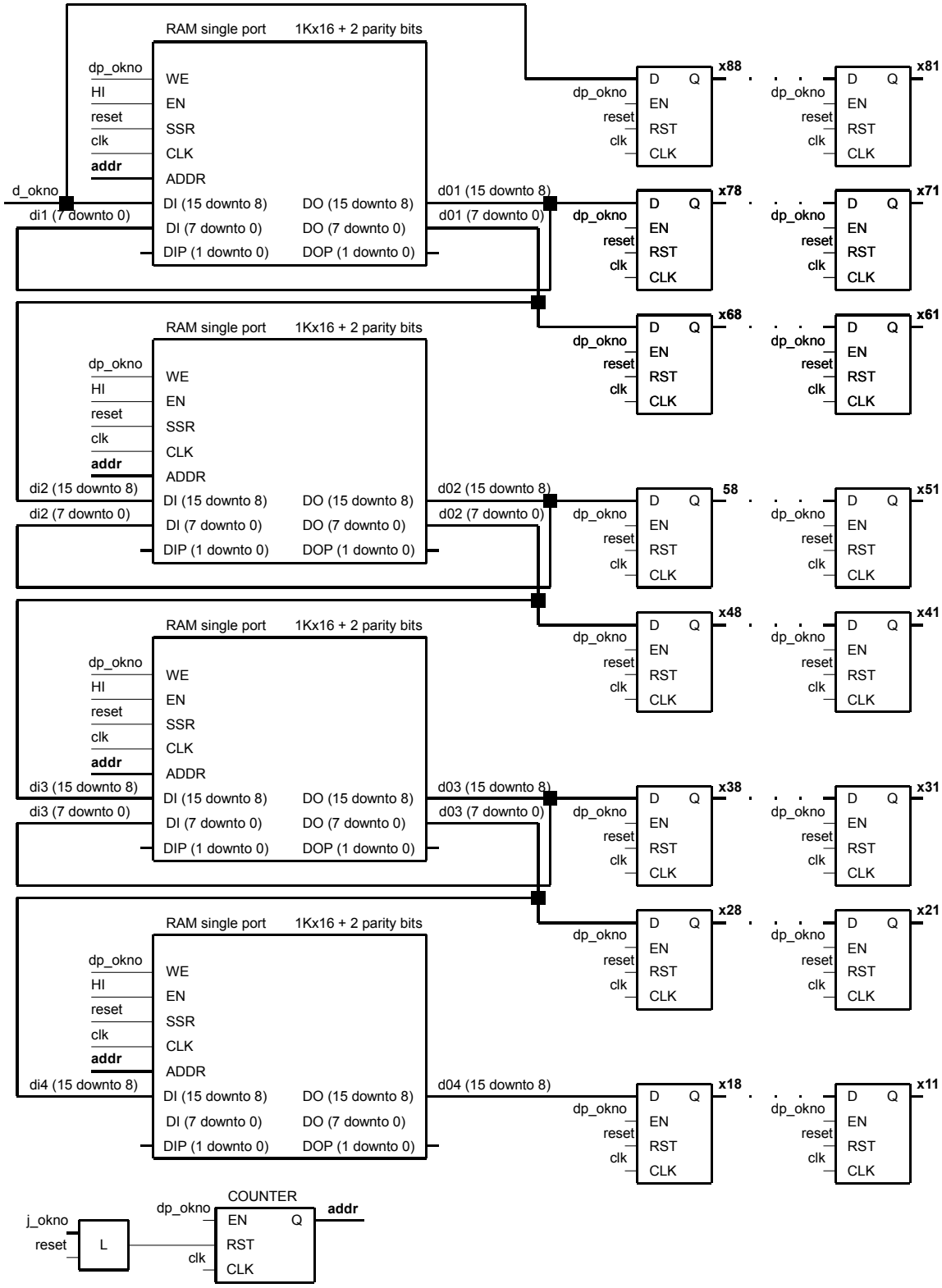
Circular buffer dle (16, s. 27) je paměť, do které se zapisují hodnoty s frekvencí hodinového kmitočtu a vyčítat se začnou samovolně též s frekvencí hodinového kmitočtu až poté, co dojde k jejímu úplnému zaplnění.

Pro obrazovou matici 8x8 pixelů byl použit 7 krát circular buffer zapojený sériově za sebe a na každý z jeho výstupů je připojena sada osmi osmibitových registrů + jedna sada na vstup prvního circular bufferu – celkem tedy 8x8 registrů, každý 8 bitů. Blokové schéma je dle Obr 7.11.

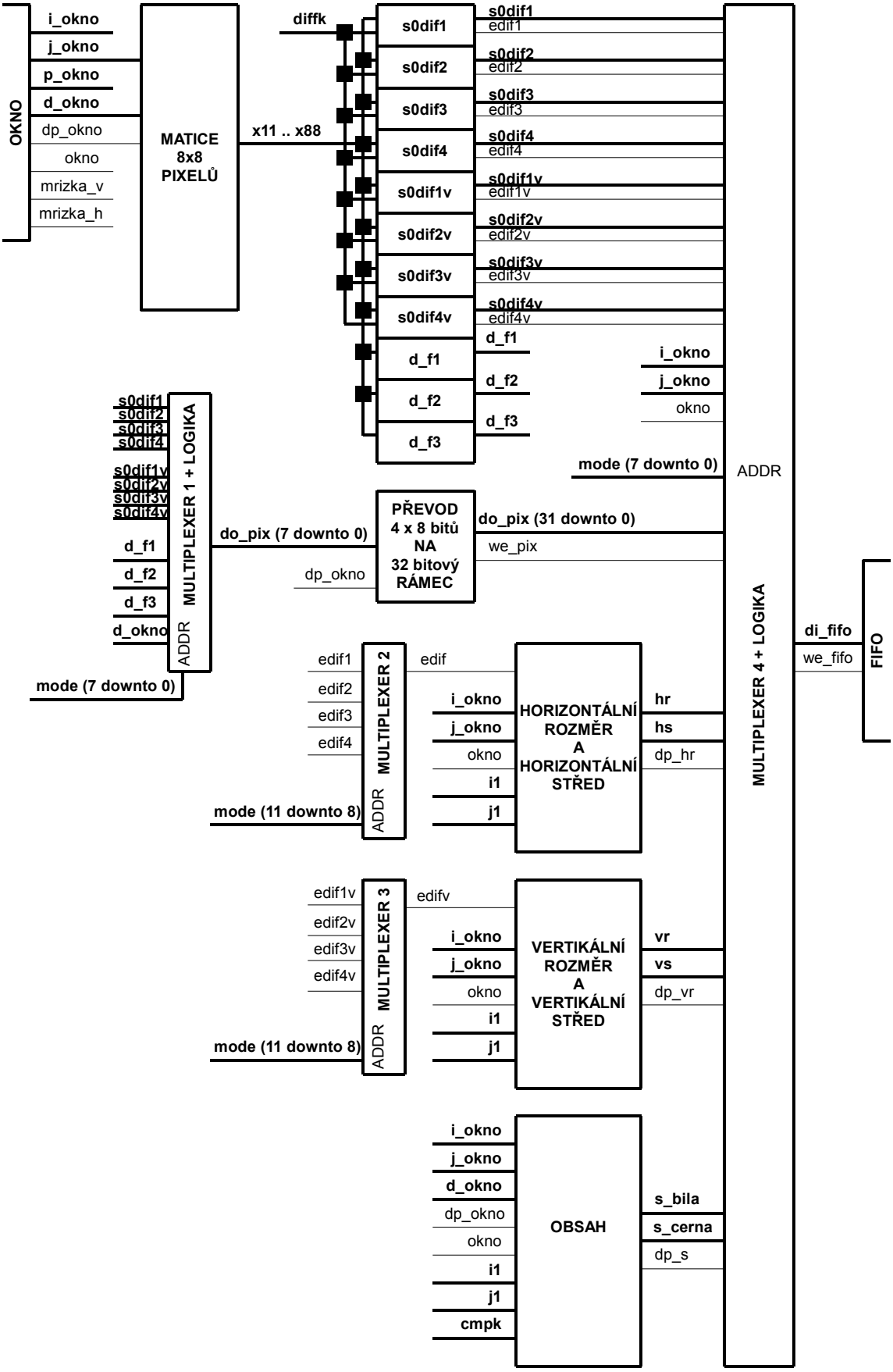
Data se do RAM načítají s frekvencí signálu *dp_okno* což odpovídá frekvenci signálu čtení dat z CMOS senzoru *pclk*. Čtení hodnot z výstupních registrů x11, x12 ... x88 je možné s frekvencí signálu *clk*.

Protože čítač adresy pro RAM je řízen signálem j_okno (sloupcový vektor), tak zapojení dle Obr 7.11 se dokáže samo přenastavit při jiném rozlišení CMOS plošného senzoru. To je jeho velká výhoda.

Další zpracování obrazového signálu je realizováno dle blokového schématu na Obr 7.12 Schéma je tvořeno bloky, které z obrazového signálu hledají hrany, rozměr atd. Tyto bloky jsou mezi sebou propojeny multiplexery s logikou. Propojení multiplexerů je závislé na nastavení hodnot speciálních funkčních registrů SFR viz. příloha *A*.



Obr 7.11 Matice 8x8 pixelů – blokové schéma



Obr 7.12 Metody zpracování obrazového signálu – blokové schéma

HLEDÁNÍ HORIZONTÁLNÍCH HRAN (hrany ve směru řádků): Je realizováno bloky s0dif1 – s0dif4. Každý z nich hledá hranu pomocí jiného výpočtu. Uživatel vybírá nastavením SFR registrů *mode (7 downto 0)* jeden z nich. Důležité jsou signály přerušení *edif1 – edif4*. Tyto signály vytváří impuls o délce jedné periody *clk* pokud je nalezena hrana. Vysvětlení hledání hrany bude ukázáno například na bloku s0dif4.

Pokud uživatel zvolí tento blok pomocí SFR registrů (signál *mode (7 downto 0)*) propojí přes multiplexer 4 signál *(edif4 and okno and mrizka_v)* se signálem *we_fifo* (signál umožnění zápisu do FIFO paměti – write enable). Současně se propojí signály *i_okno* a *j_okno* se signálem *di_fifo*. Znamená to tedy, že pokud je v signálu hrana a pokud tato hrana leží v uživatelem zvolené části obrazu (signál *okno* a signál *mrizka_v*), uloží se souřadnice řádku a sloupce hrany do výstupní FIFO paměti. Výpis z VHDL kódu pro multiplexer 4 je zde.

```
.
.

elsif mode(7 downto 0) = X"33" and mrizka_v = '1' then
      we_fifo <= edif4 and okno;
      di_fifo <= "0000000" & i_okno & "000000" & j_okno;
.
.
```

Kód 7.2: Ukázka VHDL kódu pro multiplexer 4

Samotná detekce hrany se provádí dle kapitoly *5.2 Detekce horizontálních hran*. Označení signálů je stejné.

Signál *s0dif4* by nemusel do výstupního multiplexeru 4 být veden, protože jako výsledek metody hledání hran jsou souřadnice, na kterém se hrana vyskytuje a ne hodnota tohoto signálu. Za účelem sledování hodnoty tohoto signálu však byla ponechána uživateli videopreprocesoru možnost tento signál vyčítat a proto je přes multiplexer 1 a převodní logiku do 32 bitového rámce do multiplexeru 4 připojen.

Hledání hran pomocí bloků s0dif1 – s0dif3 je systémově stejné jen výpočet diferenčního signálu je jiný:

HLEDÁNÍ VERTIKÁLNÍCH HRAN (hrany ve směru sloupců): Je systémově stejné jako hledání horizontálních hran, ale s dvěma rozdíly.

První rozdíl je zřejmý. Výpočet diferenčních signálů se provádí sloupcově (dle kapitoly *5.3 Detekce vertikálních hran*).

Druhý rozdíl vychází z problému při hledání vertikálních hran, které se od hledání horizontálních hran liší. Vysvětlení je popsáno též v uvedené kapitole.

Tento problém se podařilo velice elegantně a jednoduše vyřešit. Při realizaci masky pixelů pomocí blokových RAM zbyli u každé RAM ještě dva paritní bity, které nebyli zatím k ničemu použity. Čtení těchto bitů je se čtením pixelů synchronní a o stejné délce jednoho řádku. Paritní bity byly proto použity pro vyřešení uvedeného problému dle ukázky následujícího kódu. Paritní bit je v uvedeném kódu signál *dop1(0*).

```
if ("00000" & s0dif1v >= diffk) then
      edif1v <=  not dop1(0);
      dip1(0) <= '1';
else
      dip1(0) <= '0';
      edif1v <= '0';
end if;
```

Kód 7.3: Použití paritního bitu pro detekci hran

MĚŘENÍ ROZMĚRU A STŘEDU: Dle Obr 7.12 se pomocí multiplexerů 2 a 3 vybere metoda detekce hrany. Multiplexery jsou řízeny registry SFR. Dále je z prvních a posledních detekovaných hran vypočítán rozměr nebo střed objektu dle kapitoly *5 Metody zpracování obrazového signálu*.

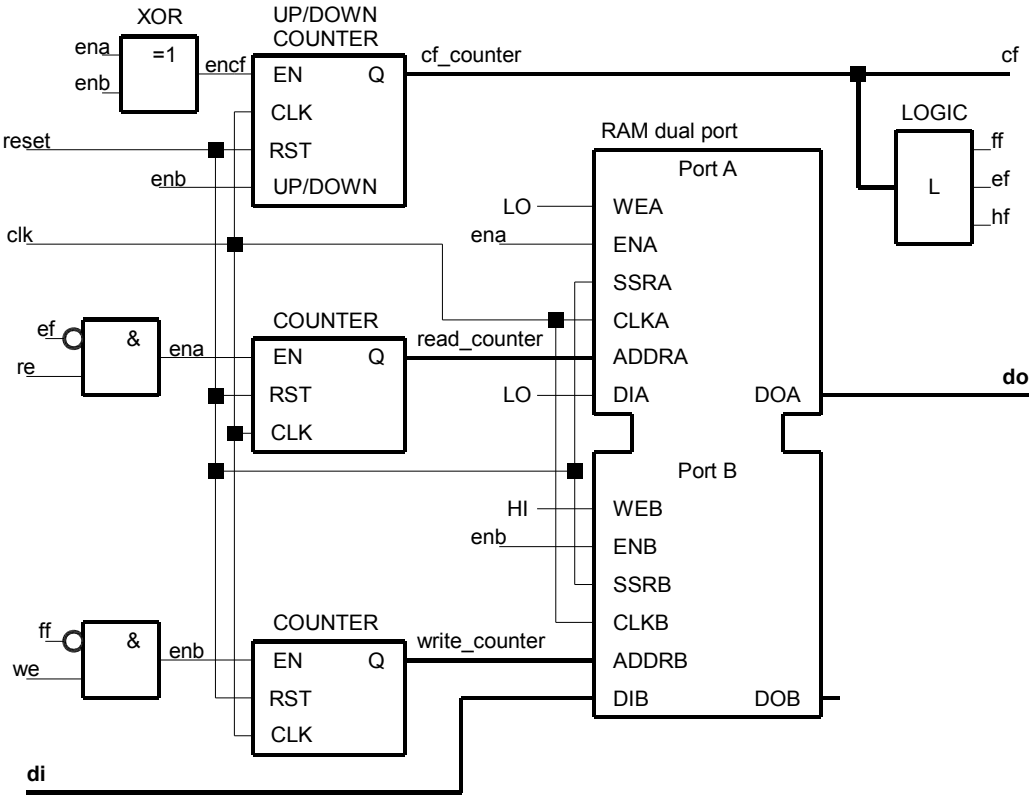
MĚŘENÍ OBSAHU: Provádí se dle kapitoly *5.6 Měření obsahu (komparační metoda)*. Blok obsah vyznačený v Obr 7.12 obsahuje čítač, který při měření světlých objektů přičítá hodnotu jedna pokud hodnota signálu *d_okno* překročí hodnotu signálu *cmpk*. Signál *cmpk* je konstanta nastavená pomocí registrů SFR. Při měření tmavých objektů čítač přičítá hodnotu jedna pokud hodnota hodnota *d_okno* podkročí hodnotu signálu *cmpk*.

POSÍLÁNÍ OBRAZOVÝCH DAT: Pro posílání původního obrazu nebo obrazu zpracovaného konvoluční maskou slouží blok vyznačený v Obr 7.12 s názvem „PŘEVOD 4 x 8 bitů NA 32 bitový RÁMEC“. Blok řadí čtveřici osmi bitů (pixelů) za sebe do formátu dle kapitoly *6.7 Formát výstupních dat* a paralelně je posílá na výstupní FIFO paměť.

7.6 FIFO paměť

Blokové schéma výstupní FIFO paměti je na Obr 7.13. Základem je použití RAM paměti s duálním portem a dvou čítačů. Jeden čítač inkrementuje svojí hodnotu (adresu pro RAM) při zápisu do RAM paměti a druhý čítač inkrementuje svojí hodnotu (adresu pro RAM) při čtení z RAM. Pokud mají čítače stejnou hodnotu je RAM a tedy FIFO paměť prázdná. Čím větší je rozdíl hodnot obou čítačů, tím je FIFO paměť více zaplněna.

Stav zaplnění FIFO paměti je vyhodnocován pomocí UP/DOWN čítače. Přes logiku L jsou z tohoto čítače vedeny stavové signály *ff* (FULL FIFO), *ef* (EMPTY FIFO) a *hf* (HALF FIFO). Přesný stav naplnění FIFO je dán hodnotou tohoto UP/DOWN čítače (signál *cf*).

