VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VÍCEKAMEROVÝ SNÍMAČ BIOMETRICKÝCH VLASTNOSTÍ LIDSKÉHO PRSTU

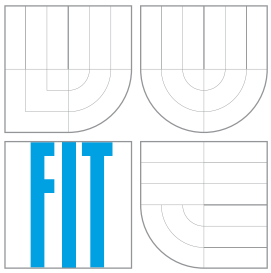
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ADAM TRHOŇ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VÍCEKAMEROVÝ SNÍMAČ BIOMETRICKÝCH VLASTNOSTÍ LIDSKÉHO PRSTU

MULTI-CAMERA SCANNER

OF BIOMETRIC FEATURES OF HUMAN FINGER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ADAM TRHOŇ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JOSEF STRNADEL, Ph.D.

BRNO 2015

Abstrakt

Tato práce se zabývá návrhem konceptu platformy pro bezdotykové snímání otisku prstu a jeho realizací. Dále je popsán návrh, implementace a testování hardware implementovaného ve VHDL a programu implementovaného v C. Výstup práce lze použít jako základ při vývoji průmyslově použitelného řešení.

Abstract

This thesis describes a conceptual design of touchless fingerprint sensor and design, implementation and testing of its firmware, which is a composition of hardware implemented in VHDL and a program implemented in C. Result of this thesis can be used as the first step of building an industrial solution.

Klíčová slova

biometrie, jazyk C, VHDL, Altera, hardware-software codesign, vícekamerový snímač otisku prstu

Keywords

biometrics, C programming language, VHDL, Altera, hardware-software codesign, multi-camera fingerprint scanner

Citace

Adam Trhoň: Vicekamerový snímač biometrických vlastností lidského prstu, diplomová práce, Brno, FIT VUT v Brně, 2015

Vícekamerový snímač biometrických vlastností lidského prstu

Prohlášení

Prohlašuji, že jsem tento diplomový projekt vypracoval samostatně pod vedením pana Ing. Josefa Strnadela Ph.D.

.....
Adam Trhoň
27. května 2015

Poděkování

Děkuji Ing. Josefu Strnadelovi, Ph.D. za vedení této práce, Ing. Tomáši Novotnému za cenné rady a firmě TBS CZ za zapůjčení ukázkového řešení.

Obsah

Kapitola 1

Úvod

Otisk prstu je v současné době nejrozšířenější biometrikou [8]. Jeho použití lze rozdělit do tří fází: sejmutí otisku, extrakce šablon a jejich porovnání (v případě verifikace) či vyhledání (v případě identifikace). Sejmutí otisku lze provést různými způsoby, které jsou podrobně rozebrány v [8]. Hrubě by se metody daly rozdělit na kontaktní a bezkontaktní, bezkontaktní pak podle použité technologie na optické a ultrazvukové.

Do kontaktních technologií patří např. optické, které otisk snímají kamerou, tlakové, které využívají piezoelektrického jevu, kapacitní, které měří kapacitu mezi vodivými ploškami a pokožkou, a další. Jejich společným rysem je právě kontakt prstu se senzorem, což může být pro některé uživatele nebo některá prostředí (např. nemocnice) problém z hlediska hygieny. Dalším rysem je typicky snímání píchaného otisku.

Bezkontaktní ultrazvukové technologie vysílají pulzy vysokofrekvenčního (v řádech MHz) zvukového vlnění, které snímají řadou přijímačů rozmístěných v rovině kolmé k vysílaným pulzům. Technologie snímá vnitřní strukturu pokožky, je tak velmi robustní k zašpinění, poraněním a falešným otiskům nalepeným na prst. Nevýhodou technologie je typicky vysoká cena a rozměry.

Bezkontaktní optické technologie otisk snímají kamerou. Přístup je hygienický, nedochází k deformacím bříška prstu přítlakem na plochu senzoru. Při použití více kamer je možné sejmout válený otisk. Nedochází také k falešným přijetím způsobených latentními otisky. Technologie však ve srovnání s dotykovými senzory vyžaduje poměrně hodně prostoru a nutnost napozicovat prst (při vymezení polohy tělem senzoru se ztrácí bezkontaktnost). Problémy také mohou nastat s detekcí živosti. Například nabarvený sádrový odlitek nebude úspěšný na dotykových senzorech, bezdotykové ho však, pokud nedetekují živost, přijmou.

Zástupcem bezkontaktní optické technologie je senzor Surround Imager firmy TBS. Senzor je tvořen uzavřeným prostorem s otvorem, skrz který uživatel vloží prst. Prst je osvětlen několika desítkami LED a jeho snímky sejmuty třemi kamerami s rozestupem 45°. Podrobnější popis senzoru a jeho technických vlastností je uveden v kapitole 2.

Cílem této práce je návrh a realizace prototypu senzoru vycházejícího ze senzoru Surround Imager. Prototyp by měl minimalizovat nevýhody stávající verze při zachování její funkcionality. V první řadě musí být senzor menší. Ovlivňujícím faktorem, na který se práce zaměřuje, je složitost a velikost desky plošných spojů. Kontrolu živosti prstu lze provést například spektrální analýzou, která vyžaduje další snímky, což vede k dalšímu zhoršení problému s rychlostí sejmutí. Zvýšení rychlosti a snížení latence je tedy dalším cílem. Aby byla navržená platforma použitelná v praxi, bude zohledněna i cena řešení. Podrobnější požadavky na nový senzor, možnosti jeho realizace a výběr finálního návrhu jsou uvedeny v kapitole 3.

Finální návrh je založen na kombinaci FPGA a mikroprocesoru, zařízení Cyclone V firmy Altera. Na rozdíl od stávajícího návrhu, kde vše kromě komunikace po USB řídí FPGA, v novém návrhu FPGA zajišťuje pouze nezbytnou část řízení v reálném čase a všechny další úlohy jsou implementovány na mikroprocesoru. Pro řízení periferních zařízení je použit OS GNU/Linux. Pro propojení OS a FPGA je použit framework RSoC [11]. Rozbor použitých technologií a vývojových prostředků je uveden v kapitole 4. Firmware pro FPGA je rozebrán v kapitole 5 a software pro mikroprocesor v kapitole 6. Kapitola 7 popisuje testování prototypu, rozbor jeho výkonu a možnosti pro další vývoj.