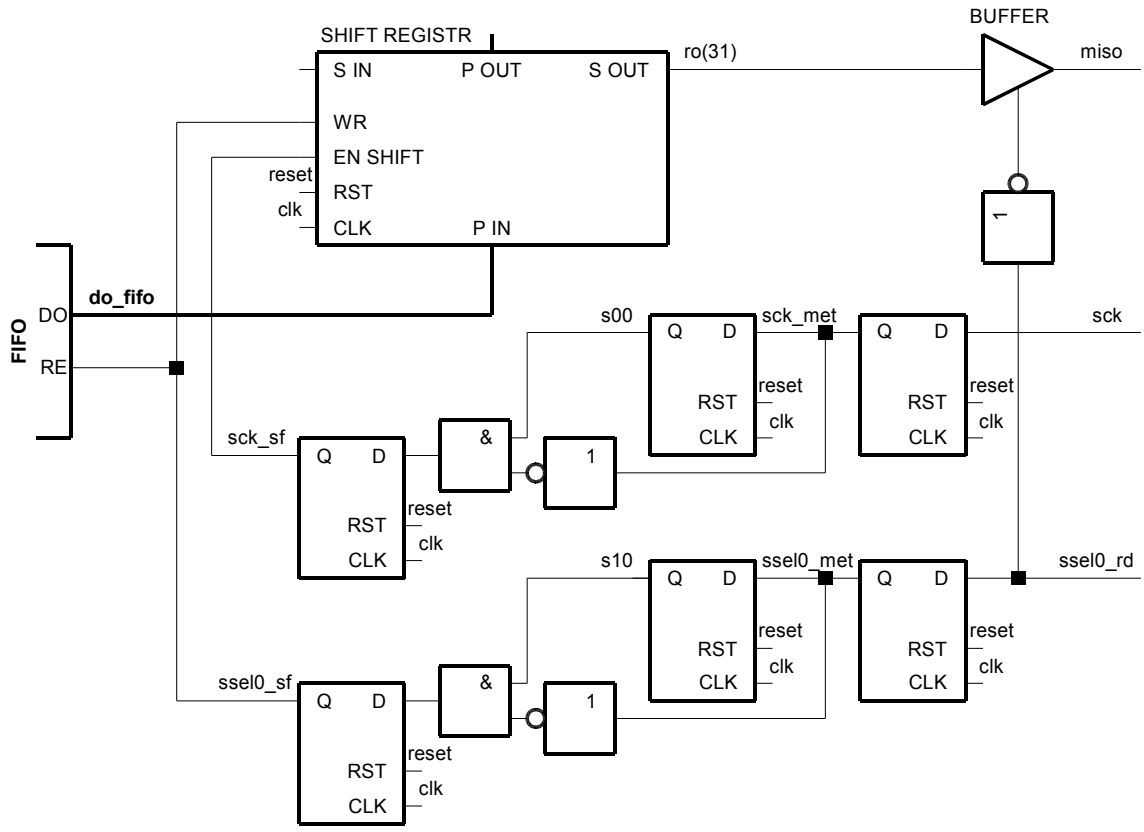


Obr 7.13 FIFO paměť – blokové schéma

Na uvedeném obrázku je zakreslena jedna RAM paměť. Při náhledu do VHDL kódu FIFO paměťi je vidět, že je použito 8 duálních RAM o velikosti 8k x 4. Mezi sebou jsou ovšem propojeny tak, že se chovají jako jedna RAM 8k x 32. Tento postup se musel zvolit, protože blokové RAM jsou uživatelem neměnné elementy obvodu FPGA.

7.7 Rozhraní SPIO

Blok SPI0 slouží pro sériové čtení dat z videopreprocesoru. Způsob jeho realizace je na Obr 7.14. Signály *sck* a *ssel_rd* jsou nejprve synchronizovány na jednu periodu signálu *clk* u jejich spádové hrany. Signálem *sck_sf* je řízeno posouvání dat z výstupního 32 bitového posuvného registru přes výstupní signál *miso*. Do výstupního registru jsou data z FIFO paměti načítány paralelně pomocí signálu *ssel0_sf*.

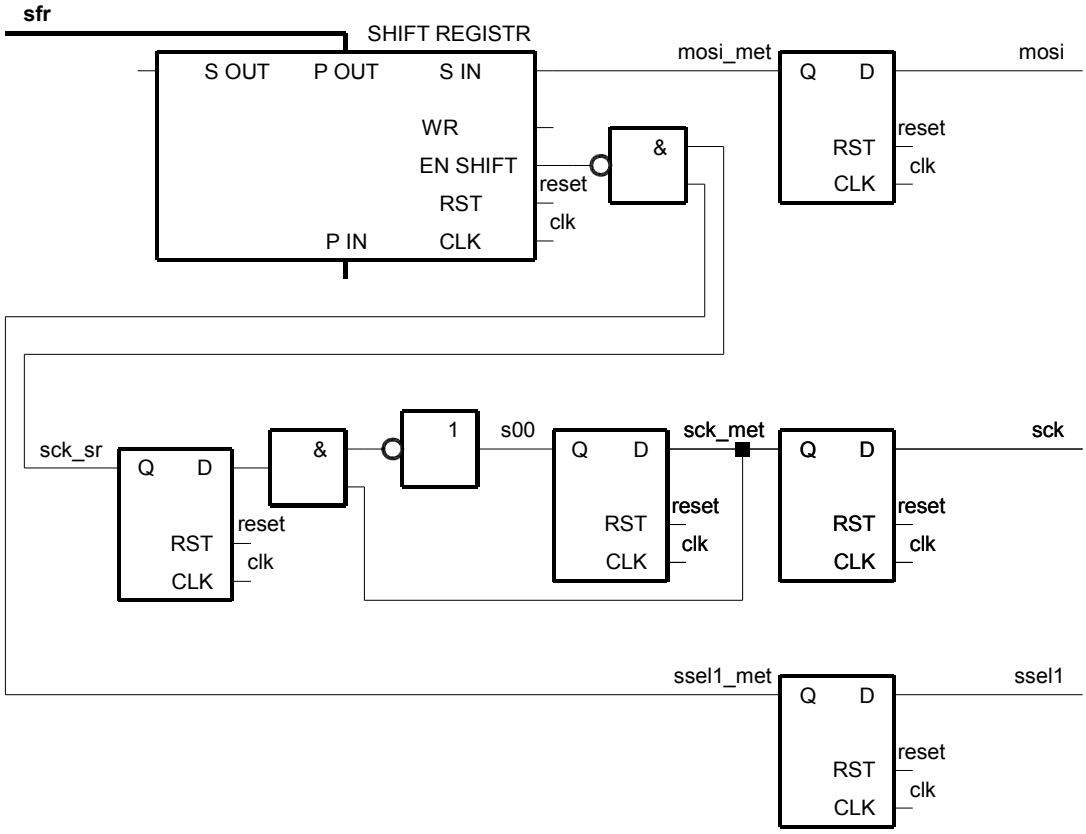


Obr 7.14 SPIO – blokové schéma

7.8 Rozhraní SPI1 a registry SFR

Rozhraní SPI1 je přímo propojeno se speciálními funkčními registry SFR a proto je uvedeno na jednom blokovém schématu dle Obr 7.15.

Signál *sck* je opět nejprve synchronizován na jednu periodu signálu *clk* u jeho náběžné hrany. Spolu se signálem *ssel1_met* řídí načítání dat do vstupního posuvného registr SFR přes vstupní signál *mosi*.



Obr 7.15 SPI1 a SFR – blokové schéma

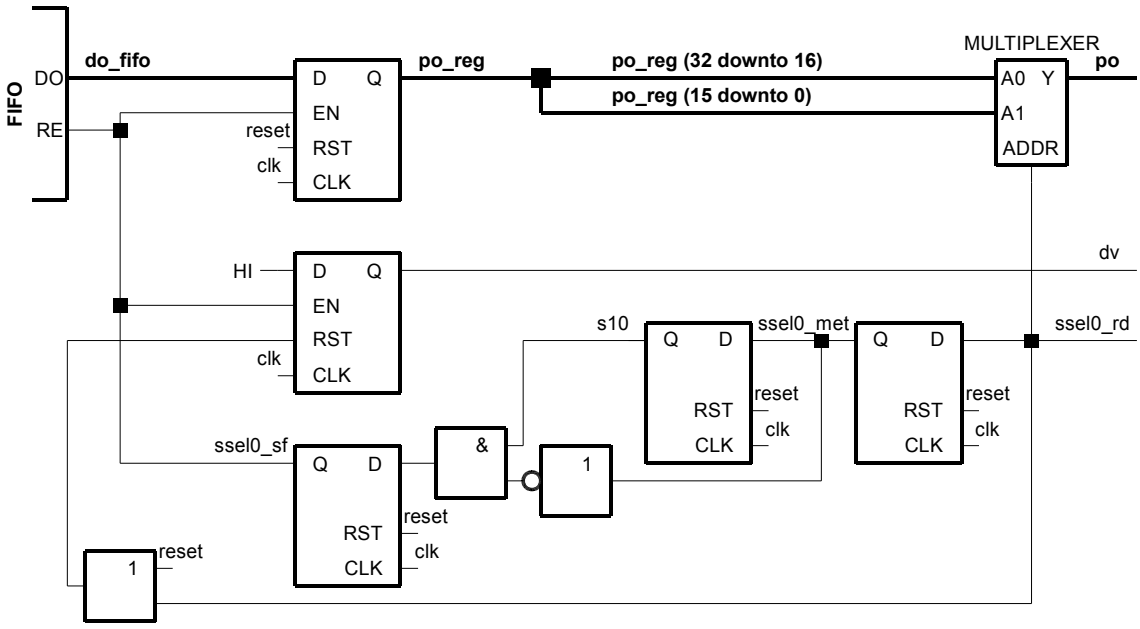
Signálu *sfr* byly na začátku VHDL kódu přiřazeny aliasy, jejichž název je stejný dle kapitoly *6.6 Nastavení funkcí videopreprocesoru*. Jedná se o názvy speciálních funkčních registrů viz. následující ukázka kódu.

```
--sfr(specialni funkcni registry)
signal sfr  : std_logic_vector (111 downto 0);
alias cmpk  : std_logic_vector(15 downto 0) is sfr(111 downto 96);
alias diffk : std_logic_vector(15 downto 0) is sfr(95 downto 80);
alias mode  : std_logic_vector(15 downto 0) is sfr(79 downto 64);
alias i1    : std_logic_vector(15 downto 0) is sfr(63 downto 48);
alias j1    : std_logic_vector(15 downto 0) is sfr(47 downto 32);
alias i2    : std_logic_vector(15 downto 0) is sfr(31 downto 16);
alias j2    : std_logic_vector(15 downto 0) is sfr(15 downto 0);
```

Kód 7.4: Registry SFR

7.9 Rozhraní PI

Blokové schéma paralelního výstupního rozhraní PI je na Obr 7.16. Signál *ssel0_rd* je synchronizován na signál *ssel0_sf* pomocí kterého je načítána 32 bitová hodnota z FIFO paměti do výstupního registru. Následně je tato hodnota multiplexována do 2 krát 16 bitů. Z toho vyplývá způsob čtení dat po paralelní sběrnice PI dle kapitoly *6.9 Paralelní čtení dat (PI)* .



Obr 7.16 PI – blokové schéma

7.10 Příznakové signály

Řešení příznakových signálů zde pro svou jednoduchost nebude rozkresleno, ale je zde ukázán jejich kód v jazyku VHDL. Signály *ef* a *sg* jsou přímo propojeny s FIFO pamětí. Signál *ovf* je veden přes paměťovou buňku.

```
ef <= ef_fifo;

--signal preteceni FIFO - owerflov
process (clk, reset)
      begin
            if reset = '1' then
                  ovf <= '0';
            elsif rising_edge(clk) then
                  ovf <= ff_fifo or ovf;
            end if;
      end process;

--signal poloprazdne FIFO - speed good
sg <= hf_fifo;
```

Kód 7.5: Příznakové signály

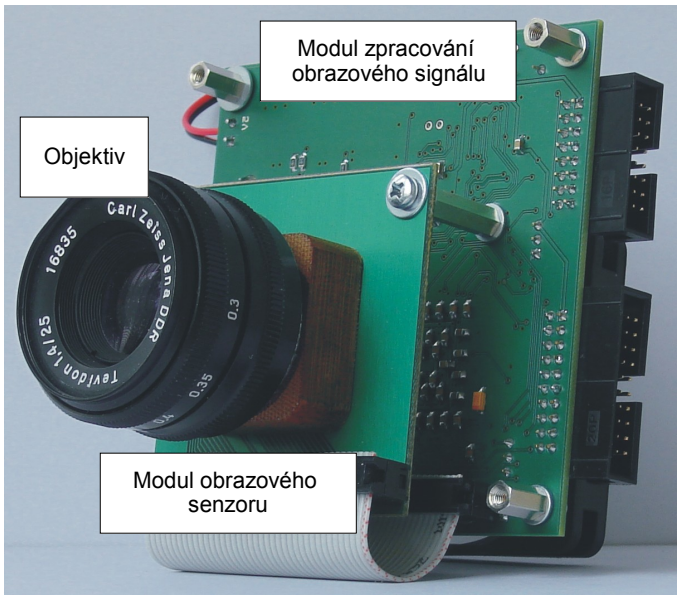
8. Seznámení s videosenzorem

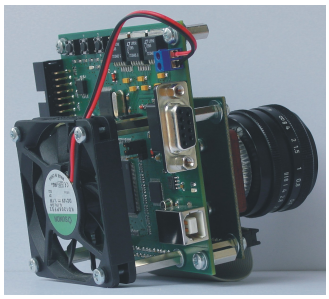
Tato kapitola uvádí základní popis videosenzoru a jeho používání. Po stručném popisu jeho základních částí následuje způsob jeho připojení prostřednictvím sériové linky k PC, způsob nahrání programu (firmware) do videosenzoru a způsob zobrazení naměřených hodnot a obrazu měřeného objektu.

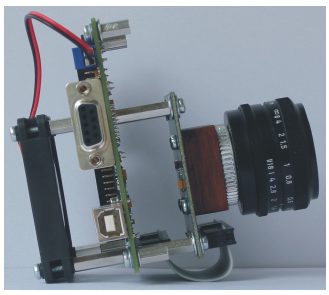
8.1 Popis videosenzoru

Videosenzor je dle Obr 8.1 složen ze tří základních částí:

- Objektiv
- Modul obrazového senzoru
- Modul zpracování obrazového signálu







Obr 8.1 Popis videosenzoru

Objektiv spolu s modulem obrazového senzoru tvoří jeden celek. Jedná se o semestrální projekt (Jan Šedivý, 2003), který byl zapůjčen a použit ke konstrukci videosenzoru. Modul obrazového senzoru je spolu s modulem zpracování obrazového signálu propojen paralelní sběrnicí (plochý kabel), po které se přenáší digitální obrazový signál.

Jádrem videosenzoru je modul zpracování obrazového signálu. Celý modul byl navržen v rámci této diplomové práce. Všechny metody zpracování obrazového signálu jsou nahrány v hradlovém poli FPGA osazeném na tomto modulu, program pro řízení modulu je nahrán procesoru ARM7 a program pro přenos obrazu do PC je nahrán v obvodu EZ-USB.

8.2 Co měří videosenzor

Videosenzor (kamera pro bezkontaktní měření) měří rozměry objektů, které se nacházejí v jeho zorném poli. Videosenzor nedisponuje žádnými metodami pro rozpoznávání nebo vyhledávání objektů. Jedná se o prosté měření objektů, které se provede výpočtem rozměrů z nalezených hran objektů. Zejména z tohoto důvodu musí být při měření splněny alespoň tři podmínky:

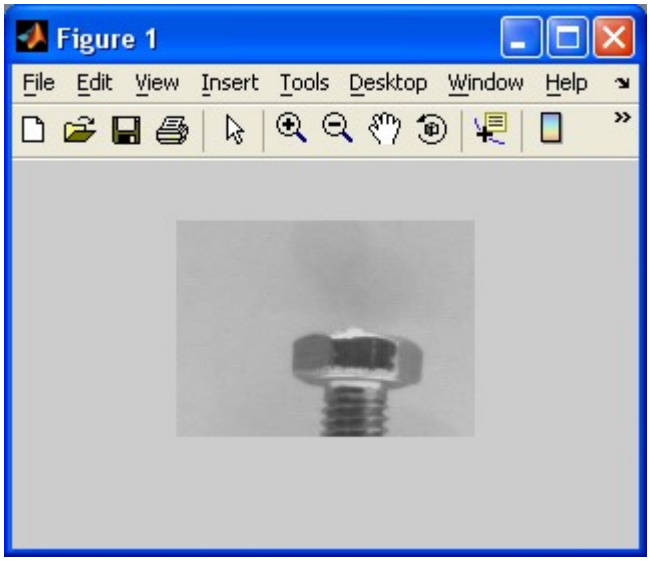
- jedná se o tmavý objekt na světlém pozadí nebo naopak
- videosenzor je na měřený objekt zaostřen objektivem
- při opakovaném měření se zaostření objektivu nemění

Ukázka měření polohy hran objektu (šroubu) videosenzorem je na Obr 8.2 a Obr 8.3. Videosenzor byl propojen pomocí sériové linky s PC a byly spuštěny dva jednoduché programy v MATLABU.

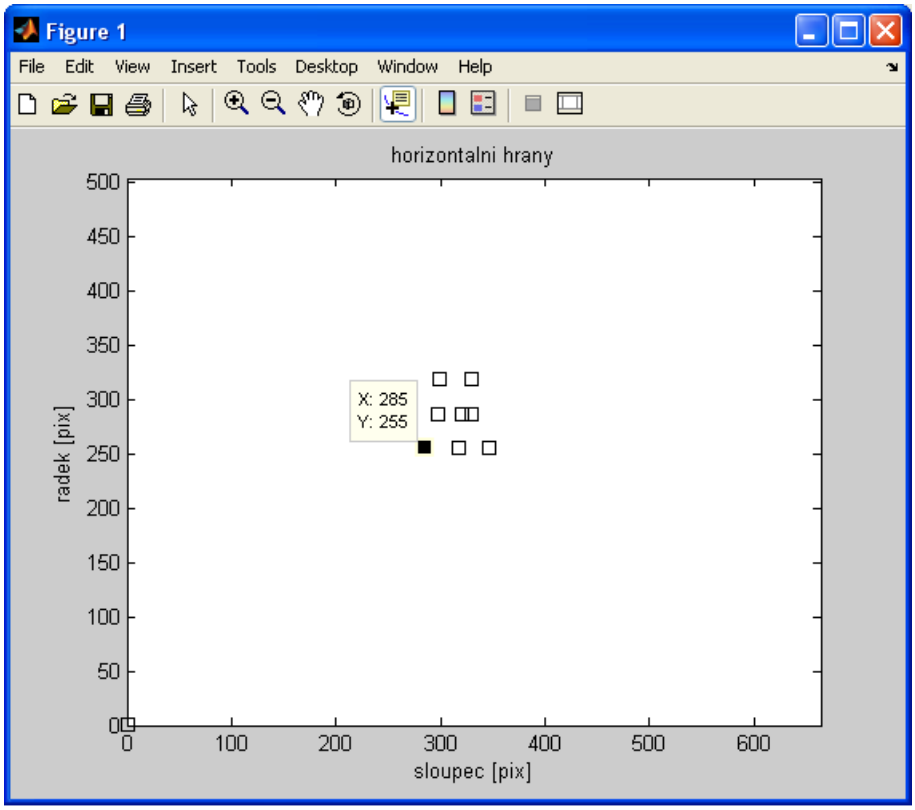
První program nastavil videosenzor na mód posílání obrazu a zobrazuje sledovanou scénu viz. Obr 8.2. Šroub, který je položen na bílé papíře a je patřičně zaostřen.