Kapitola 2

Současný stav

Surround Imager je bezkontaktní, vícekamerový biometrický senzor pro snímání otisku prstu a jeho optických vlastností (například pro účely detekce živosti). Jak bylo uvedeno v úvodu, prst snímá třemi kamerami v rozestupu 45°. Snímky z kamer ukládá do interní paměti zařízení, ze které jsou čteny po sběrnici USB *uživatelskou aplikací*, která je spuštěna na *uživatelském zařízení*. Pro osvětlení je použito několika desítek LED. Senzor neobsahuje žádné biometrické algoritmy, pouze režii nutnou k vytváření snímků a jejich přenosu do uživatelské aplikace. Senzor je typicky vestavěn do přístupového terminálu, který je na obrázku 2.1. Tato kapitola se věnuje podrobnému popisu senzoru a jeho nedostatků.

2.1 Činnost senzoru

Proces snímání je vždy řízen uživatelskou aplikací. Ta na základě obrazu kamer v nízkém rozlišení periodicky kontroluje, zda se v zařízení nachází prst. Pokud ano, uživatelská aplikace zodpovídá za správné napozicování prstu (uživateli zobrazí živý náhled a instrukce pro případnou změnu pozice) a vydání příkazu pro spuštění snímání ve vysokém rozlišení. Po dokončení snímání jsou snímky uloženy v interní paměti senzoru, odkud je uživatelská aplikace přečte a zpracuje svými biometrickými algoritmy.

Surround Imager má dva režimy snímání, *preview* a *capture*. Režim preview v nekonečné smyčce ukládá snímky z kamer do trojitého bufferu — zatímco z jednoho bufferu je možné číst, do zbývajících dvou se střídavě zapisuje. Pro uživatelskou aplikaci je buffer transparentní. Režim je typicky použit pro snímkování v nízkém rozlišení za účelem detekce prstu a jeho pozicování.

Nedostatkem režimu preview je, že zatímco je čten obraz z jedné kamery, senzor může aktualizovat obrazy zbývajících dvou. Proto není možné získat trojici snímků vytvořenou v jeden okamžik. To zajišťuje režim capture, který na příkaz z uživatelské aplikace vytvoří až 16 trojic snímků a uloží je na vyhrazené adresy v paměti senzoru, kde jsou uchovány do příštího spuštění režimu capture. Senzor poté automaticky přejde zpět do režimu preview.

Na oba režimy se lze dívat obecně, jako na iterační procesy. Preview provádí nekonečný počet iterací a snímky z nich ukládá do trojitého bufferu. Capture provádí maximálně 16 iterací a snímky ukládá na vyhrazené adresy. V dalším textu budou tyto iterace označovány jako *fáze*.

Obraz nasnímaný kamerami je možné před uložením do RAM zmenšit (2x, 4x nebo 8x), případně zpracovat libovolnou bodovou funkcí (ta je v senzoru realizována pomocí konfigurovatelné look-up tabulky — LUT). Při čtení obrazu z RAM je navíc možné nastavit

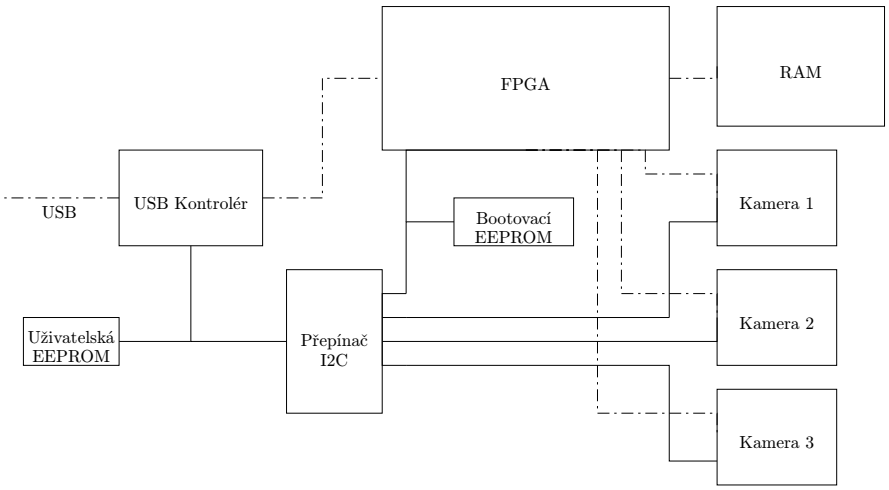
Obrázek 2.1: 3D terminál firmy TBS se senzorem Surround Imager.

ROI (region of interest) a přečíst tak jen část obrazu. Tato funkce však nefunguje správně, pokud byl obraz před uložením do RAM zmenšen.

Nejdůležitějšími parametry, které senzor umožňuje nastavovat, jsou registry kamer, časování snímání, časování osvětlení a intenzita osvětlení. Nastavení registrů kamer je globální pro preview i capture, zbylá nastavení jsou specifická pro každou fázi.

2.2 Popis Hardware

Funkcionalita senzoru je rozdělena mezi několik hlavních komponent. Komunikaci přes USB zajišťuje kontrolér Cypress EZ-USB FX2LP s vestavěným mikrokontrolérem 8051, jehož firmware je třeba do senzoru nahrát při každém startu. Poskytuje USB transceiver, I^2C master pro komunikaci s kamerami a EEPROM a paralelní rozhraní, přes které komunikuje s FPGA.



Obrázek 2.2: Hlavní komponenty senzoru. Plnou čarou je vyznačeno I^2C, čerchovanou čarou ostatní datové a řídící sběrnice. Signály LED (výstupy PWM generátorů v FPGA) zobrazeny nejsou.

Jako FPGA je na desce Cyclone III firmy Altera. Varianta EP3C16F484 poskytuje 15 k logických elementů a 346 I/O pinů. Čip je paralelní sběrnicí spojen s USB kontrolérem, sběrnicemi I^2C s kamerami a EEPROM. FPGA je navíc zodpovědné za řízení RAM. Firmware pro FPGA je uložen ve vyhrazené EEPROM. Upgrade firmware je možné provést buď přes rozhraní JTAG nebo přes bootloader uložený v EEPROM. Problémem je případná aktualizace samotného bootloaderu. Kritické selhání (např. výpadek proudu) může při aktualizaci v poli vést až ke znehodnocení zařízení.