Druhý program nastavil videosenzor na mód vyhledávaní hran v každém třicátém druhém řádku viz. Obr 8.3. Na obrázků jsou vyhledané hrany, jejich souřadnice, které byly odeslány ze senzoru do PC po sériové lince.

Obdobně vypadají i ostatní měření videosenzorem. Například při nastavení videosenzoru na mód měření horizontálního rozměru bude tento rozměru zpětně posílán do PC, kde je zobrazen.



Obr 8.2 Skutečný obraz šroubu zaostřený videosenzorem



Obr 8.3 Hrany šroubu nezelené videosenzorem

8.3 Napájení videosenzoru

Napájení videosenzoru je vyvedeno na modulu zpracování obrazového signálu. Napájecí zdroj

musí mít jmenovitou hodnotu 5V DC. Polarita napájení je uvedena na spodní straně modulu. Kladný pól je na pinu, který je blíže nápisu 5V.

8.4 Připojení videosenzoru k PC

Videosenzor se připojuje na sériovou linku PC pomocí přímého (nekříženého) kabelu zakončeného konektory CANNON 9. Nezbytné signály jsou TXD, RXD, RTS a DTR.

8.5 Nahrání programu do ARM7

První krok pro zprovoznění videosenzoru je nahrání řídícího programu (firmware) do procesoru ARM7. K tomu slouží program *Flash Utility*, jehož cesta na doprovodném CD je:

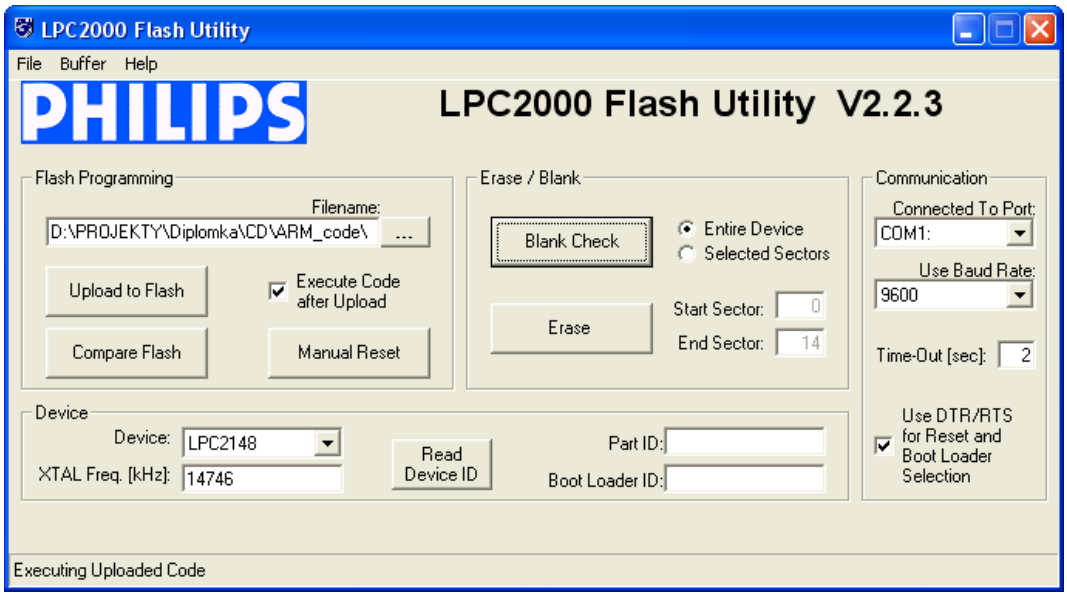
Programy\Philips Flash Utility Installation.exe.

Program se musí nainstalovat. Po jeho spuštění uživatel provede nastavení dle Obr 8.4 a vybere číslo sériového. Na modulu zpracování obrazového signálu musí být osazen jumper J3.

Následuje výběr programovacího souboru. Jeho cesta na doprovodném CD je:

ARM_code\Start_ARM\Obj\Blinky.hex

Po stisknutí tlačítka „Upload to Flash“ se nahraje program do procesoru ARM7.



Obr 8.4 Nastavení programu *Flash Utility*

8.6 Nahrání konfiguračního souboru do FPGA

Po úspěšném zavedení programu do ARM7 následuje nahrání konfiguračního souboru do hradlového pole FPGA. Nahrání probíhá pomocí terminálu. Lze použít jakýkoliv terminál, který umí komunikovat po sériové lince a odesílat soubor.

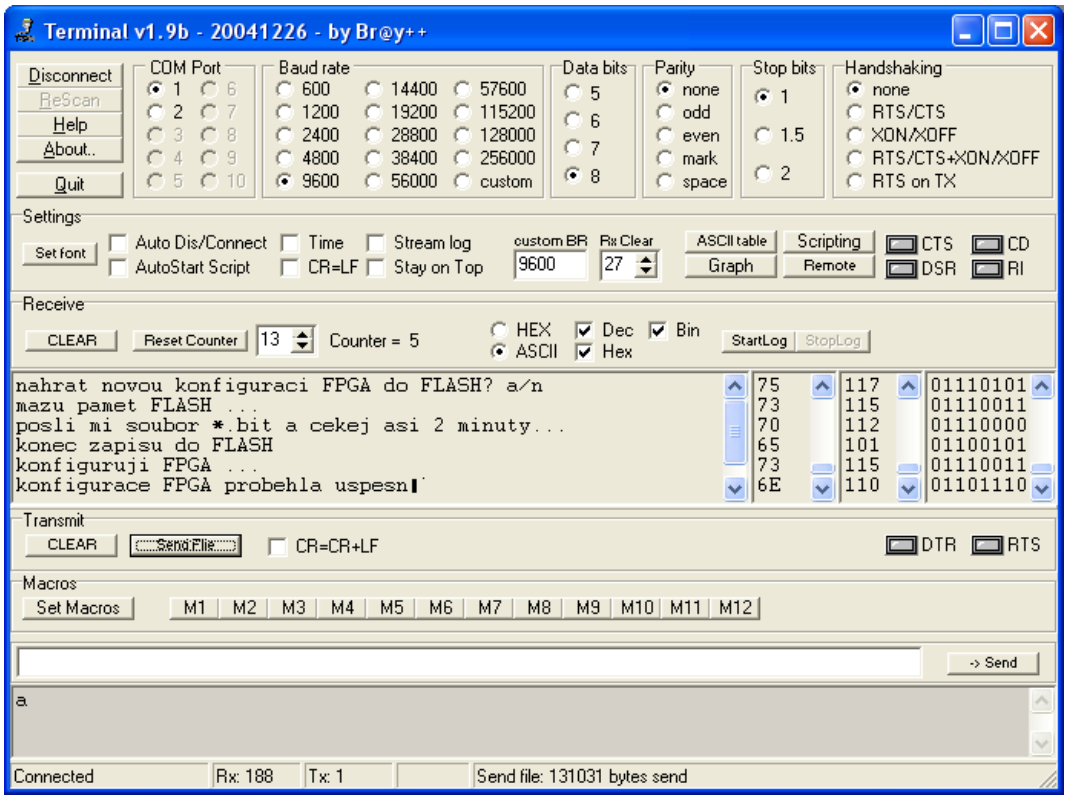
Na terminálu nesmí být aktivní signál DTR, který se používá při programování procesoru ARM7 (tento signál je připojen na jeho signál RESET). V případě, že signál DTR nelze v terminálu vypnout, je nutno odpojit jumper J3.

Vhodný terminál pro sériovou komunikaci, který se nemusí instalovat je umístěn na doprovodném CD:

Programy\Terminal - pro odchytávání z com.exe

Na terminálu uživatel nastaví přenosovou rychlost 9600 Bd, vybere číslo příslušného sériového portu a stiskne se tlačítko „Connect“. Následně na modulu zpracování obrazového signálu musí stisknout tlačítko P0.30 a současně tlačítko „RESET ARM“. Dále se postupuje dle pokynů v okně terminálu viz. Obr 8.5. Konfigurační soubor pro hradlové pole FPGA je na doprovodném CD:

FPGA_code\ISE_WebPack_7_1i\Videoprocesor\Videoprocesor.bit



Obr 8.5 Terminál sériové linky

8.7 Zobrazení naměřených dat

Za účelem zobrazení naměřených hodnot byly napsány krátké programy v jazyku MATLAB. Programy komunikují s videosenzorem po sériové lince lince. Propojovací kabel je stejný jako v kapitole *8.4 Připojení videosenzoru k PC*. Při spuštění programu musí být na modulu zpracování obrazového signálu odpojen jumper J3.

Programy jsou jednoúčelové a stejně jako firmware do procesoru ARM7 byly napsány se záměrem ověření metod zpracování obrazového signálu implenentovaných do obvodu FPGA. Každý z programů nejprve nastaví po sériové lince videosenzor na příslušnou metodu a dále přijímá od senzoru data, které následně zobrazí.

Měření videosenzorem se spustí ve dvou krocích. Za prvé se provede reset videosenzoru tlačítkem „RESET“ na modulu zpracování obrazového signálu. Za druhé se otevře příslušný soubor *.m a spustí se pomocí funkční klávesy F5. Následuje zobrazení naměřených dat. Program se musí nechat doběhnou až do konce.

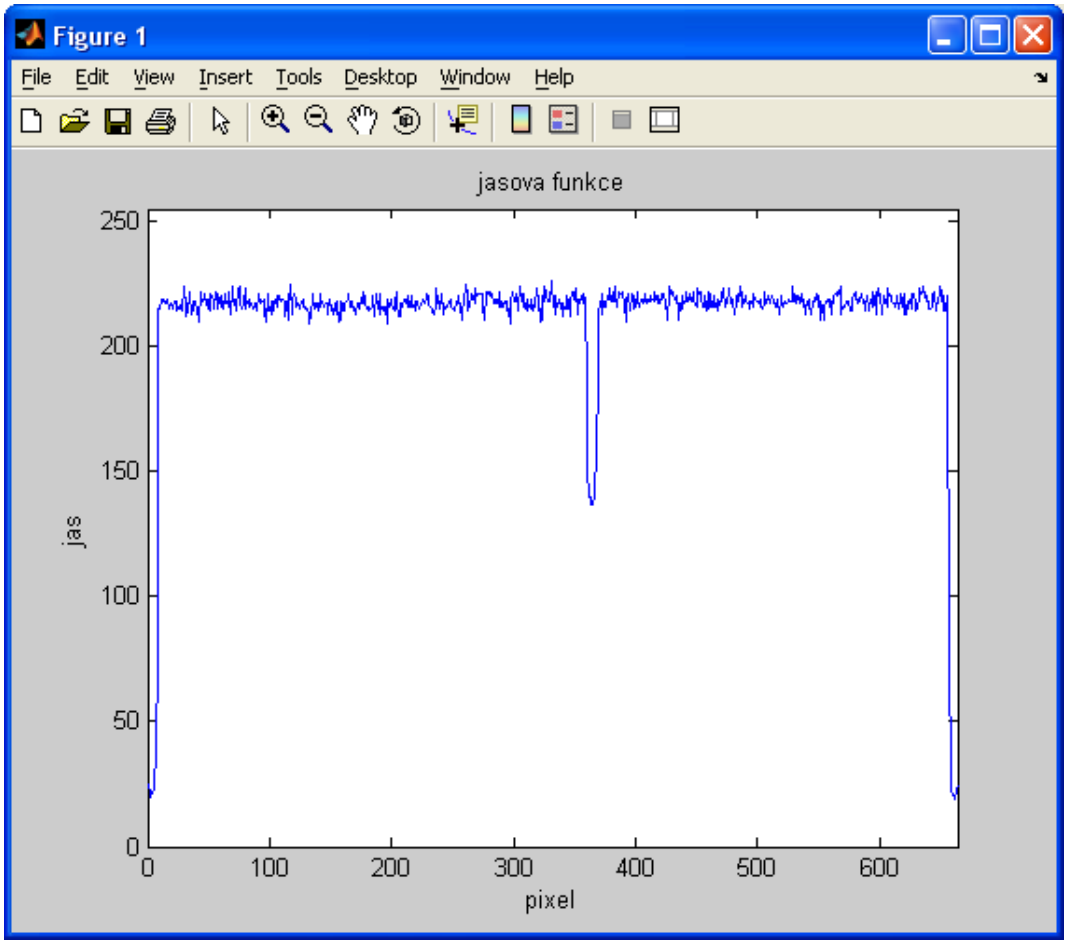
V tabulce 2 je uveden seznam všech souborů *.m vytvořených pro otestování funkce videosenzoru včetně jejich popisu. Všechny soubory jsou nahrány na doprovodném CD:

CD\Matlab_code

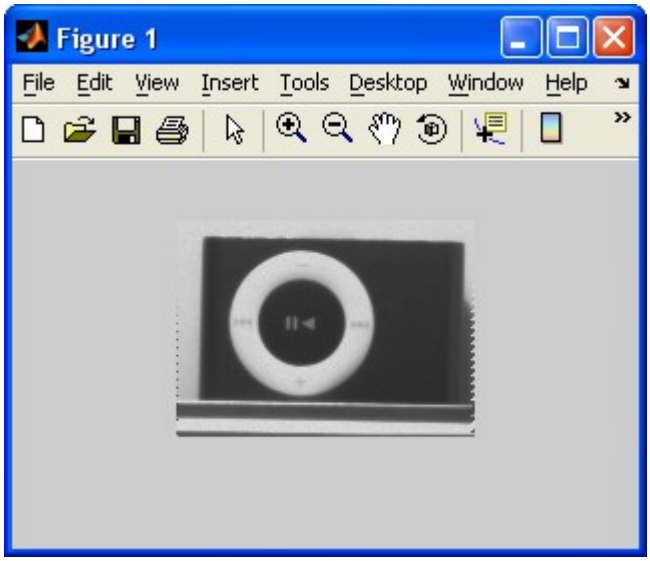
Jejich funkčnost byla ověřena v MATLABU verze R2008a, který je ke stažení na serveru ČVUT pro studenty ČVUT. Dále jsou uvedeny ukázky ze spuštěných *.m souborů.

NÁZEV SOUBORU	POPIS
radek.m	Zobrazuje jasovou funkci jednoho řádku obrazové matice (sériové čtení dat z FPGA).
radek_PI.m	Zobrazuje jasovou funkci jednoho řádku obrazové matice (paralelní čtení dat z FPGA).
hrany_vsechny.m	Hledání hran v obraze v horizontálním i vertikálním směru.
hrany_vertikalni.m	Hledání hran v obraze ve vertikálním směru.
hrany_horizontalni.m	Hledání hran v obraze v horizontálním směru.
rozmer_horizontalni.m	Měření horizontálního rozměru.
rozmer_vertikalni.m	Měření vertikálního rozměru.
stred.m	Určení středu.
plocha_bila.m	Měření plochy bílých objektů.
plocha_cerna.m	Měření plochy černých objektů.
obraz_puvodni.m	Zobrazuje obraz sledovaný videosenzorem.
obraz_laplace.m	Obraz po filtraci konvoluční maskou s laplaceovým operátorem.
obraz_mexicky_klobouk.m	Obraz po filtraci konvoluční maskou s operátorem „mexický klobouk“.
obraz_prewitt.m	Obraz po filtraci konvoluční maskou s prewittovým operátorem.
obraz_s0dif1.m	Obraz po filtraci konvoluční maskou [1 -1].
obraz_s0dif2.m	Obraz po filtraci konvoluční maskou [1 1 -1 -1].
obraz_s0dif3.m	Obraz po filtraci konvoluční maskou [1 1 1 -1 -1 -1].
obraz_s0dif4.m	Obraz po filtraci konvoluční maskou [1 1 1 1 -1 -1 -1 -1].
obraz_s0dif1v.m	Obraz po filtraci konvoluční maskou $[1\ -1]^T$.
obraz_s0dif2v.m	Obraz po filtraci konvoluční maskou $[1\ 1\ -1\ -1]^T$.
obraz_s0dif3v.m	Obraz po filtraci konvoluční maskou $[1\ 1\ 1\ -1\ -1\ -1]^T$.
obraz_s0dif4v.m	Obraz po filtraci konvoluční maskou $[1\ 1\ 1\ 1\ -1\ -1\ -1\ -1]^T$.

Tabulka 2: Seznam programů pro zobrazení naměřených hodnot

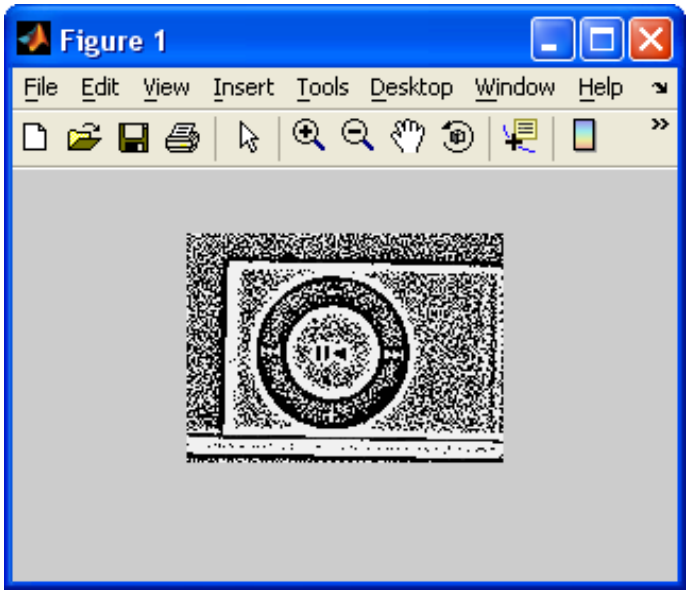


Obr 8.6: Jasová funkce řádku (*radek.m*)



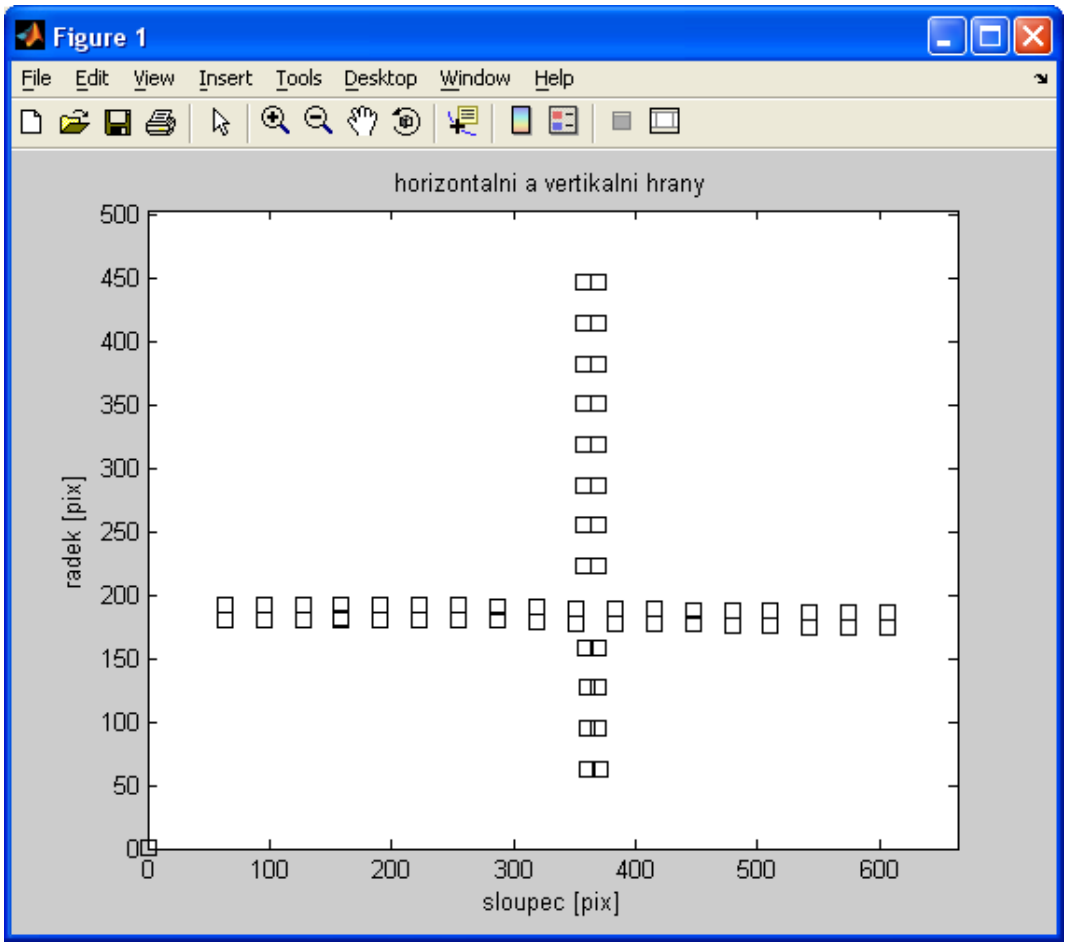
Obr 8.7 Původní obraz sledované scény (

obraz_puvodni.m)



Obr 8.8: Filtrace operátorem „mexický klobouk“

(*obraz_mexicky_klobouk.m*)



Obr 8.9: Hledání hran ve sloupcích a řádcích (*hrany_vsechny.m*)

8.8 Zobrazení sledované scény pomocí rozhraní USB

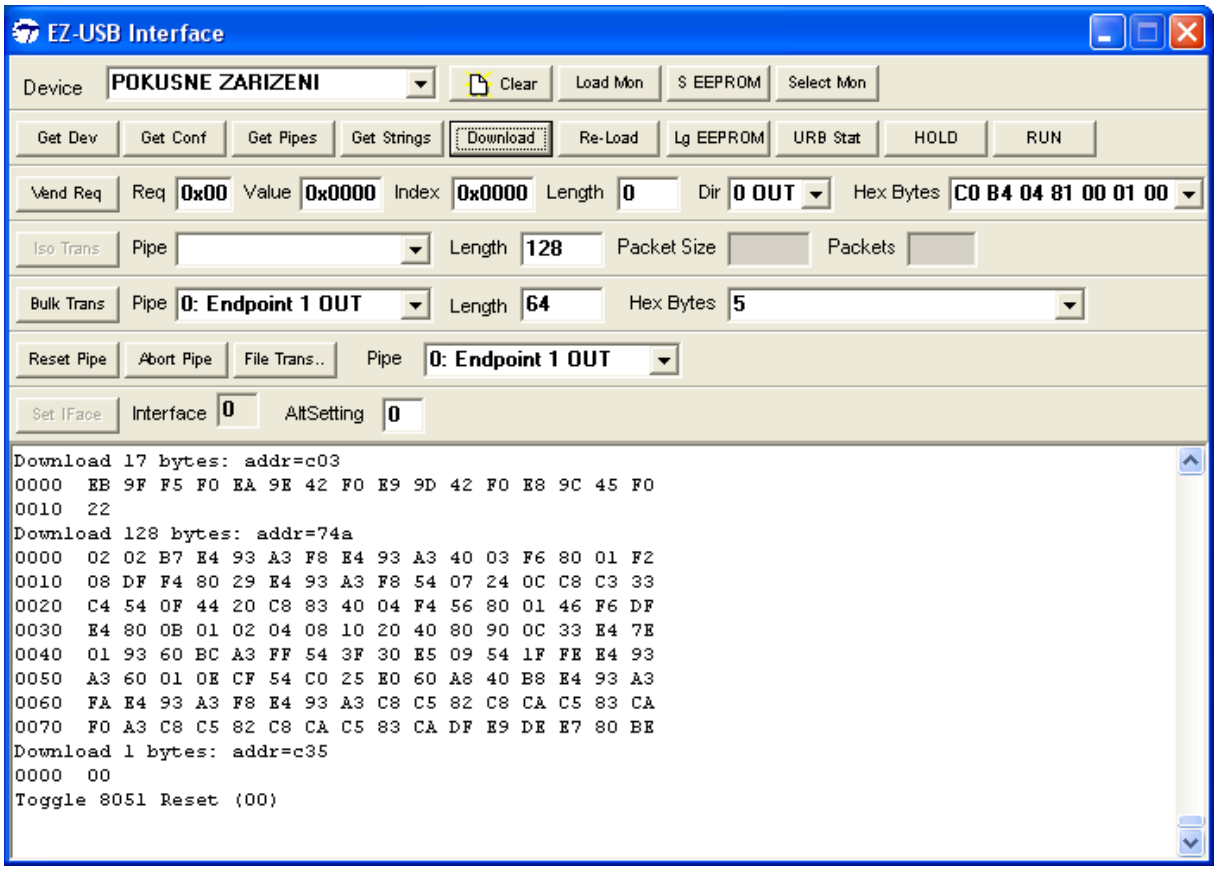
V kapitole *7 Videopreprocesor – vnitřní popis (VHDL)* byl na Obr 6.2 byl uveden blok rozhraní USB s názvem EZ-USB DATA. Tento blok umožňuje přenos sledované scény do PC za účelem zaostření videosenzoru na měřený objekt.

Verze videopreprocesoru (soubor *videoprocesor.vhd* a konfigurační soubor *videoprocesor.bit*), která tento blok implementuje spolu se všemi původními metodami videopreprocesoru se nachází na doprovodném CD:

FPGA_code\ISE_WebPack_7_1i\Videoprocesor+asych_USB\videoprocesor.bit

Pro zobrazení sledované scény se uvedený soubor musí nahrát dle kapitoly *8.6 Nahrání konfiguračního souboru do FPGA* do hradlového pole.

Dále se musí nahrát program do obvodu EZ-USB (Cypress CY7C68013A). To se provádí pomocí programu CyConsole. V programu je potřeba otevřít okno „EZ-USB interface" dle Obr 8.10).



Obr 8.10 EZ-USB interface

Program CyConsole se automaticky nainstaluje spolu vývojovým software, který se nachází na doprovodném CD:

Programy\SETUP_FX2LP_DVK_1004.exe

Program, který se do obvodu EZ USB musí nahrát pomocí tlačítka „download“ se nachází na doprovodném CD:

EZUSB_code\v0.1.2-odzkousena\EP_Interrupts.hex

Posledním krokem je spuštění zobrazovacího programu na PC. Ten se nachází na doprovodném CD:

EZUSB_code\VideoStreaming\VideoStreaming.exe

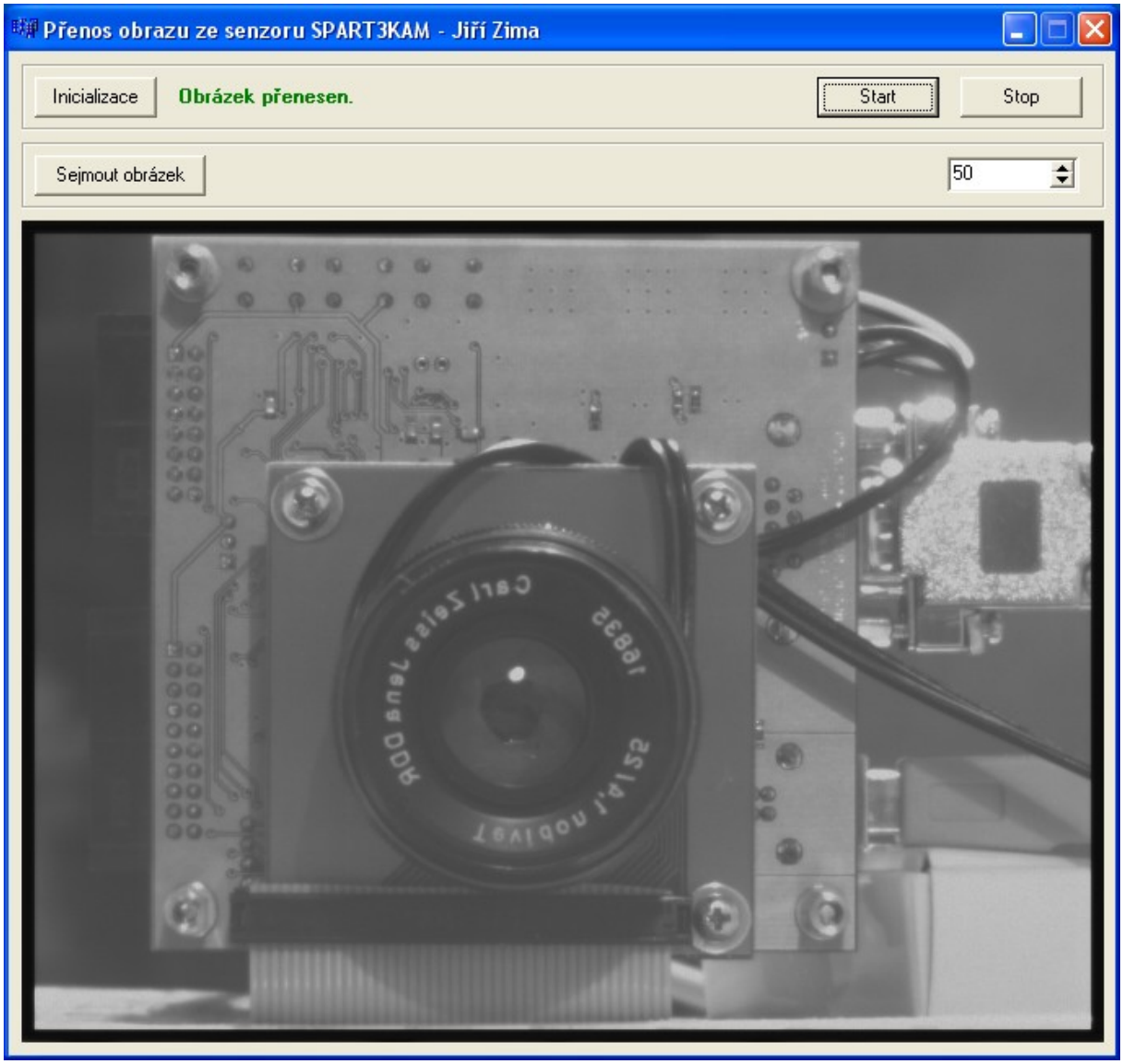
Pokud je na PC nainstalováno vývojové prostředí C++ Builder, program se otevře a po kliknutí na tlačítka Inicializace a Start se začne přenášet obraz viz. Obr 8.11.

Program (firmware) do obvodu EZ-USB (programovací soubor *EP_Interrupts.hex*) byl napsán v rámci práce (6) a pro ověření funkce přenosu obrazu po USB byl do této práce zapůjčen a použit. Jedná se o asynchronní verzi firmware.

Program *VideoStreaming.exe* byl napsán v rámci práce (7) a pro ověření funkce přenosu obrazu po USB byl do této práce zapůjčen a použit.

Blok videopreprocesoru EZ-USB DATA byl připsán autorem této práce do souboru *videoprocesor.vhd*. Tento blok upravuje časování výstupních signálů CMOS senzoru tak, aby mohly být signály připojeny na vstupní paměť FIFO obvodu EZ-USB.

V rámci této práce byl ověřen přenos obrazu po USB pomocí asynchronní verze firmware do obvodu EZ-USB. Videosenzor (a propojení obvodů FPGA a EZ-USB) je však navržen tak, že půjde použít i synchronní verze firmware s tím, že bude muset být poupraven blok EZ-USB DATA videopreprocesoru.



Obr 8.11 Program pro zobrazení sledované scény pomocí USB

9. Přehled odvedené práce

Návrh videosenzoru byl časově náročný nejenom kvůli jeho konstrukci (návrh modulu zpracováno obrazového signálu a testování propojení obvodů), ale také kvůli tomu, že musela být nejprve nastudována problematika programování všech použitých obvodů a problematika konstrukce plošných spojů. Zde je uveden přehled jednotlivých kroků, které byly v průběhu návrhu a realizace videosenzoru vykonány.

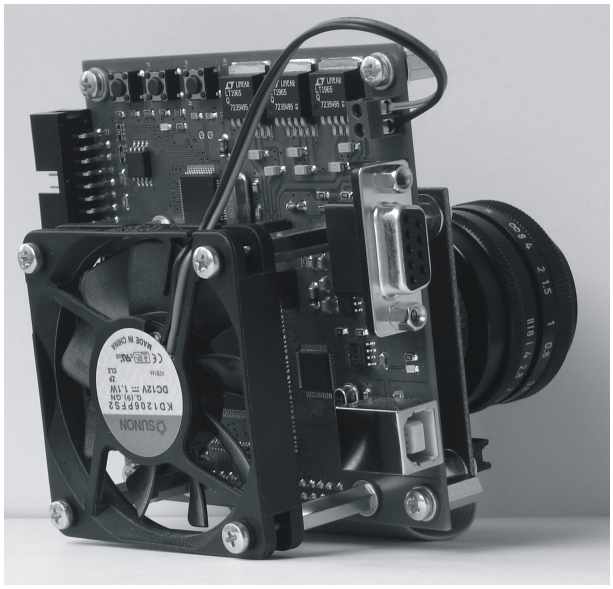
1. **Seznámení se systémy pro bezkontaktní měření:** Studium literatury (1) a (2). Nastudována problematika optické zobrazovací soustavy, vlastnosti a použití CCD a CMOS plošných snímačů a varianty systémů pro bezkontaktní měření.
2. **Seznámení s hradlovým polem FPGA:** Byla nastudována vnitřní architektura FPGA dle (4), použití blokových RAM FPGA dle (16) a způsob konfigurace FPGA dle (17) a (18).
3. **Programovací jazyk VHDL:** Byla nastudována problematika programování FPGA pomocí jazyka VHDL. Z tímto účelem byl již v bakalářské etapě zapsán předmět *Programovatelné součástky (34PRS)* a dále v magisterské etapě předmět *Aplikace hradlových polí (X38APH)*. To zahrnovalo studium literatury (8), (9) a (15), zejména návrh stavových automatů a synchronních obvodů.
4. **Studium architektury procesoru ARM7:** Seznámení s bloky a rozhraními procesoru. Nastavení komunikačních rozhraní pomocí speciálních funkčních registrů. Studium literatury (12) pro seznámení s programováním obvodu a (13) pro seznámení s architekturou obvodu.
5. **Studium architektury řadiče USB**: Seznámení s blokovým schématem obvodu EZ-USB dle literatury (11). Režimy použití vnitřní FIFO paměti obvodu, typy END POINTŮ a způsob programování.
6. **Studium problematiky konstrukce plošných spojů:** Za tímto účelem byl již během bakalářské etapy zapsán předmět *Metodika návrhu propojování součástek (34MPS)*. Během magisterské etapy, pak k návrhu modulu zpracování obrazového signálu přispělo také zapsání předmětu *Elektromagnetická kompatibilita přístrojů a systémů (X38EMK)*. Dále musel být nastudován způsob práce v programu OrCad 10.3 (návrh plošných spojů).

7. **Návrh vývojového modulu pro ARM7:** Zatímco pro seznámení s programováním hradlového pole FPGA byl použit vývojový modul firmy DIGILENT, pro seznámení a ověření komunikace ARM7-FPGA byl navržen vývojový modul s ARM7 LPC2148 dle kapitoly *3.1 Testování pomocí vývojových modulů*. Na tomto modulu se také testovala komunikace ARM7-SPI FLASH. Studium paměti FLASH bylo dle literatury (14).
8. **Návrh modulu zpracování obrazového signálu:** Po nastudování příslušné problematiky byl navržen uvedený modul. Jedna z časově nejnáročnějších etap. Pro všechny použité součástky muselo být navrženo v programu OrCad pouzdro (předpokládalo se totiž ruční pájení).
9. **Výroba modulu:** Osazení celého modulu bylo provedeno ručně autorem této práce. Některé součástky, např. obvod FPGA, který obsahuje 144 kontaktů, mají kontakty od sebe vzdáleny pouhých 12 milů. Osazení vyžaduje velkou zručnost.
10. **Napsán kód videopreprocesoru v jazyku VHDL:** Jazykem VHDL byl popsán videopreprocesor v FPGA. Použito bylo vývojové prostředí ISE Web Pack.
11. **Napsán řídící program pro ARM7:** Byl napsán řídící program do nadřazeného procesoru ARM7. Program zajišťuje konfiguraci FPGA a přenos naměřených dat do počítače.
12. **Ověření funkce:** Za účelem zobrazení naměřených hodnoty byly napsány, krátké programy v jazyku MATLAB viz. kapitola *8.7 Zobrazení naměřených dat*. Naměřené a uložené hodnoty měření vertikálních a horizontálních hran lze najít na doprovodném CD:

Mereni\...