Kamerami zařízení jsou MT9M001 firmy Aptina. Jedná se černobílé CMOS kamery s rozlišením až 1280 × 1024, hloubkou obrazu 10 b a maximálním framerate 30 fps. Kamera je konfigurovatelná přes sběrnici I^2C a obrazová data posílá paralelní sběrnicí. Datové signály jsou synchronizovány hodinovým signálem PIXCLK, který generuje kamera. Odesílání řádku nebo celého snímku kamera signalizuje signálem LINE_VALID, resp. FRAME_VALID. Data jsou generována na náběžné hraně PIXCLK, jejich čtení se předpokládá na hraně sestupné. Pro přesné časování snímání musí být všechny kamery při inicializaci zařízení přepnuty do módu *triggered*, ve kterém sejmou snímek pouze po detekci náběžné hrany signálu TRIGGER. Podrobné informace o kameře lze nalézt jejím datasheetu [6] a popisu registrů [7].

Protože I^2C adresy kamer jsou fixní, je potřeba prostředek pro přepínání I^2C sběrnice. Tím je PCA9548, 8kanálový přepínač I^2C. Jedná se o aktivní prvek se svou vlastní adresou, který sběrnici větví na větev master a až 8 větví slave. Ke větvi master připojí vždy jednou větev slave, který je uložen v registru přepínače. Rozhraní pro zápis registru je vystaveno na větvi master. Před komunikací s kamerou je tedy vždy potřeba vybrat příslušnou větev.

Na jedné z ze slave sběrnicí je i EEPROM s firmware pro FPGA, teoreticky by tedy bylo možné firmware pro FPGA nahrávat přímo z USB kontroléru přes I^2C, nikoliv přes FPGA. To by umožnilo bezpečnou aktualizaci jak firmware, tak bootloaderu. Adresa EEPROM pro uložení firmware je však stejná jako adresa EEPROM uživatelských dat a nastavení USB kontroléru. Tato EEPROM je na sběrnici vždy (připojena k master větvi), čímž je přímé nahrávání znemožněno.

O osvětlení prstu se stará 24 skupin LED. Každá skupina je přes zesilovač řízena z FPGA. Skupiny jsou rozděleny na přední zelené (12 skupin po 5 LED), zadní zelené (9 × 4), modré, červené a infračervené (1 × 2).

Propojení hlavních komponent je zobrazeno na obrázku 2.2.

2.3 Model komunikace po USB

S uživatelským zařízením senzor komunikuje po sběrnici USB. Nevyužívá žádné existující třídy, ale protokol si definuje sám.

Pro veškerou konfiguraci, čtení stavových informací a komunikaci s komponentami na desce jsou použity uživatelské požadavky zasílané přes endpoint 0. Pole `bRequest` identifikuje subsystém, pro který je požadavek určen (FPGA, I2C, nebo USB řadič). Pro FPGA se jedná o aktualizaci a přístupy do adresového prostoru, kde adresa je určena hodnotou pole `wIndex`. Pole `wValue` definuje směr přenosu. Aktualizace je identifikována hodnotou `bRequest` a řízena hodnotami `wValue`, `wIndex` použito není.

Jediná funkce, která nevyužívá endpoint 0, je čtení snímků. Ty jsou přenášeny endpointem typu bulk. Nejdříve je odeslán požadavek přes endpoint 0 pro inicializaci čtení v řadiči USB. Následně je přečteno potřebné množství dat. Před čtením snímku je potřeba snímek vybrat — zařízení v paměti obsahuje snímky z režimů preview i capture. Výběr je proveden zápisem do registru FPGA.

2.4 Nedostatky senzoru

Při přepínání režimů je typicky potřeba i rekonfigurace některých globálních nastavení. Například při přechodu z preview do capture uživatelská aplikace před spuštěním capture vypíná zmenšení obrazu. Protože je při tom spuštěn režim preview, sejme několik snímků ve vysokém rozlišení. Neatomické přepínání někdy vede až k souběhu.

Tento problém by se dal vyřešit dočasným zastavením snímání preview, takovou funkci však senzor neposkytuje.

Extrémně vzácnou, nicméně pro přístupový terminál stejně tak vážnou, chybou je tzv. `caching effect`. Z neznámých příčin dojde k tomu, že po dokončení snímání jsou ze senzoru přečteny snímky ze snímání předchozího. K chybě dochází pouze na některých zařízeních, pouze po některých restartech, a je možné ji detekovat v aplikaci (snímky mají stejný CRC). Ačkoliv nikdy není možné se stoprocentně vyhnout chybám, absence jakéhokoliv rozhraní pro diagnostiku senzoru jejich řešení zbytečně komplikuje.

Dalším problémem je již zmíněná nemožnost výběru ROI a současně jeho zmenšení. Tato kombinace by byla užitečná pro některé optimalizace procesu snímání. Problém by se dal vyřešit zadáním těchto parametrů do kamery, zde je ale problém se stanovením velikosti výsledného obrazu — experimentálně bylo zjištěno, že pro různé kombinace ROI a zmenšení kamery vrací různě veliké snímky. V datasheetech kamery toto chování popsané není.

I když senzor umožňuje při více fázích capture číst již dokončené fáze, zatímco se ostatní fáze stále snímají, signalizuje pouze dokončení celého capture. Klientská aplikace tedy musí odhadnout, kdy je první fáze dokončena, což nemusí být za každých okolností spolehlivé. Proto této formy zřetězení není využito.

Kapitola 3

Návrh prototypu

3.1 Specifikace požadavků

Nový senzor musí obsahovat všechny funkce stávajícího senzoru a zmírnit problémy vyjmenované v kapitole 2. Kromě toho by řešení postavené na prototypu mohlo být menší a levnější, čehož by šlo dosáhnout zjednodušením a zmenšením desky. Následuje podrobná definice požadavků:

1. Režim preview pro živý náhled kamer. Režim obsluhuje násobný buffer a musí zajistit, že uživatelská aplikace vždy z uložených snímků dostane nejnovější možný a zároveň nikdy nedojde k rozdílnému pořadí zápisu a čtení.
2. Režim capture pro snímání ve vysokém rozlišení. Režim musí být schopný uložit alespoň 16 fází.
3. Režim idle, ve kterém neběží ani capture, ani preview.
4. Intenzita osvětlení. Pro každou fázi a pro každou skupinu LED musí být definovatelná intenzita, se kterou bude svítit.
5. Časování osvětlení. Pro každou fázi a pro každou skupinu LED musí být kromě definovatelný počátek a konec časového úseku, po který bude během fáze svítit.
6. Časování snímání. V každé fázi musí být pro každou kameru definovatelné zpoždění mezi začátkem fáze a začátkem snímání. Systém musí být schopný ukládat data ze všech kamer zároveň.
7. Dynamické parametry kamer. Řídicí jednotka musí být schopná nastavovat parametry kamer pro každou fázi zvlášť.
8. Kombinace ROI a zmenšení obrazu. Obě funkce musí být k dispozici v řídicí jednotce senzoru, pro zvýšení rychlosti čtení dat z kamer musí jednotka zvládnout případ, kdy jedna z funkcí je nastavena v kameře. Nastavení musí být definovatelné pro každou fázi zvlášť.
9. LUT pro převod z desetibitové do osmibitové hloubky obrazu. Nastavení musí být definovatelné pro každou fázi zvlášť.
10. Komunikace s uživatelskou aplikací přes USB.