10. Závěr

Tato diplomová práce se zabývala nejen návrhem **videopreprocesoru s FPGA**, ale také návrhem **modulu zpracování obrazového signálu**, na kterém je videopreprocesor osazen. Připojením tohoto modulu k modulu CMOS plošného senzoru (semestrální projekt, Jan Šedivý, 2003) vznikla malá kompaktní kamera pro bezkontaktní měření rozměrů objektu v reálném čase, která byla nazvána **VIDEOSENZOR** dle Obr 10.1.



Obr 10.1 Videosenzor

Za účelem nahrání konfiguračního souboru do FPGA je modul osazen nadřazeným procesorem ARM7 typu LPC2148 a pamětí SPI FLASH typu M25P80. Dle kapitoly *4 Konfigurace FPGA pomocí ARM7* byl navržen a realizován způsob jejich propojení s FPGA. Způsob propojení má tyto výborné vlastnosti: Minimum signálů na plošném spoji (rozhraní SPI pro přenos dat + řídící signály, celkem 8 signálů). Maximální využití přenosového kanálu, tj. po zapnutí napájení je FPGA nakonfigurován za 2s. V paměti FLASH je prostor pro **8 konfiguračních souborů**. Stejné signály (SPI) jsou po ukončení konfigurace použity pro komunikaci videopreprocesor – ARM7.

Modul disponuje mnoha periferiemi pro komunikaci s nadřazenými systémy. Obsahuje

rozhraní **USB** (obvod EZ-USB CY7C68013A) pro přenos obrazu do počítače v reálném čase. Přenos obrazu byl ověřen dle kapitoly *8.8 Zobrazení sledované scény pomocí rozhraní USB*. Dále obsahuje jednoduchou sériovou linku **RS232**, pro přenos naměřených hodnot do počítače, pro nastavení funkcí videopreprocesoru a pro aktualizaci FW modulu. V poslední řadě modul obsahuje jeden konektor s rozhraním **SPI** a dvěma **PWM** výstupy a jeden konektor s 16-ti univerzálními **IO** piny FPGA.

Do nadřazeného procesoru byl napsán program v jazyku C, který zajišťuje nahrání konfiguračního souboru z počítače do modulu. Nahrání konfigurace probíhá jednoduše pomocí terminálu sériové linky. Program v nadřazeném procesoru také odesílá po sériové lince naměřené hodnoty, které jsou na straně počítače zobrazeny pomocí krátkých programů v jazyku MATLAB.

Důležitou částí této práce byl návrh videopreprocesoru v FPGA typu Spartan 3 XC3S200 pomocí jazyka VHDL. Pro návrh bylo v FPGA využito 893 registrů z celkového počtu 3,840 tedy 23% a 1,420 čtyř-vstupé logiky z celkového počtu 3,840 tedy 36%. Blokových RAM bylo použito 100% tedy 288Kb.

Videopreprocesor obsahuje dle zadání diplomové práce základní metody pro bezkontaktní měření rozměrů objektu v reálném čase a navíc nad rámec zadání obsahuje metody vyhledávání optických hran ve směru sloupců a dále zpracování digitálního obrazového signálu s konvoluční maskou Laplace, Prewitt, „Mexický klobouk“. Zde byl velice efektivně a jednoduše vyřešen způsob načítání obrazové matice 8x8 pixelů s paralelním čtením, na kterou je v reálném čase prováděn výpočet konvoluce.

V poslední řadě byly do videopreprocesoru navrženy pomocí jazyka VHDL bloky pro komunikaci s nadřazeným procesorem a výstupní FIFO paměť 4Kx32b. Jedná se o blok sériové komunikace SPI a blok paralelní komunikace PI.

Seznam obrázků

Seznam tabulek

Seznam literatury

[1] Fischer, J.: *Optoelektronické senzory a videometrie*. Skripta ČVUT, Praha 2002.

[2] Říha, V.: Diplomová práce, ČVUT – FEL, Praha 2006.

[3] Martínek, Š.: Diplomová práce, ČVUT – FEL, Praha 2005.

[4] Xilinx: *Spartan-3 Generation FPGA User Guide*. Uživatelský manuál 2007.

[5] Nádvorník V.: Diplomová práce, ČVUT – FEL, Praha 2008.

[6] Souček P.: Bakalářská práce ČVUT – FEL. Praha 2007.

[7] Zima J.: Bakalářská práce ČVUT – FEL. Praha 2009.

[8] Hazdra P.: *VHSIC HDL (VHDL) pro syntézu IO*. Přednášky. Praha 2004.

[9] Musil V., Kolouch J., Prokop R.: *Návrh digitálních integrovaných obvodů a jazyk VHDL*. Brno 2002.

[10] National Semiconductor Corporation: *LM9617 Monochrome CMOS Image Sensor VGA 30 FPS*. Katalogový list 2000.

[11] Cypress: EZ-USB FX2LP USB Microcontroller. Katalogový list 2006.

[12] Philips: LPC2141/42/44/46/48 Microcontroller. Katalogový list 2005.

[13] Hitex: *LPC ARM book*. Příručka programování ARM LPC2100, 2006.

[14] ST: M25P80 serial Flash memory. Katalogový list 2005.

[15] Šťastný J: *Návrh specifických struktur na programovatelných hradlových polích*. ČVUT, Praha.

[16] Xilinx: *Using Block RAM in Spartan-3 FPGAs*. Aplikační poznámky 2003.

[17] Xilinx: *Spartan-3 Generation Configuration User Guide*. Aplikační poznámky 2007.

[18] Xilinx: *The Low-Cost, Efficient Serial Configuration of Spartan FPGAs*. Aplikační poznámky 1998.

Obsah přiloženého CD

\ARM_code\...	- programy v jazyku C do procesoru ARM7 (FW).
\Datasheet\...	- katalogové listy obvodů, manuály, dokumentace.
\EZUSB_code\...	- programy v jazyku C do obvodu EZ-USB (FW).
\FPGA_code\...	- kód VHDL do hradlového pole FPGA Spartan 3.
\Matlab_code\...	- programy v jazyku MATLAB pro PC.
\Moduly_dokumentace\...	- schémata, fólie a seznamy součástek navržených modulů.
\Orcad_projekty\...	- projekty v programu Orcad 10.3 navržených modulů.
\Programy\...	- podpůrné a vývojové programy pro PC.
\Text\...	- text diplomové práce s obrázky, fotkami, tabulkami, grafy.
\Mereni\...	- naměřené hodnoty pozic hran videosenzorem.

11. Příloha A (speciální funkční registry SFR)

CMPK: komparační konstanta															
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	**bit 7**	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
MSB															LSB

HEXA HODNOTA	VÝZNAM
0x0000 až 0x00FF	Registr pro uložení komparační konstanty. Používá se při měření obsahu světlých a tmavých objektů. Pokud jasová funkce překročí tuto hodnotu je objekt považován za světlý. V opačném případě je objekt považován za tmavý. Bity 15-8 jsou rezerva a jejich nastavení nemá žádný význam.

Tabulka 3: Registr CMPK

DIFFK: diferenční konstanta															
bit 15	bit 14	bit 13	bit 12	**bit 11**	**bit 10**	**bit 9**	**bit 8**	**bit 7**	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
MSB															LSB

HEXA HODNOTA	VÝZNAM
0x0000 až 0x03CF	Registr pro uložení diferenční konstanty. Používá se při měření pozice vertikálních a horizontálních hran, měření vertikálního a horizontálního rozměru objektu a jeho středu. Bity 15-12 jsou rezerva a jejich nastavení nemá žádný význam.

Tabulka 4: Registr DIFFK

I1: levý horní okraj okna - řádek															
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	**bit 8**	**bit 7**	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
MSB															LSB

HEXA HODNOTA	VÝZNAM
0x0000 až 0x0200	Registr pro uložení hodnoty řádku levé horní souřadnice okna. Oknem se rozumí část plošného obrazu, která se použije při zpracování obrazového signálu. Bity 15-9 jsou rezerva a jejich nastavení nemá žádný význam.

Tabulka 5: Registr I1

J1: levý horní okraj okna - sloupec															
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	**bit 9**	**bit 8**	**bit 7**	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
MSB															LSB

HEXA HODNOTA	VÝZNAM
0x0000 až 0x0400	Registr pro uložení hodnoty sloupce levé horní souřadnice okna. Oknem se rozumí část plošného obrazu, která se použije při zpracování obrazového signálu. Bity 15-10 jsou rezerva a jejich nastavení nemá žádný význam.

Tabulka 6: Registr J1

I2: pravý dolní okraj okna - řádek															
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	**bit 8**	**bit 7**	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
MSB															LSB

HEXA HODNOTA	VÝZNAM
0x0000 až 0x0200	Registr pro uložení hodnoty řádku pravé dolní souřadnice okna. Oknem se rozumí část plošného obrazu, která se použije při zpracování obrazového signálu. Bity 15-9 jsou rezerva a jejich nastavení nemá žádný význam.

Tabulka 7: Registr I2

J2: pravý dolní okraj okna - sloupec															
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	**bit 9**	**bit 8**	**bit 7**	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
MSB															LSB

HEXA HODNOTA	VÝZNAM
0x0000 až 0x0400	Registr pro uložení hodnoty sloupce pravé dolní souřadnice okna. Oknem se rozumí část plošného obrazu, která se použije při zpracování obrazového signálu. Bity 15-10 jsou rezerva a jejich nastavení nemá žádný význam.

Tabulka 8: Registr J2

MODE: výběr metody zpracování obrazového signálu															
M2	**M1**	**M0**	**FI**	**D3**	**D2**	**D1**	**D0**	**F7**	**F6**	**F5**	**F4**	**F3**	**F2**	**F1**	**F0**
MSB															LSB

HEXA HOD.	**VÝZNAM**
	F[7:0]: výběr samotné metody
F[7:0] = 0x01	**Uložení vybrané části obrazu** dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti.
F[7:0] = 0x02	Uložení vybrané části obrazu po průchodu konvoluční maskou „**Mexický klobouk**“ dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti. Hodnota výstupních pixelů je dělena 256 a prahována hodnotou 16.
F[7:0] = 0x03	Uložení vybrané části obrazu po průchodu konvoluční maskou **Prewitt** dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti. Hodnota výstupních pixelů je dělena 16 a prahována hodnotou 16.
F[7:0] = 0x04	Uložení vybrané části obrazu po průchodu konvoluční maskou **Laplace** dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti. Hodnota výstupních pixelů je dělena 16 a prahována hodnotou 16.
F[7:0] = 0x05	Uložení **horizontálního diferenčního signálu *s0dif1*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti. Ukládá se spodních 8 bitů a poslední bit je vždy nastaven na '1'.
F[7:0] = 0x06	Uložení **horizontálního diferenčního signálu s0dif2** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti. Ukládá se spodních 8 bitů a poslední bit je vždy nastaven na '1'.
F[7:0] = 0x07	Uložení **horizontálního diferenčního signálu *s0dif3*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti. Ukládá se spodních 8 bitů a poslední bit je vždy nastaven na '1'.
F[7:0] = 0x08	Uložení **horizontálního diferenčního signálu s0dif4** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti. Ukládá se spodních 8 bitů a poslední bit je vždy nastaven na '1'.
F[7:0] =	Uložení **vertikálního diferenčního signálu *s0dif1v*** vybrané části obrazu dle

0x09	nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti. Ukládá se spodních 8 bitů a poslední bit je vždy nastaven na '1'.
F[7:0] = 0x0A	Uložení **vertikálního diferenčního signálu *s0dif2v*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti. Ukládá se spodních 8 bitů a poslední bit je vždy nastaven na '1'.
F[7:0] = 0x0B	Uložení **vertikálního diferenčního signálu *s0dif3v*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti. Ukládá se spodních 8 bitů a poslední bit je vždy nastaven na '1'.
F[7:0] = 0x0C	Uložení **vertikálního diferenčního signálu *s0dif4v*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti. Ukládá se spodních 8 bitů a poslední bit je vždy nastaven na '1'.
F[7:0] = 0x20	Uložení **horizontálního diferenčního signálu *s0dif1*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti.
F[7:0] = 0x21	Uložení **horizontálního diferenčního signálu *s0dif2*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti.
F[7:0] = 0x22	Uložení **horizontálního diferenčního signálu *s0dif3*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti.
F[7:0] = 0x23	Uložení **horizontálního diferenčního signálu *s0dif4*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti.
F[7:0] = 0x24	Uložení **vertikálního diferenčního signálu *s0dif1v*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti.
F[7:0] = 0x25	Uložení **vertikálního diferenčního signálu *s0dif2v*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti.
F[7:0] = 0x26	Uložení **vertikálního diferenčního signálu *s0dif3v*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti.
F[7:0] = 0x27	Uložení **vertikálního diferenčního signálu *s0dif4v*** vybrané části obrazu dle nastavení okna o souřadnicích [i1, j1] a [i2, j2] do výstupní FIFO paměti.
F[7:0] = 0x30	**Hledání horizontálních hran** pro jejichž detekci se využívá diferenční signál ***s0dif1***.
F[7:0] = 0x31	**Hledání horizontálních hran** pro jejichž detekci se využívá diferenční signál ***s0dif2***.
F[7:0] = 0x32	**Hledání horizontálních hran** pro jejichž detekci se využívá diferenční signál ***s0dif3***.
F[7:0] = 0x33	**Hledání horizontálních hran** pro jejichž detekci se využívá diferenční signál ***s0dif4***.

F[7:0] = 0x40	**Hledání vertikálních hran** pro jejichž detekci se využívá diferenční signál ***s0dif1***.
F[7:0] = 0x41	**Hledání vertikálních hran** pro jejichž detekci se využívá diferenční signál ***s0dif2***.
F[7:0] = 0x42	**Hledání vertikálních hran** pro jejichž detekci se využívá diferenční signál ***s0dif3***.
F[7:0] = 0x43	**Hledání vertikálních hran** pro jejichž detekci se využívá diferenční signál ***s0dif4***.
F[7:0] = 0x50	**Hledání všech hran** pro jejichž detekci se využívá diferenční signál ***s0dif1*** a ***s0dif1v***.
F[7:0] = 0x51	**Hledání všech hran** pro jejichž detekci se využívá diferenční signál ***s0dif2*** a ***s0dif2v***.
F[7:0] = 0x52	**Hledání všech hran** pro jejichž detekci se využívá diferenční signál ***s0dif3*** a ***s0dif3v***.
F[7:0] = 0x53	**Hledání všech hran** pro jejichž detekci se využívá diferenční signál ***s0dif4*** a ***s0dif4v***.
F[7:0] = 0x60	**Určování horizontálního rozměru** dle metody nalezení hrany vybrané pomocí bitů D[3:0].
F[7:0] = 0x70	**Určování vertikálního rozměru** dle metody nalezení hrany vybrané pomocí bitů D[3:0].
F[7:0] = 0x80	**Určování horizontálního středu** dle metody nalezení hrany vybrané pomocí bitů D[3:0].
F[7:0] = 0x90	**Určování vertikálního středu** dle metody nalezení hrany vybrané pomocí bitů D[3:0].
F[7:0] = 0xA0	Určování obsahu světlých objektů.
F[7:0] = 0xB0	Určování obsahu tmavých objektů.
F[7:0] = 0xC0	**Určování horizontálního a vertikálního středu** dle metody nalezení hrany vybrané pomocí bitů D[3:0].
	D[3:0]: výběr metody hledání hran při měření rozměru a středu
D[3:0] = 0x1	**Hledání hran** pro jejichž detekci se využívá diferenční signál ***s0dif1*** a ***s0dif1v***.
D[3:0] = 0x2	**Hledání hran** pro jejichž detekci se využívá diferenční signál ***s0dif2*** a ***s0dif2v***.