11. Detekce prstu.

12. Signalizace dokončení jednotlivých fází capture.

13. Možnost bezpečné aktualizace firmware. Bezpečností je zde myšleno, aby za žádných okolností (např. ani při výpadku proudu) nedošlo přerušením aktualizace k zablokování senzoru.

14. Diagnostické rozhraní. K dispozici musí být signalizace stavu a signalizace detekovatelných chyb (například přetečení bufferu, chyba komunikace s kamerou, atd.), například bitovou maskou. Dále musí senzor poskytovat rozhraní pro reset do definovaného stavu.

15. Použití stávajících kamer.

Detekce prstu bude jediným algoritmem pro zpracování snímků umístěným na senzoru. Při stávající detekci se obraz přenáší do uživatelské aplikace, která velmi jednoduchým algoritmem určí, zda se prst v senzoru nachází, či ne. Kontrola se provádí zhruba dvakrát za vteřinu. Experimenty bylo zjištěno, že zvýšením frekvence kontrol je objektivní rozdíl v reakční době sice malý (snížení průměrné doby reakce z 200 ms na 50 ms při 10 kontrolách za vteřinu), subjektivně však senzor reaguje mnohem rychleji. Cenou za zrychlení je vysoká zátěž procesoru v terminálu způsobená přenosem po USB, což je mimo interakci s uživatelem nepřípustné. Pokud by detekci prováděl přímo senzor, režie USB nebude mít vliv.

3.2 Rozbor návrhu hardware

Při návrhu hardware byly nejdříve identifikovány hlavní otázky:

- Jaká bude použita technologie? FPGA, MCU nebo kombinace? Které komponenty zajistí generování PWM pro osvětlení a příjem dat z kamer?
- Musí návrh obsahovat paměť RAM? Nemohla by se obrazová data odesílat přímo do uživatelského zařízení?
- Jakým způsobem se bude komunikovat s uživatelským zařízením?
- Jakým způsobem bude možné senzor ladit?

Tyto otázky jsou rozvedeny a zodpovězeny v následujících kapitolách, po kterých následuje shrnutí a finální návrh platformy.

3.2.1 Použitá technologie

Dalším krokem bylo rozdělení úloh mezi mikroprocesor a FPGA. Veškerou funkcionalitu by bylo možné implementovat přímo ve VHDL, tak jako ve stávající verzi. To je vhodné pro implementaci PWM generátorů pro LED a čtení data kamer, na druhou stranu vzniknou problémy se správou nastavení a ovládáním periferních zařízení (např. konfigurace kamer přes I^2C). Komunikaci s hostitelským zařízením by v případě složitého protokolu (např. USB) mohla zajistit externí komponenta, ta by však znamenala zvětšení desky.

Tyto problémy by řešilo použití mikroprocesoru, na druhou stranu v něm bude nemožné realizovat dostatek kanálů PWM generátoru a čtení kamer a musely by se opět použít externí komponenty.

Třetí možností je kombinace dvou předchozích, tedy implementace přesného časování jednotkou ve VHDL a řízení této jednotky procesorem. Kromě splnění všech požadavků má řešení další výhody:

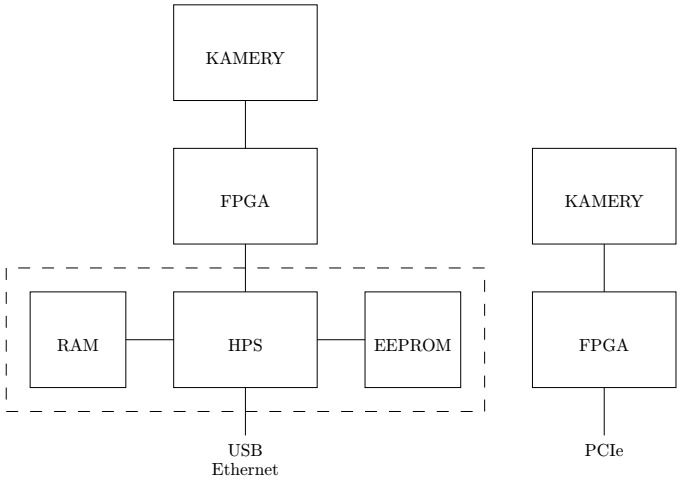
- Zjednodušení kódu FPGA firmware. Ručně psaná real-time jednotka bude jednodušší, protože v FPGA nebude potřeba řešit uchovávání konfigurací, komunikaci s periferiemi ani s uživatelským systémem. To povede k redukci množství chyb.
- Oddělení logiky snímání (akcelerátor v FPGA) od real-time řízení (program na MCU). Díky oddělení nebudou změny firmwaru FPGA ovlivňovat MCU a naopak.
- Větší možnosti nastavení. Například díky napojení procesoru na sběrnice I^2C může MCU před každou fází capture rekonfigurovat kamery.
- Možnost implementace pomocných procedur, například self-testu, v rámci firmware MCU. Ve stávající verzi musí být tyto procedury implementovány v software uživatelské aplikace.

Požadovaná funkcionalita bude rozdělena mezi dvě komponenty — akcelerátor snímání implementovaný ve VHDL a řízení senzoru implementované v C. Pro takové uspořádání je ideální architektura SoC, ve které je jak MCU, tak FPGA na jednom čipu. Typicky jsou k dispozici i řadiče pro periferie, pro tuto práci jsou důležité hlavně I^2C, USB a SDRAM.