D[3:0] = 0x3	**Hledání hran** pro jejichž detekci se využívá diferenční signál ***s0dif3*** a ***s0dif3v***.
D[3:0] = 0x4	**Hledání hran** pro jejichž detekci se využívá diferenční signál ***s0dif4*** a ***s0dif4v***.
	FI: způsob načítání dat do FIFO
FI = '0'	Do FIFO se načtou data z jednoho snímku (pixely, hrany, rozměr atd.) a čeká se na vyčtení dat nadřazeným procesorem. Po vyprázdnění FIFO se automaticky načítají data nová.
FI = '1'	Do FIFO se načtou data z několika snímků jdoucích za sebou (pixely, hrany, rozměr atd.) dokud naplnění FIFO nepřesáhne ½ a čeká se na vyčtení dat nadřazeným procesorem. Po vyprázdnění FIFO pod ½ se automaticky načítají data nová.
	M[2:0]: volba obrazové mřížky při hledání hran
M[2:0] = '000'	Mřížka vypnuta.
M[2:0] = '001'	Hrany se hledají v každém **8. řádku a sloupci.**
M[2:0] = '010'	Hrany se hledají v každém **16. řádku a sloupci.**
M[2:0] = '011'	Hrany se hledají v každém **32. řádku a sloupci.**
M[2:0] = '100'	Hrany se hledají v každém **64. řádku a sloupci.**
M[2:0] = '101'	Hrany se hledají v každém **128. řádku a sloupci.**
M[2:0] = '110'	Hrany se hledají v každém **256. řádku a sloupci.**

Tabulka 9: Registr MODE

12. Příloha B (IO konektory modulu)

CMOS SENSOR LM9617 KONEKTOR							
ARM	**FPGA (UCF)**	**CMOS**	**PIN**	**PIN**	**CMOS**	**FPGA (UCF)**	**ARM**
	P1	EXTSYNC	**1**	**2**	OE	P2	
P0.3/SDA0		SDA	**3**	**4**	SCLK		P0.2/SCL0
	P141	SNAPSHOT	**5**	**6**	RESETB	P140	
	P137	PWDN	**7**	**8**	HSYNC	P135	
	P132	VSYNC	**9**	**10**	PCLK	P127	
GND	GND	GND	**11**	**12**	GND	GND	GND
GND	GND	GND	**13**	**14**	MCLK	P128	
GND	GND	GND	**15**	**16**	GND	GND	GND
		NC	**17**	**18**	NC		
	P131	D0	**19**	**20**	D1	P130	
	P129	D2	**21**	**22**	D3	P123	
	P122	D4	**23**	**24**	D5	P119	
	P118	D6	**25**	**26**	D7	P116	
	P113	D8	**27**	**28**	D9	P112	
VCCO 3,3V	VCCO 3,3V	VDD IN	**29**	**30**	VDD IN	VCCO 3,3V	VCCO 3,3V

Tabulka 10: Konektor pro CMOS SENSOR LM9617

IO ARM KONEKTOR			
ARM	**PIN**	**PIN**	**ARM**
VCCO 3,3V	**1**	**2**	GND
GND	**3**	**4**	P0.8/PWM4
GND	**5**	**6**	P0.21/PWM5
GND	**7**	**8**	GND
GND	**9**	**10**	P0.17/SCK1
GND	**11**	**12**	P0.18/MISO1
GND	**13**	**14**	P0.19/MOSI1
GND	**15**	**16**	P0.20/SSEL1

Tabulka 11: IO konektor procesoru ARM7

IO FPGA KONEKTOR			
FPGA	**PIN**	**PIN**	**FPGA**
VCCO 3,3V	**1**	**2**	GND
P90	**3**	**4**	P89
P93	**5**	**6**	P92
P96	**7**	**8**	P95
P98	**9**	**10**	P97
GND	**11**	**12**	GND
P99	**13**	**14**	P100
P102	**15**	**16**	P103
P104	**17**	**18**	P105
P107	**19**	**20**	P108

Tabulka 12: IO konektor obvodu FPGA

13. Příloha C (tlačítka, led a oscilátor na modulu)

LED FPGA
P87

Tabulka 13: Uživatelská led dioda obvodu FPGA

TLAČÍTKO FPGA
P125

Tabulka 14: Uživatelské tlačítko obvodu FPGA

LED ARM
P0.31

Tabulka 15: Uživatelská led dioda procesoru ARM7

TLAČÍTKO ARM
P0.30

Tabulka 16: Uživatelské tlačítko procesoru ARM7

KRISTALOVÝ OSCILÁTOR
P124

Tabulka 17: Zdroj hodinového kmitočtu obvodu FPGA

14. Příloha D (propojení pinů obvodů modulu)

PROPOJENÍ PINŮ FX2PL - FPGA	
FX2LP	**FPGA**
PB0/FD0	P35
PB1/FD1	P33
PB2/FD2	P32
PB3/FD3	P31
PB4/FD4	P30
PB5/FD5	P28
PB6/FD6	P27
PB7/FD7	P26
PA0/INT0	P21
PA1/INT1	P20
PA2/SLOE	P18
PA3/WU2	P17
PA4/FIFOADR0	P15
PA5/FIFOADR1	P14
PA6/PKTEND	P13
PA7/FLAGD/SLCS	P12
PD0/FD8	P8
PD1/FD9	P7
CTL0/FLAGA	P25
CTL1/FLAGB	P24
CTL2/FLAGC	P23
RDY0/SLRD	P5
RDY1/SLWR	P4
CLKOUT/PE1/T1OUT	P6
IFCLK/PE0/T0OUT	P36
RESET	P11
WAKEUP	P10

Tabulka 18: Propojení pinů obvodu FPGA a FX2PL

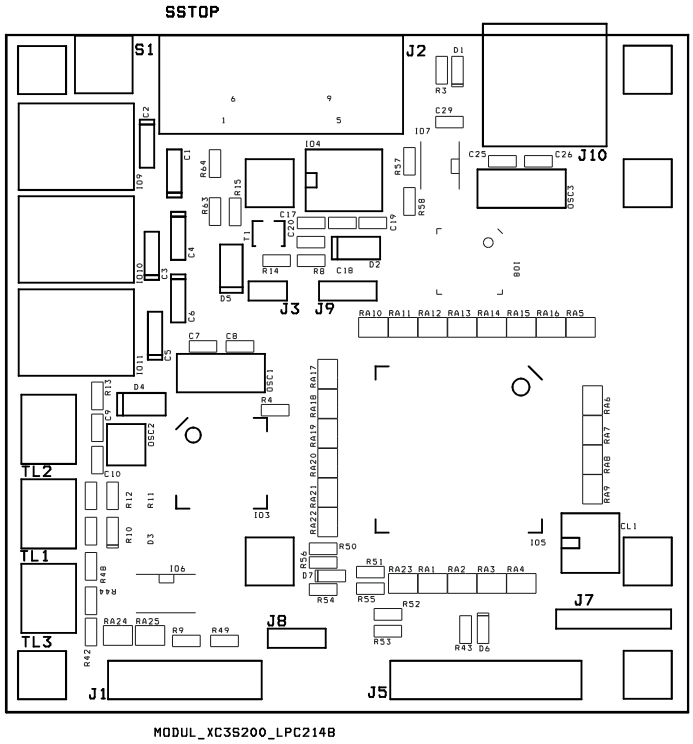
PROPOJENÍ PINŮ ARM - FPGA	
ARM	**FPGA**
SPI0	
P0.4/SCK0	P74
P0.5/MISO0	P70
P0.6/MOSI	P65
PI	
P1.16	P79
P1.17	P80
P1.18	P83
P1.19	P84
P1.20	P47
P1.21	P52
P1.22	P53
P1.23	P60
P1.24	P68
P1.25	P73
P1.26	P77
P1.27	P85
P1.28	P40
P1.29	P44
P1.30	P46
P1.31	P78
FLAGS	
P0.9	P56 (gclk1)
P0.10	P63
P0.11	P59
P0.13	P55 (gclk0)
P0.15	P51
P0.16	P50
P0.23	P41
P0.25	P82
P0.29	P58

Tabulka 19: Propojení pinů obvodu FPGA a procesoru ARM7

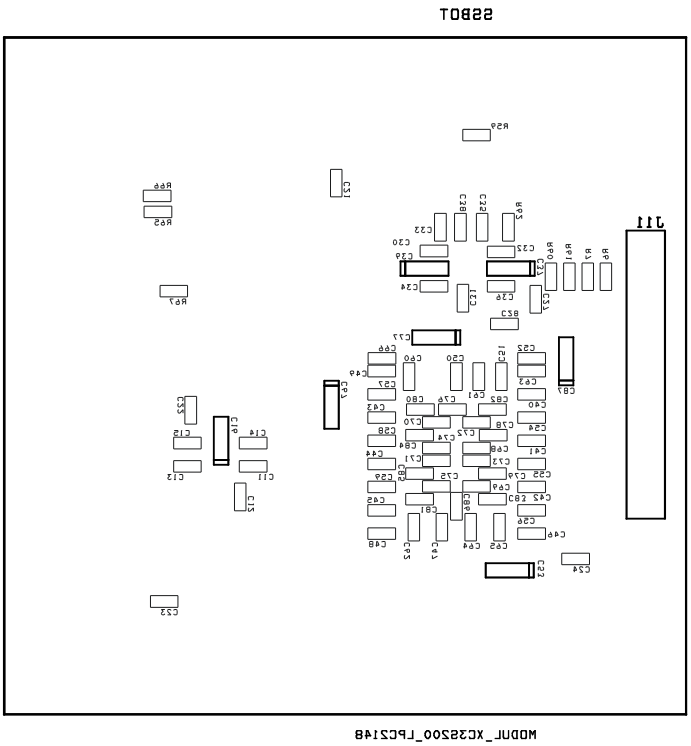
PROPOJENÍ PINŮ ARM – FPGA pouze signály pro konfiguraci FPGA	
ARM	**FPGA**
SPI0	
P0.4/SCK0	CCLK
P0.6/MOSI0	DIN
Řídící konfigurační signály	
P0.22	DONE
P0.28	PROG_B
P0.29	INIT_B

Tabulka 20: Propojení pinů pro konfiguraci FPGA pomocí ARM7

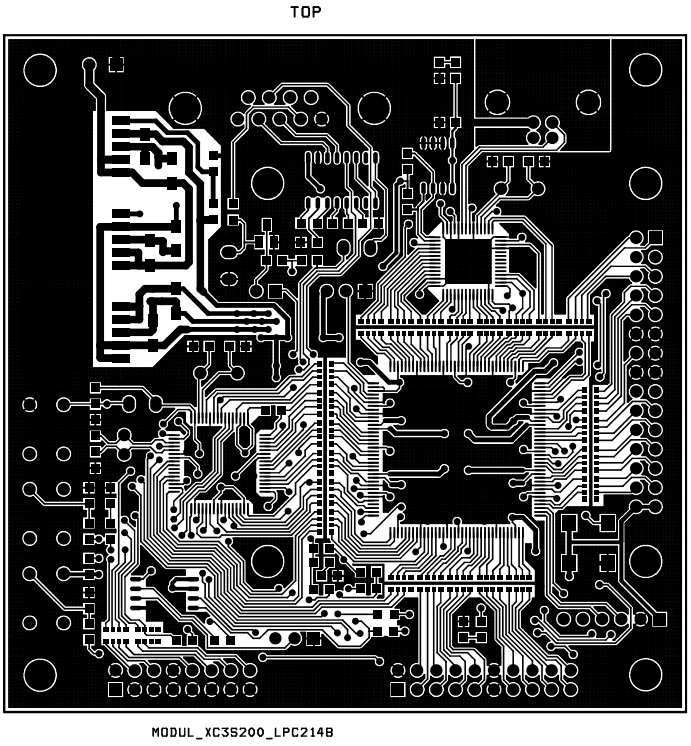
15. Příloha E (fólie plošných spojů modulu)



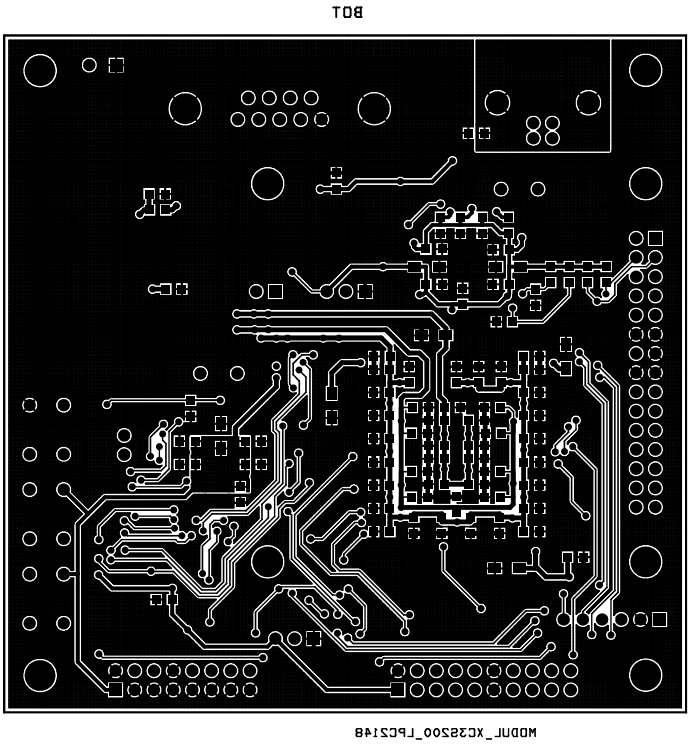
Obr 15.1 Potisk – vrstva SSTOP



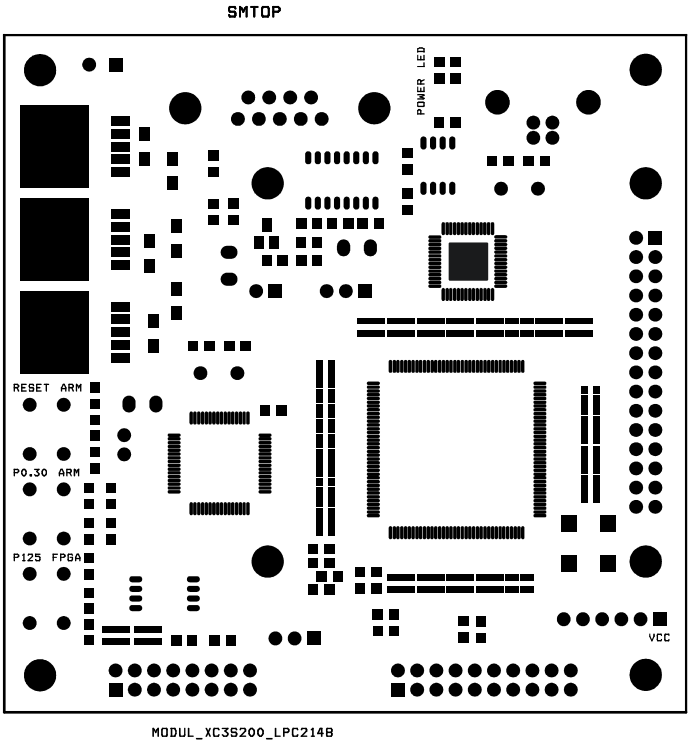
Obr 15.2 Potisk – vrstva SSBOT



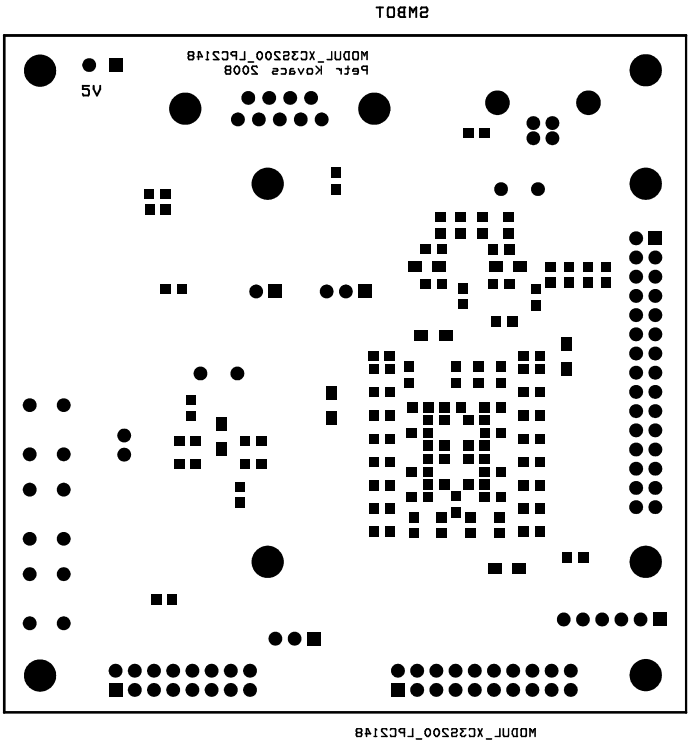
Obr 15.3 Spoje – vrstva TOP



Obr 15.4 Spoje – vrstva BOT



Obr 15.5 Maska – vrstva SMTOP



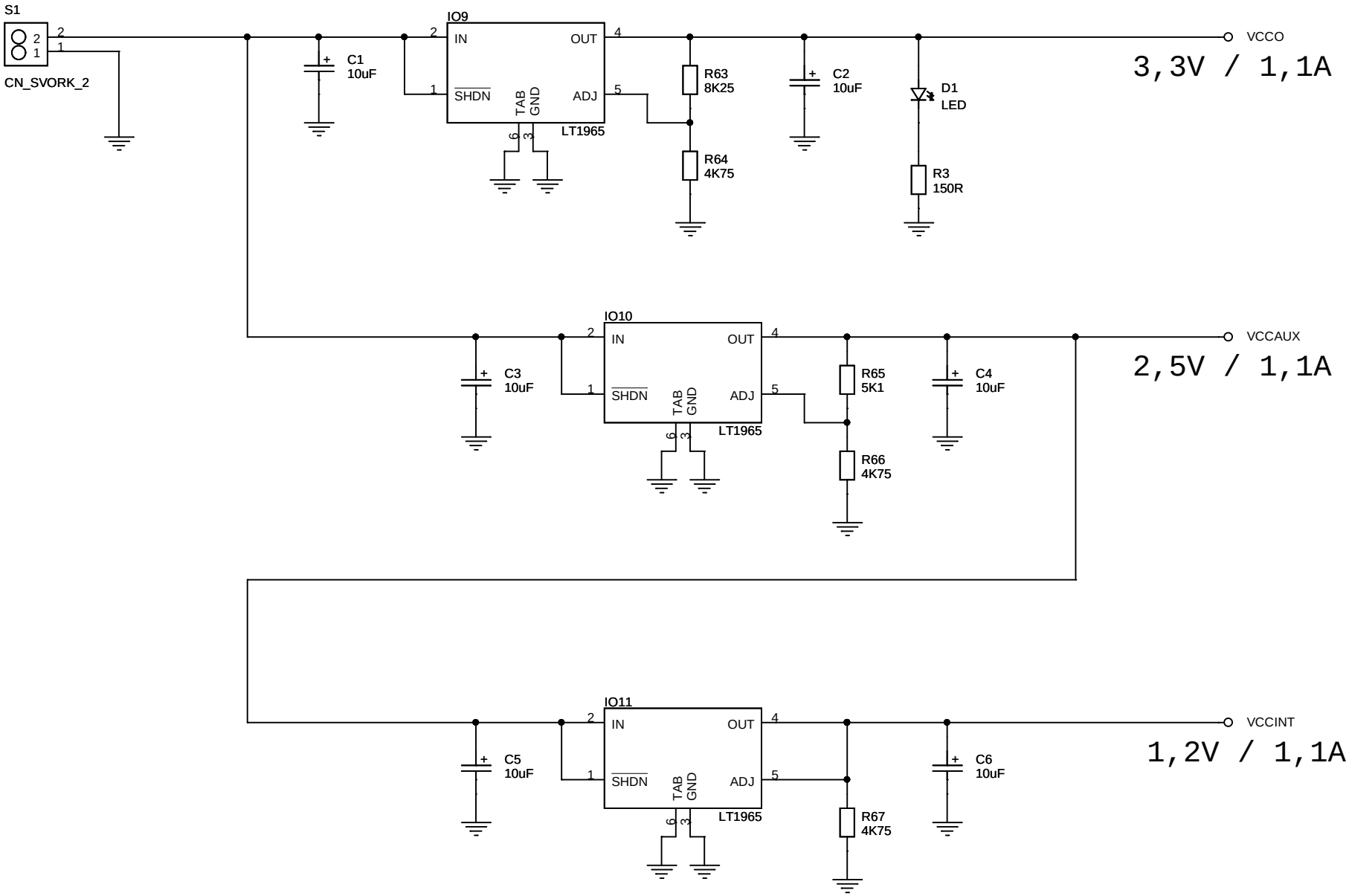
Obr 15.6 Maska – vrstva SMBOT

16. Příloha F (seznam součástek)

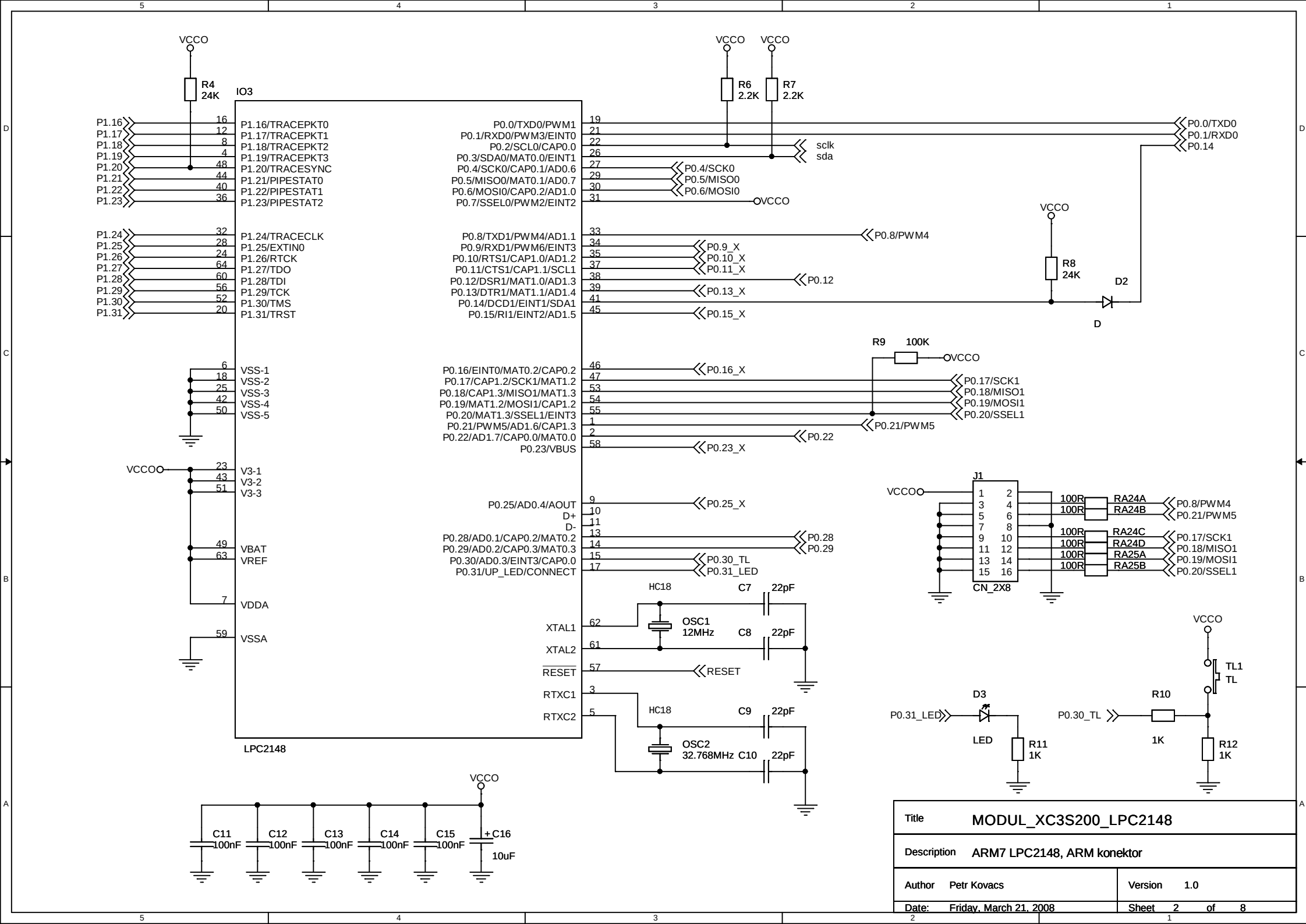
počet	označení	hodnota	pouzdro	poznámka
1	CL1	JF PCB 050.000000MHz	SG-8002JF	krystalový oscilátor
13	C1,C2,C3,C4,C5,C6,C16, C37,C39,C53,C67,C77,C87	10uF	A	tantal
6	C7,C8,C9,C10,C25,C26	22pF	0805	
12	C11,C12,C13,C14,C15,C17, C18,C19,C20,C21,C22,C23	100nF	0805	
32	C24,C27,C28,C29,C30,C31, C32,C33,C34,C35,C36,C38, C54,C55,C56,C57,C58,C59, C60,C61,C62,C63,C64,C65, C72,C73,C74,C75,C82,C83, C84,C85	47nF	0805	
20	C40,C41,C42,C43,C44,C45, C46,C47,C48,C49,C50,C51, C68,C69,C70,C71,C78,C79, C80,C81	10nF	0805	
4	C52,C66,C76,C86	0.47uF	0805	
4	D1,D3,D6,D7	LED červená	0805	
3	D2,D4,D5	1N4007	SOD87	
1	IO3	LPC2148	LQFP64	ARM7
1	IO4	MAX3232	SOIC16	
1	IO5	XC3S200	TQ144	FPGA
1	IO6	M25P80	SO8W	SPI flash
1	IO7	M24C02	S08	EEPROM
1	IO8	CY7C68013A	LF56A	USB řadič
3	IO9,IO10,IO11	LT1965	DDPAK	
1	J1	2X8	kolíková lišta	
1	J3	1X2	kolíková lišta	
1	J5	2X10	kolíková lišta	
1	J7	1X6	kolíková lišta	
2	J8,J9	1X3	kolíková lišta	
1	J11	2X15	kolíková lišta	
1	J2		cannon 9 female	
1	J10		USB-B	
1	OSC1	12MHz	HC49/U3H	oscilátor
1	OSC2	32.768MHz	MTF32	oscilátor
1	OSC3	24MHz	HC49/U3H	oscilátor

30	RA1,RA2,RA3,RA4,RA5,RA6, RA7,RA8,RA9,RA10,RA11, RA12,RA13,RA14,RA15,RA16, RA17,RA18,RA19,RA20,RA21, RA22,RA23,RA24,RA25,R49, R52,R53,R54,R55	100R	1206	odporová síť
1	R3	150R	0805	
2	R4,R8	24K	0805	
4	R6,R7,R57,R58	2.2K	0805	
4	R9,R48,R60,R61	100K	0805	
6	R10,R11,R12,R42,R43,R44	1K	0805	
1	R13	47k	0805	
1	R14	22k	0805	
1	R15	33k	0805	
1	R50	270R	0805	
1	R51	4.7K	0805	
1	R56	390R	0805	
2	R59,R62	10	0805	
1	R63	8K25	0805	
3	R64,R66,R67	4K75	0805	
1	R65	5K1	0805	
1	S1	1X2	svorkovnice 3,5 mm	
3	TL1,TL2,TL3	TL	mikrotlačítko 4 piny	
1	T1	BC817	SOT23	NPN tranzistor

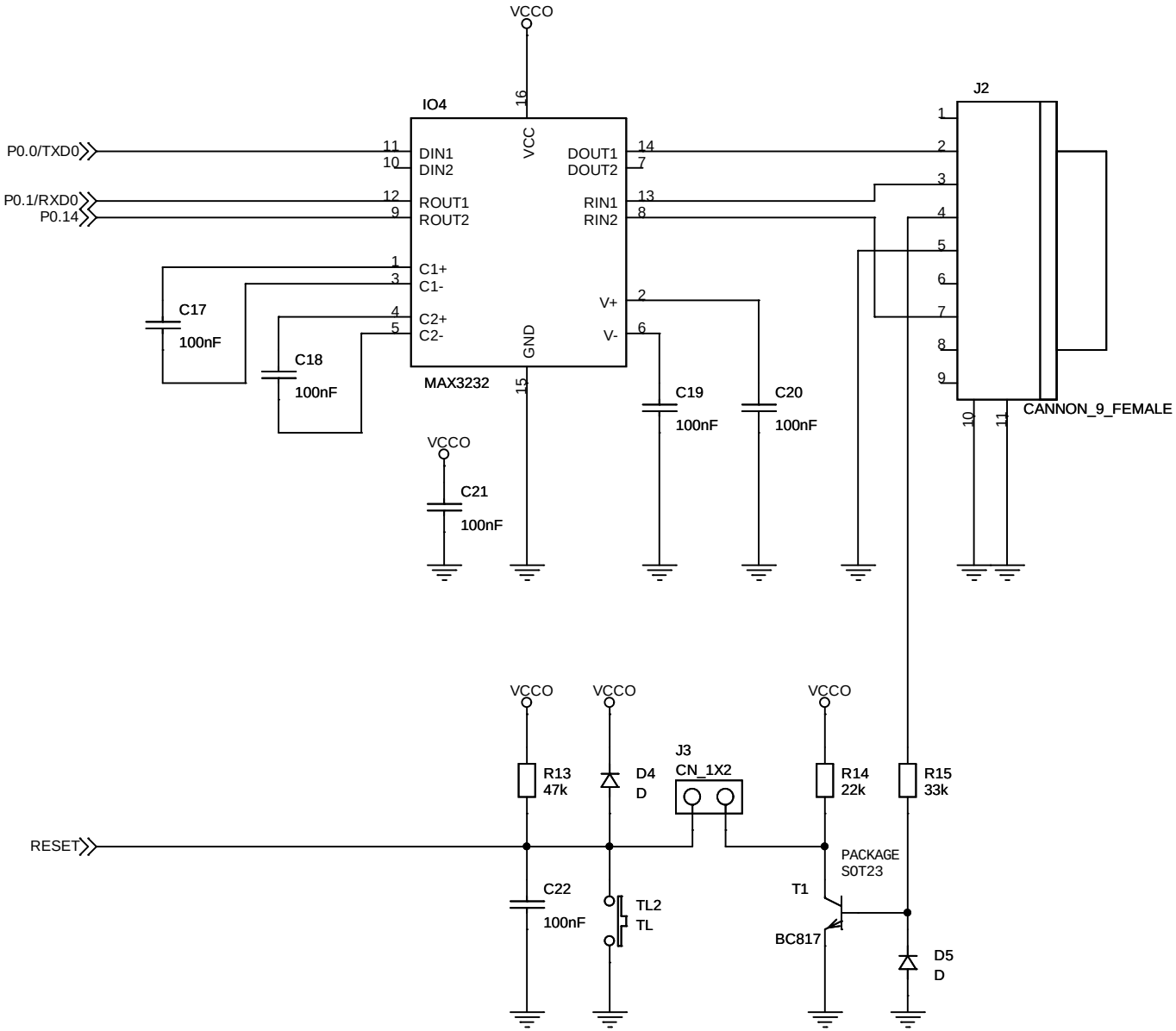
17. Příloha G (schéma modulu)



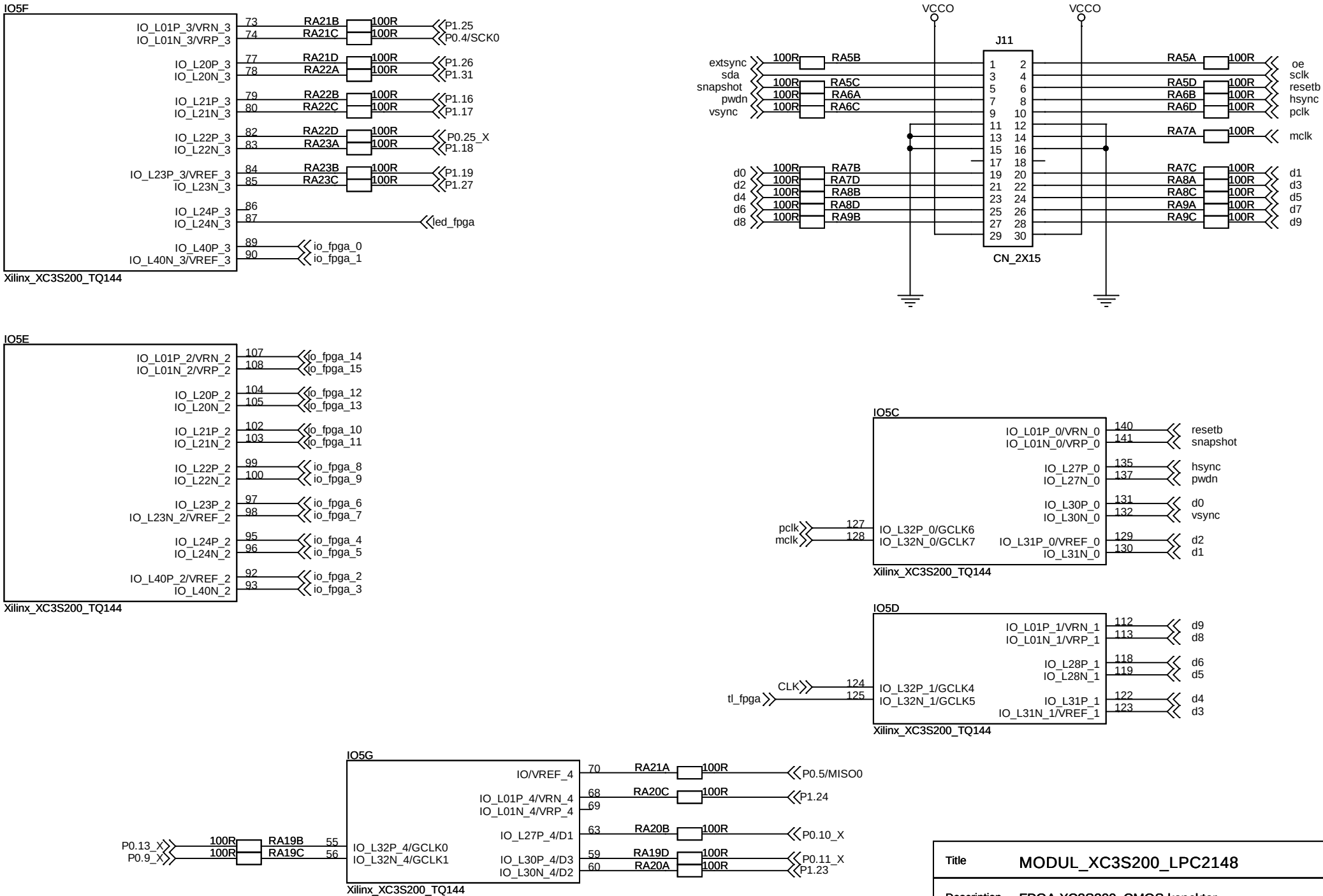
Title	MODUL_XC3S200_LPC2148
Description	Napajeni
Author: Petr Kovacs	Version: 1.0
Date: Friday, March 21, 2008	Sheet 1 of 8

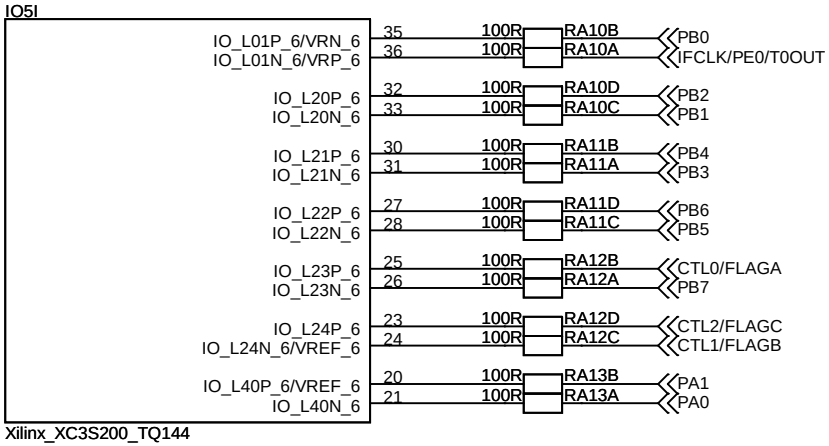


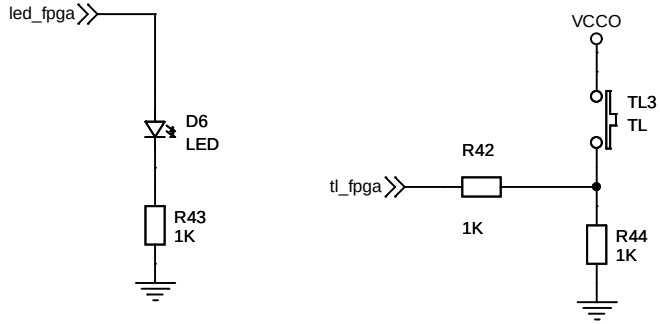
Title	MODUL_XC3S200_LPC2148
Description	ARM7 LPC2148, ARM konektor
Author: Petr Kovacs	Version: 1.0
Date: Friday, March 21, 2008	Sheet 2 of 8