3.2.2 Ukládání snímků do paměti zařízení

Senzor obsahuje tři kamery s rozlišením až 1280×1024 pixelů, ty v jedné fázi dohromady vyprodukují přes 3.6 MB dat. Taková kapacita není k dispozici v žádném z běžně dostupných FPGA, tedy pokud snímky nebudou odeslány do hostitelského zařízení v reálném čase, musí být použita externí paměť. Kamery jsou synchronizovány hodinami s frekvencí 50 MHz a data odesílají po paralelní sběrnici. Maximální datový tok je tedy 150 MB/s pro všechny tři kamery. Tuto rychlost nemůže garantovat žádná z běžně používaných sběrnic pro připojení externích periferií k počítači. Vyhovovat by mohla USB 3, senzor však musí být schopný komunikovat i se staršími PC, které USB 3 nedisponují. Zpětná kompatibilita mezi USB 2 a 3 zde nepomůže, protože kritická je rychlost. Ze stejného důvodu (potřeba širokého rozšíření) odpadají i další sběrnice, jako firewire. Požadavek na ukládání snímků v paměti senzoru tedy zůstává.

Jinou možností by bylo použití vysokorychlostní sběrnice mezi senzor a hostitelské zařízení v terminálu a v enroll stanici použít mezičlánek. Například PCIe v jednolinkové variantě poskytuje dostatečně vysokou přenosovou rychlost pro přenos mezi senzorem a terminálem, nebylo by však možné jednoduše připojit enroll stanici k PC. Proto by enroll stanice obsahovala mezičlánek — malou desku s MCU, který by snímky ukládal do RAM a s PC komunikoval přes USB. V této variantě odpadá i potřeba MCU v senzoru, protože funkce řízení senzoru může díky rychlé sběrnici provádět MCU terminálu či mezičlánku. Výhodou by byla i rychlejší komunikace mezi senzorem a terminálem. Ta však není kritickým místem. Navíc by bylo potřeba navrhnout další desku pro mezičlánek enroll stanice. Dokud tedy nevznikne požadavek na další zmenšení desky senzoru v terminálu, přidaná hodnota takového řešení není příliš velká. Srovnání obou možností je na obrázku 3.1.



Obrázek 3.1: Blokové schéma architektury připojené pomalou (nalevo) a rychlou sběrnicí. Připojení rychlou sběrnicí nevyžaduje komponenty v čárkovaně vyznačeném segmentu, protože v případě terminálu jsou zastoupeny komponentami terminálu. V případě enroll stanice však pomalá sběrnice nutná bude a komponenty musí být na speciální desce v enroll stanici.

3.2.3 Komunikace s uživatelským zařízením

Ve stávajícím senzoru je pro komunikaci s uživatelským zařízením použita sběrnice USB pro svoji rozšířenost a rychlost. Na rozdíl od jiných sběrnic je však USB poměrně složité, a pro specializovaná zařízení, která nespadají pod žádnou ze tříd definovaných USB, je potřeba napsat vlastní driver jak na straně uživatelského zařízení, tak na straně senzoru. Pro hostitelská zařízení je situace usnadněna knihovnou libUSB, která umožňuje použití sběrnice čistě z uživatelského prostoru. Na straně senzoru existuje podpora pro USB v systémech OpenRTOS nebo Linux. V Linuxu se jedná o jaderný subsystém USB Gadget, nicméně existuje i nadstavba, která opět umožňuje vývoj služby uživatelského prostoru — GadgetFS. Ten pro každý endpoint vytvoří uzel souborového systému, přes který je možné komunikovat s hostitelem.

Spíše experimentální možností by mohlo být vytvoření kompozitního USB zařízení, které by pro konfiguraci a zjišťování stavu obsahovalo sériovou linku, pro přenos videa režimu preview kameru a pro přenos snímků z režimu preview velkokapacitní úložiště. Pro všechna tato dílčí zařízení jsou k dispozici standardní ovladače. Nevýhodou by byla vyšší složitost na straně senzoru i uživatelské aplikace.

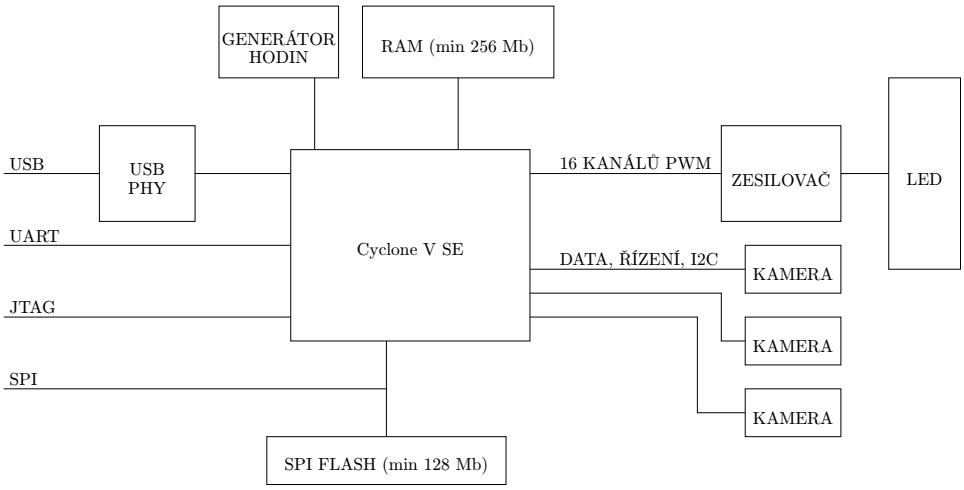
Pokud senzor podporuje ukládání snímků do interní paměti, je možné uvažovat i jiné sběrnice. Jediným dostatečně rozšířeným kandidátem je Ethernet. Senzor by fungoval jako TCP server. Pro režim preview by bylo vhodnější použít UDP, nicméně TCP je vyžadováno pro konfiguraci a režim capture. Zavedení druhého kanálu přes UDP by zkomplikovalo architekturu software, a datový tok režimu preview není natolik veliký, aby přenos přes TCP způsobil problémy. Jako TCP server by senzor mohl podporovat střídavě více klientů, i bezdrátových, což je výhodné zejména pro enroll stanice. Velkou výhodou je existence standardních driverů jak na straně senzoru, tak na straně klientů. Na druhou stranu pak bude potřeba zabezpečení komunikace mezi senzorem a klientem, a jejich autentizace (neznámý klient může otisk ukrást, v neznámém senzoru může být zase snímán útočník). Pro návrh senzoru by Ethernet znamenal více hardware (fyzická vrstva protokolu, konektor) a software.