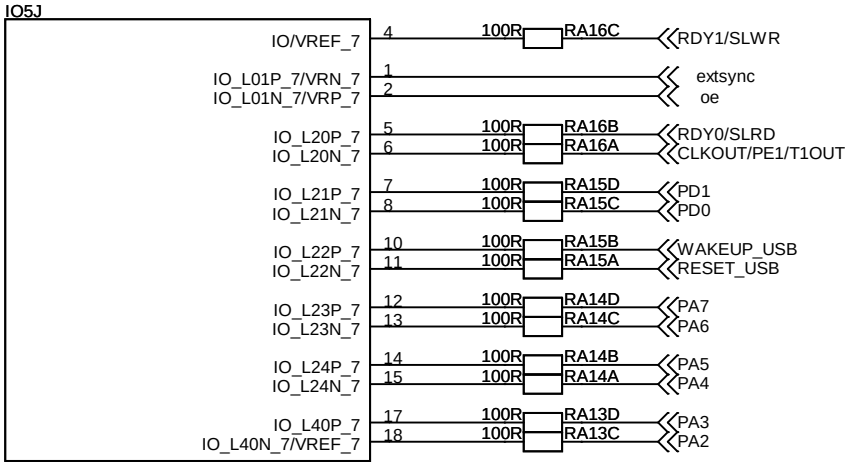


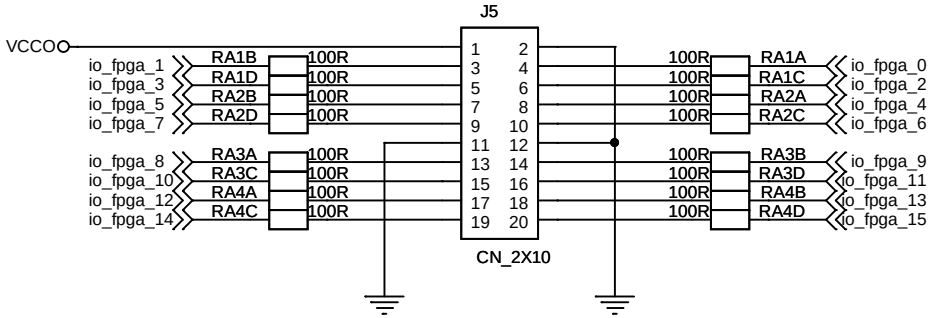
Title	MODUL_XC3S200_LPC2148
Description	RS232 - seriovy port, reset ARM7
Author: Petr Kovacs	Version: 1.0
Date: Friday, March 21, 2008	Sheet 3 of 8

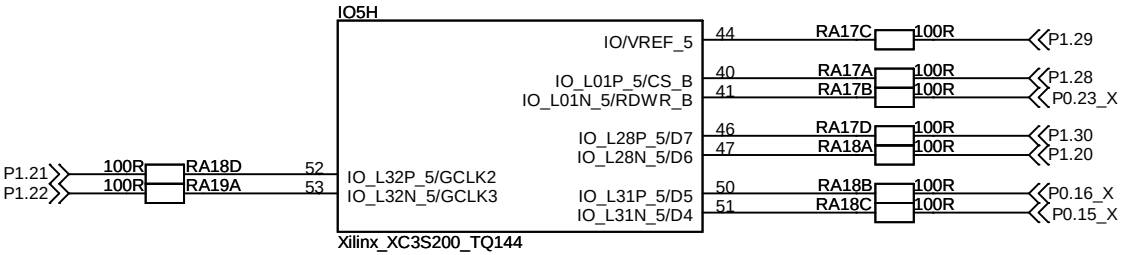

IO5F
IO_L01P_3/VRN_3
IO_L01N_3/VRP_3
IO_L20P_3
IO_L20N_3
IO_L21P_3
IO_L21N_3
IO_L22P_3
IO_L22N_3
IO_L23P_3/VREF_3
IO_L23N_3
IO_L24P_3
IO_L24N_3
IO_L40P_3
IO_L40N_3/VREF_3
73
74
77
78
79
80
82
83
84
85
86
87
89
90
RA21B
RA21C
RA21D
RA22A
RA22B
RA22C
RA22D
RA23A
RA23B
RA23C
100R
P1.25
P0.4/SCK0
P1.26
P1.31
P1.16
P1.17
P0.25_X
P1.18
P1.19
P1.27
led_fpga
io_fpga_0
io_fpga_1
Xilinx_XC3S200_TQ144
VCCO
VCCO
J11
CN_2X15
extsync
sda
snapshot
pwdn
vsync
RA5B
RA5C
RA6A
RA6C
RA5A
RA5D
RA6B
RA6D
RA7A
oe
sclk
resetb
hsync
pclk
mclk
d0
d2
d4
d6
d8
RA7B
RA7D
RA8B
RA8D
RA9B
RA7C
RA8A
RA8C
RA9A
RA9C
d1
d3
d5
d7
d9
IO5E
IO_L01P_2/VRN_2
IO_L01N_2/VRP_2
IO_L20P_2
IO_L20N_2
IO_L21P_2
IO_L21N_2
IO_L22P_2
IO_L22N_2
IO_L23P_2
IO_L23N_2/VREF_2
IO_L24P_2
IO_L24N_2
IO_L40P_2/VREF_2
IO_L40N_2
107
108
104
105
102
103
99
100
97
98
95
96
92
93
io_fpga_14
io_fpga_15
io_fpga_12
io_fpga_13
io_fpga_10
io_fpga_11
io_fpga_8
io_fpga_9
io_fpga_6
io_fpga_7
io_fpga_4
io_fpga_5
io_fpga_2
io_fpga_3
Xilinx_XC3S200_TQ144
IO5C
IO_L01P_0/VRN_0
IO_L01N_0/VRP_0
IO_L27P_0
IO_L27N_0
IO_L30P_0
IO_L30N_0
IO_L32P_0/GCLK6
IO_L32N_0/GCLK7
IO_L31P_0/VREF_0
IO_L31N_0
pclk
mclk
127
128
140
141
135
137
131
132
129
130
resetb
snapshot
hsync
pwdn
d0
vsync
d2
d1
Xilinx_XC3S200_TQ144
IO5D
IO_L01P_1/VRN_1
IO_L01N_1/VRP_1
IO_L28P_1
IO_L28N_1
IO_L32P_1/GCLK4
IO_L32N_1/GCLK5
IO_L31P_1
IO_L31N_1/VREF_1
CLK
tl_fpga
124
125
112
113
118
119
122
123
d9
d8
d6
d5
d4
d3
Xilinx_XC3S200_TQ144
IO5G
IO/VREF_4
IO_L01P_4/VRN_4
IO_L01N_4/VRP_4
IO_L27P_4/D1
IO_L30P_4/D3
IO_L30N_4/D2
IO_L32P_4/GCLK0
IO_L32N_4/GCLK1
70
68
69
63
59
60
55
56
RA21A
RA20C
RA20B
RA19D
RA20A
RA19B
RA19C
P0.5/MISO0
P1.24
P0.10_X
P0.11_X
P1.23
P0.13_X
P0.9_X
Xilinx_XC3S200_TQ144
Title
MODUL_XC3S200_LPC2148
Description
FPGA XC3S200, CMOS konektor
Author
Petr Kovacs
Version
1.0
Date:
Friday, March 21, 2008
Sheet
4
of
8

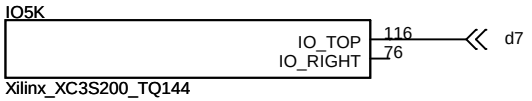




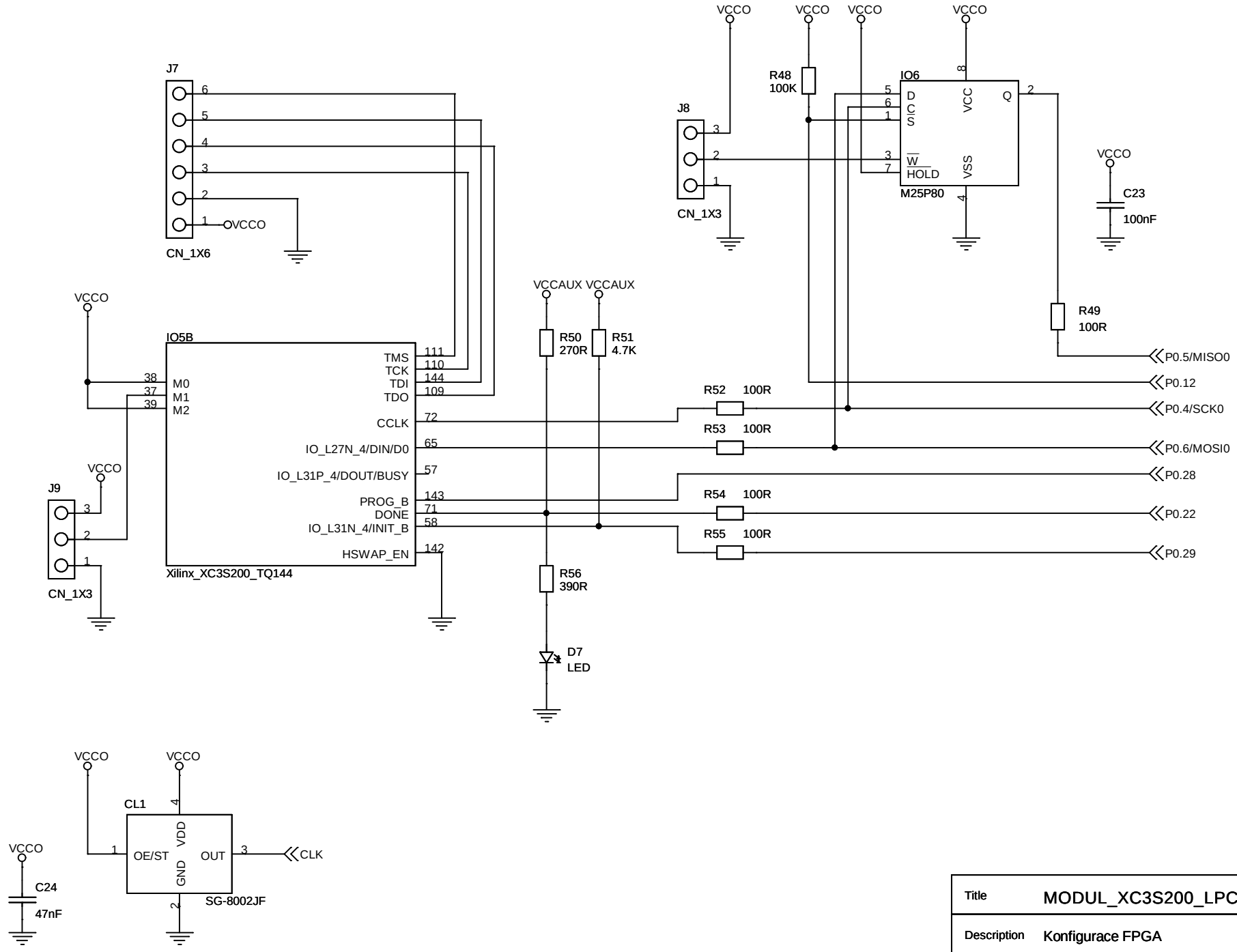




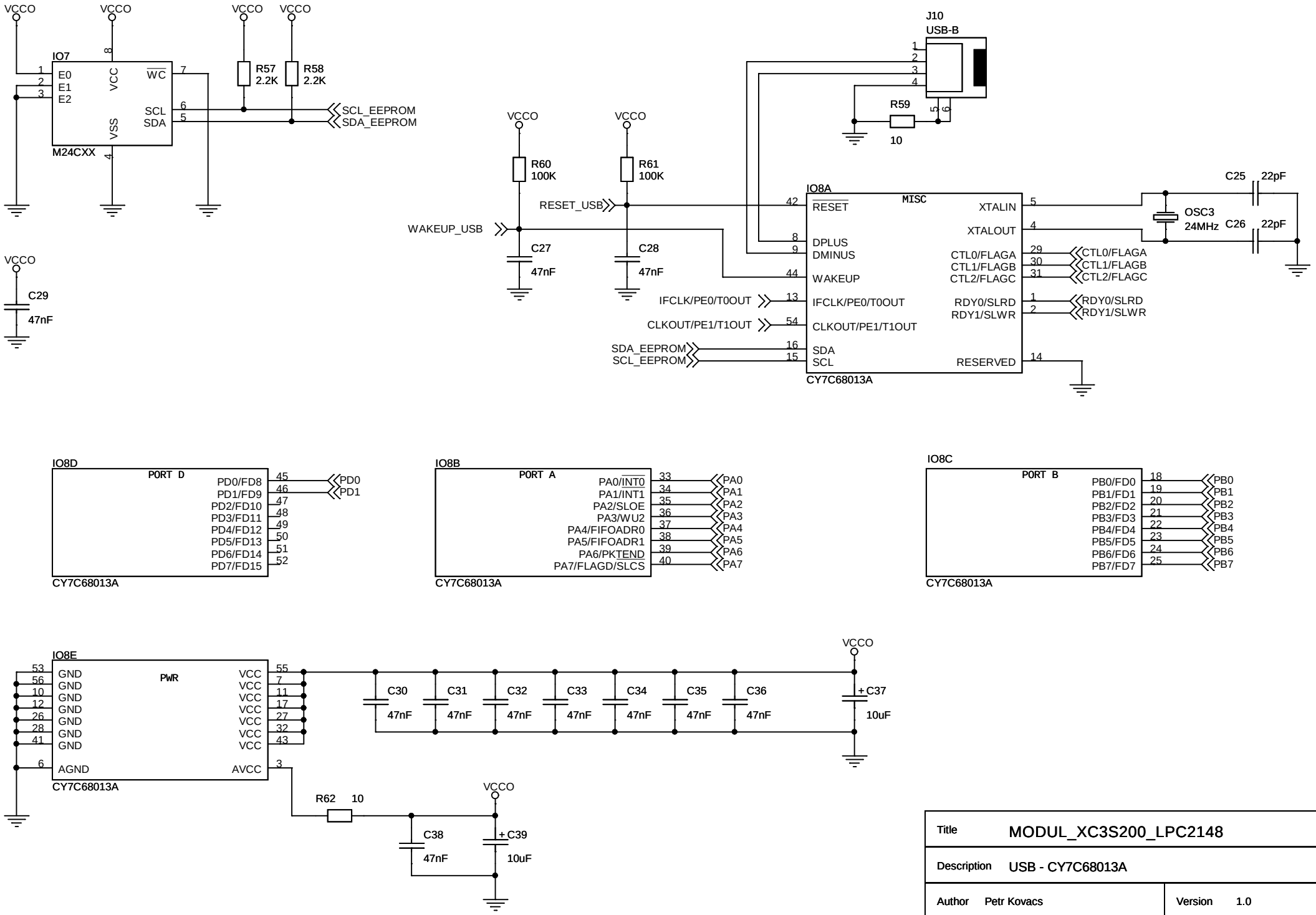




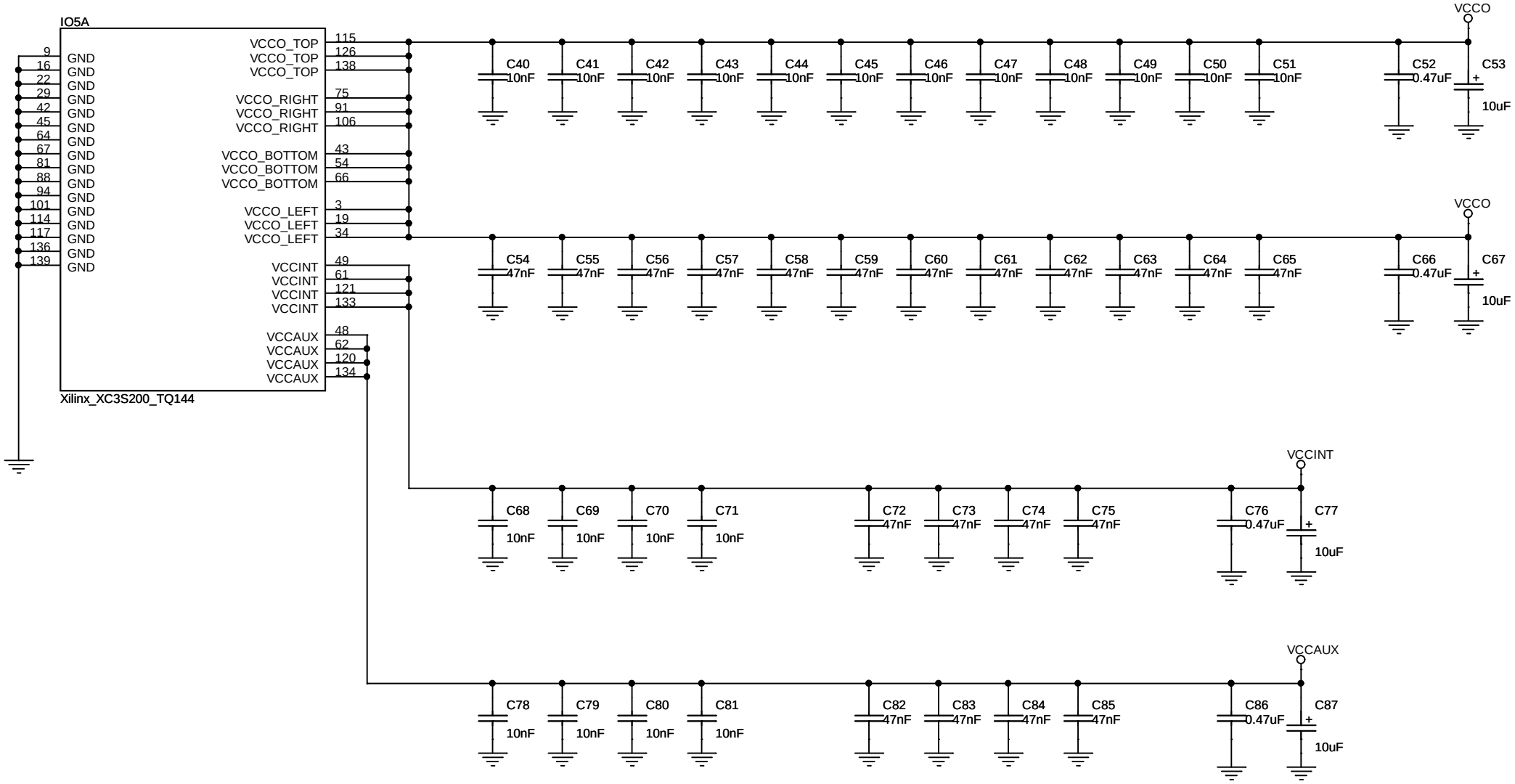
Title	MODUL_XC3S200_LPC2148
Description	FPGA XC3S200, FPGA konektor
Author: Petr Kovacs	Version: 1.0
Date: Friday, March 21, 2008	Sheet 5 of 8



Title	MODUL_XC3S200_LPC2148
Description	Konfigurace FPGA
Author	Petr Kovacs
Version	1.0
Date:	Friday, March 21, 2008
Sheet	6 of 8



Title	MODUL_XC3S200_LPC2148
Description	USB - CY7C68013A
Author: Petr Kovacs	Version: 1.0
Date: Friday, March 21, 2008	Sheet 7 of 8



Title	MODUL_XC3S200_LPC2148
Description	Blokovaci kondezatory pro FPGA
Author: Petr Kovacs	Version: 1.0
Date: Friday, March 21, 2008	Sheet 8 of 8