3.2.4 Ladění senzoru

Pro ladění a počáteční programování senzoru bude na desce vyveden konektor JTAG tak, jako ve stávající verzi. Přes JTAG je možné ladit jak firmware procesoru, tak FPGA. Tento způsob je však poněkud těžkopádný, pokud je potřeba zařízení sledovat dlouhodobě, nebo je sledovaných zařízení více. Z tohoto důvodu bude senzor konektor pro připojení sběrnice UART. Ta lze jednoduše převést na USB a připojit k PC nebo jinému sledovacímu zařízení. Jiné způsoby monitorování, jako logování do souboru, kvůli snaze o minimalizaci a zjednodušení hardware vhodné nebudou.

3.2.5 Finální návrh

Finální návrh jednoznačně vychází z odpovědí na hlavní otázky. Srdcem senzoru bude SoC, který obsahuje FPGA, MCU a hardwarové řadiče periferií a sběrnic. S uživatelským zařízením bude senzor komunikovat po sběrnici USB či Ethernet. Příjem dat z kamer, generátory PWM a časování těchto funkcí bude zajišťovat FPGA, ostatní funkcionalita (uchování konfigurace, komunikace s kamerami po I^2C, komunikace s uživatelským zařízením) bude implementována v MCU. Propojení mezi mikrokontrolérem a FPGA zajistí framework RSoC. Data z kamer senzor uloží v interní paměti a odešle na vyžádání uživatelského zařízení. Senzor bude možné ladit přes sběrnice UART a JTAG. Podrobný návrh výkonových částí (napájení senzoru, LED) a realizace hardware jsou mimo rámec práce, proto bude koncept otestován na vhodném vývojovém kitu. Blokové schéma konceptu hardware je na obrázku 3.2.



Obrázek 3.2: Blokové schéma hardware nového konceptu senzoru a jeho rozhraní. USB (analogicky Ethernet) pro komunikaci s hostitelským zařízením, UART a JTAG pro ladění, SPI pro snadnější nahrávání firmware a kontrolu jeho integrity.

3.3 Rozbor řízení senzoru

Řízení senzoru řídí akcelerátor snímání a ostatní hardware na desce. Musí poskytovat oba režimy práce senzoru (preview i capture), klidový režim a být schopno mezi nimi atomicky přepínat. Musí také poskytovat několik konfigurací pro režim preview. Zajišťuje i komunikaci s uživatelským zařízením. Lze ho rozdělit do tří částí. První částí jsou ovladače periferií.

Druhou je ovladač akcelerátoru snímání, který do ovladačů periferií není zahrnut, protože se nejedná o zařízení, pro které by existoval ovladač. Třetí částí je vlastní služba řízení, která pracuje nad již abstrahovaným hardwarem.

3.3.1 Ovladače periferií

Firmware je možné napsat bez operačního systému, znamenalo by to však zahrnout do něj implementaci vlastních ovladačů periferií. To však nepřinese žádné výhody oproti použití operačního systému RTOS. Jedná se o lehkotonážní real-time operační systém, který nepřináší ani výrazný nárůst velikosti firmware (takový, který by vedl k potřebě větší kapacity než nabízejí EEPROM), ani degradaci výkonu (takovou, která by ovlivnila cílovou aplikaci). Nevýhodou systému RTOS pro tento projekt je absence ovladačů USB, které jsou zahrnuty pouze v komerčních edicích. V systému USB by se tedy projevily nevýhody ručně psaného firmware, hlavně chybovost.

Druhou možností je použití GNU/Linuxu. Jednalo by se o velmi robustní řešení, které obsahuje všechny potřebné komponenty [13]. Nevýhodou je velikost systému, která se však dá zredukovat pod 8 MB, a jistý overhead při běhu, který však také nevadí, protože procesy náročné na časování jsou řešeny v FPGA. Výkonové problémy by mohly nastat při přenosu dat z FPGA do paměti RAM, tuto charakteristiku tedy bude nutné ověřit v implementovaném prototypu.

3.3.2 Ovladač akcelerátoru snímání

Bez požadavků na rychlost je komunikace mezi procesorem a akcelerátorem s použitím systému QSys relativně jednoduchá. Pro vysokorychlostní přenos obrazových dat z akcelerátoru do RAM je však potřeba DMA řadič na straně FPGA a ovladač na straně procesoru. Jednodušší možností (bez implementace DMA) by byl zápis z akcelerátoru do paměti přímo přes řadič RAM. Na trhu je pro tuto úlohu dostupný framework RSoC, který poskytuje komponentu pro FPGA a driver pro GNU/Linux a umožňuje jednoduchou výměnu dat mezi oběma systémy [11].

3.3.3 Služba řízení

Úlohu služby řízení lze rozdělit na dvě části. První je ukládání nastavení přijatých od uživatelského zařízení a jejich distribuce do ovladačů periferií ve správný čas. To zajistí možnost nastavení více fází. Druhou částí je ukládání snímků vytvořených akcelerátorem snímání a jejich distribuce do uživatelského zařízení. Mimo jiné zde bude implementován trojitý buffer pro režim preview.

Nastavení lze rozdělit do několika vrstev. Na nejnižší vrstvě je přímá komunikace s hardwarem. Pokud uživatelský program odešle instrukci pro zápis určitého registru, instrukce projde službou řízení, nicméně bude ihned provedena. To je vhodné například pro experimentování se zařízením či ladění chyb. Nevýhodou je nízká rychlost, kde každý požadavek pro hardware senzoru musí mít svůj požadavek přenesený po USB.

Služba řízení může provádět i některé operace nad obrazem, například detekci prstu. Vždy bude potřeba pečlivě zvážit, zda zisk při implementaci v senzoru převáží problémy způsobené obtížnější aktualizací algoritmu (oproti aktualizaci v uživatelském zařízení). V případě detekce prstu je ziskem snížení dobu odezvy senzoru při vložení prstu a zlepšení subjektivního vnímání celého procesu uživatelem. Výhodný je i fakt, že v případě problémů lze tento úkon provádět i v uživatelském zařízení.

3.4 Akcelerátor snímání

Úlohou akcelerátoru snímání je provedení jedné fáze jako reakce na signál z řízení senzoru. V rámci provádění musí po daném zpoždění spustit snímání kamer, spouštět PWM generátory pro řízení LED, přenést snímky z kamer do RAM, ukončit fázi a vyčkat na další signál. Za opakování fáze preview nebo přechod na další fázi capture zodpovídá řízení senzoru. Činnost jednotky je oproti stávající verzi jednodušší, což je jedním z cílů tohoto návrhu.

Pro každou z těchto částí musí akcelerátor obsahovat komponentu. Pro jednotlivé komponenty vyplývají tyto požadavky:

- Generátory PWM mohou být relativně jednoduché, protože LED diody nepatří mezi zařízení citlivé na vstupní signál.
- Akcelerátor bude synchronizován hodinami MCU, komponenta pro čtení dat z kamery tedy bude synchronizovat data z hodinové domény kamery do hodinové domény akcelerátoru.
- Před odesláním dat do RAM bude snímek možné zmenšit nebo ořezat. Velikost snímků nebude předem známa a je potřeba ji odvodit ze signalizace kamer.
- Kvůli zdrojům na vybraném FPGA a ceně frameworku bude použit pouze jeden kanál pro přenos z FPGA do RAM. Data proto budou před odesláním rozdělena do paketů.
- Při přenosu dat z FPGA do RAM musí akcelerátor signalizovat odeslání všech dat, například odesláním prázdného paketu.
- Každá komponenta bude poskytovat informace o svém stavu komponentě pro diagnostiku. Ta informace sjednotí do masky, která bude přes RSoC poskytnuta řídicí jednotce.

3.5 Zavádění systému a jeho aktualizace

Pro zavádění systému bude použit u-boot. Obraz souborového systému bude uložen jako další soubor vedle obrazu jádra. To neumožní permanentní modifikaci souborového systému, modifikace však nejsou potřeba. Na druhou stranu to výrazně zjednoduší jeho případnou aktualizaci.

Pro účely aktualizace bude u-boot podporovat dvě různá jména souborů obrazu jádra a obrazu souborového systému, mezi kterými se bude přepínat proměnnou prostředí. Sám u-boot do procesu aktualizace zasahovat nebude.

Součástí návrhu firmware platformy je tedy i návrh mechanizmů jeho aktualizace. Aktualizace se týkají firmware FPGA, obrazu jádra, obrazu souborového systému a služby řízení senzoru. Ačkoliv by služba řízení mohla být součástí obrazu souborového systému, na rozdíl od obrazu se její aktualizace v budoucnosti očekává i v případě potřeby rozšíření funkcionality. Ostatní softwarové komponenty budou aktualizovány co nejméně, pouze v případě potřeby opravy chyb. Proto bude součástí obrazu souborového systému pouze modul pro komunikaci s uživatelským zařízením a služba aktualizací.

Po spuštění systému GNU/Linux se spustí služba aktualizací. Pokud by se systém nacházel uprostřed přerušené aktualizace, služba se pokusí problém vyřešit. Dále zkontroluje,

že je k dispozici služba řízení a že je inicializované FPGA. Pokud je vše v pořádku, služba aktualizací se ukončí a je spuštěna služba řízení. V případě, že bude potřeba spustit aktualizaci, se služba řízení ukončí a spustí se služba aktualizace.

Díky oddělení služeb bude možné službu řízení a firmware FPGA aktualizovat bez rizika. Pokud bude některá z komponent nefunkční, služba aktualizace ji může přijmout znovu a opravit. I pokud dojde k výpadku proudu, zařízení bude zablokováno pouze v případě, že by došlo k poškození souborového systému EEPROM.

Pro aktualizaci obrazu jádra nebo obrazu souborového systému je potřeba složitější mechanizmus, protože komponenty aktualizují samy sebe. Při aktualizaci služba zapíše komponentu do nového souboru, vedle stávající verze. Zkontroluje CRC zapsaného souboru a pokud je v pořádku, modifikuje proměnné prostředí u-bootu tak, aby při příštím restartu použil novou verzi. Starou verzi je po restartu možné smazat.

3.5.1 Zabezpečení aktualizací

Ačkoliv útok provedený nahráním upravené verze firwmare vyžaduje detailní znalost systému útočníkem, na neznalost se nelze spolehnout. Útočník by teoreticky mohl z jiného zařízení přečíst EEPROM. Tím získá kompletní systém, který může upravit, aby např. vracel falešné snímky. S novým souborem může vzdáleně zaútočit na počítač s enroll stanicí a stanici infikovat.

Jednoduchý způsob, který tento typ útoku téměř znemožní, je veškeré aktualizace podepisovat a ve službě aktualizací podpis kontrolovat. Pro podpisy může být použit např. systém PGP. Stanici je stále možné infikovat přímým zápisem do EEPROM, to již ale nejde provést vzdáleně.

Kapitola 4

Použité prostředky

V kapitole 3 byl navržen koncept nového senzoru. V této kapitole jsou popsány technologie a prostředky pro jeho vývoj.

4.1 USB

USB je univerzální sériová sběrnice široce používaná v oblasti osobních počítačů. K dipozici jsou jak oficiální specifikace sběrnice (dostupné na `http://usb.org`), tak velmi kvalitní průvodce sběrnicí [10], ze kterého čerpá tato práce.

Ve verzi 2.0 se USB fyzicky skládá ze čtyř vodičů, dvou pro napájení a diferenciálního páru pro poloduplexní přenos dat. Sběrnice jako taková je point-to-point, pro její větvení lze použít rozbočovače. Vzniká stromová struktura, v jejímž kořeni se nachází hostitelský počítač, v listech zařízení a v ostatních uzlech rozbočovače. Veškerá komunikace na sběrnici je řízena hostitelským počítačem.

Na nejnižší úrovni jsou data zakódována kódem NRZI. Jednotlivé bity jsou organizovány do paketů, ze kterých jsou sestaveny transakce. Transakce typicky odpovídá provedení čtecího nebo zápisového (směr je vždy určen z pohledu hostitelského počítače) požadavku.

Každé zařízení má na sběrnici unikátní sedmibitovou adresu. Komunikace s hostitelským zařízením je možná více kanály, ve sběrnici USB nazvané roura (pipe). Roury jsou napojeny na koncové body zařízení (endpoint), které jsou adresovány dalšími čtyřmi bity. Ve srovnání s TCP/IP by adresa zařízení odpovídala IP adrese, adresa endpointu portu a roura vytvořenému spojení mezi klientem a serverem.

Endpointy (a na ně připojené roury) můžou mít různé typy. Každé zařízení musí podporovat endpoint 0, který je typu control. Jako jediný přenáší data strukturovaně a umožňuje poloduplexní komunikaci. Slouží pro inicializaci zařízení.

Dalšími typy jsou interrupt endpoint, bulk endpoint a isochronous endpoint. Všechny endpointy jsou simplexní. Ideálně by bylo možné splnit tři požadavky na přenos: rezervované pásmo, libovolně velké množství dat, oprava chyb na sběrnici. Reálně je však možné splnit maximálně dva požadavky zároveň. Tomu odpovídají tři typy endpointů.

Typ interrupt slouží pro rychlé upozorňování na stav zařízení. Má rezervované pásmo a opravu chyb, množství dat je však silně limitováno. Typ se používá tam, kde je potřeba pravidelně posílat malé množství dat, např. v myších.

Typ bulk podporuje velké množství dat a opravu chyb, nemá však rezervované pásmo. Slouží tam, kde je potřeba přesný přenos velkého množství dat - úložiště, tiskárny, atd.

Typ isochronous přenese velké množství dat v rezervovaném pásmu, nicméně při chybě

přenosu se nepokusí zotavit. Hodí se tam, kde je rychlost a latence přenosu důležitější než konzistence dat, např. pro webkamery.

Vlastní komunikace začíná připojením zařízení na sběrnici. Po ustálení napětí je adresa zařízení nastavena na 0. Hostitel zařízení přiřadí unikátní adresu, přes endpoint 0 si načte možné konfigurace a následně jednu z konfigurací vybere. Zařízení se inicializuje, nastaví případné další endpointy a je připraveno pro komunikaci se svým ovladačem na straně hostitele.

4.2 libUSB

Pro vývoj aplikací pracujících s USB je možné použít knihovnu libUSB. Ta v pracuje nad nízkoúrovňovými ovladači USB a umožňuje práci se sběrnicí z uživatelského prostoru. Existují dvě verze knihovny s nekompatibilním rozhraním, uživatelská aplikace by ideálně měla rozhraní abstrahovat a být na knihovně nezávislá. Reference ke knihovně je k dispozici na [2].

Verze 0.1 je nyní zastaralá, na Linuxových systémech není dále vyvíjena a její implementace typicky bývá pouze adaptér nad verzí 1.0. Na systémech Windows je však tato verze podporovaná lépe.

LibUSB 1.0 je modernější, aktivně vyvíjená verze knihovny, která je v této práci použita pro demonstrační aplikaci. Na rozdíl od verze 0.1 podporuje kontexty, asynchronní I/O operace a isochronní přenosy. Nevýhodou je horší podpora na systémech Windows.

4.3 Arrow SoCKit

SoCKit je evaluační deska platformy Cyclone V ve verzi 5CSXFC6D6F31 [9]. Ta disponuje (mimo jiné) 110 k logických elementů v FPGA a dvěma jádry ARM A9. Na desce je pak (mimo jiné) osazeno 1 GB RAM pro MCU, konektor USB OTG, čip USB Blaster pro JTAG, převodník mezi USB a UART a rozšiřující HSMC port, který umožní připojení dvou kamer a několika LED pro otestování funkce.

4.4 Altera IP

Vývojové prostředí Quartus II obsahuje knihovnu s optimalizovanými komponentami, od nejjednodušších, jako registry a logické členy, až po procesory. Pro komponenty je typicky dostupná dokumentace on–line. Tato sekce zběžně popisuje komponenty použité v návrhu či při vývoji.

4.4.1 FIFO

FIFO, jak napovídá název, je fronta typu first in, first out. Implementace dostupná v IDE je implementována nad blokovými pamětmi zařízení. Dostupných je několik variant rozhraní. Varianta synchronizovaná dvěma hodinovými signály, jedním pro zápis, druhým pro četní, je vhodná pro převod mezi hodinovými doménami. Může mít stejnou, nebo různou šířku čtecího a zápisového portu, což je použitelné pro převod mezi různě širokými sběrnicemi. Varianta synchronizovaná jedním hodinovým signálem je použitelná jako klasická vyrovnávací paměť. Kompletní referenci lze nalézt v [5].

Za zmínku stojí zpoždění od zápisu dat ke změně na rozhraní pro čtení, které silně závisí na nastavení komponenty. Může dojít k situaci, kdy je do FIFO zapsáno slovo, nicméně ke změně příznaku `EMPTY` dojde až za dva hodinové cykly. Pokud je obvod závislý na okamžité změně (např. detekce ukončení příjmu dat), může dojít k souběhu.

4.4.2 SignalTap

SignalTap je logický analyzátor, který je možné jako IP vložit do architektury FPGA a komunikovat s ním přes rozhraní JTAG. Analyzátor musí být připojen na hodiny domény, ve které se nacházejí sledované signály. Dále využívá prostředky FPGA a stává se součástí návrhu, proto ho do jisté míry ovlivní. Nad těmito nevýhodami však výrazně převyšují výhody (oproti klasickému, externímu analyzátoru). Analyzátorem je možné sledovat interní signály, jejichž vyvedení na externí piny (pokud jsou vůbec k dispozici) by vyžadovalo změny rozhraní komponent. Pokud jsou hodiny v pořádku, synchronizace jejich použitím je další výhodou — SignalTap může být pro dané potřeby přesnější, než běžný externí analyzátor. Referenci použití analyzátoru lze nalézt v [4].

4.5 Protokol AMBA 4 AXI4–Stream

AMBA 4 AXI4–Stream je protokol pro výměnu dat mezi komponentami. Je použit jako rozhraní pro výměnu dat s frameworkem RSoC, proto je i jedním z hlavních rozhraní akcelerátoru snímání. Většina signálů rozhraní není povinných a některé skutečně nejsou v akcelerátoru použity. V této kapitole bude popis omezen jen na signály a parametry, které použity jsou, a vztahy mezi nimi. Kompletní specifikaci rozhraní lze nalézt např. v [1].

Sběrnice propojuje jedno nebo více zařízení *master*, které odesílají data, s jedním nebo více zařízeními *slave*, která data přijímají. Sběrnice je parametrizována parametrem n, který udává počet bytů datové sběrnice `TDATA[(8n-1) downto 0]`. Pokud zařízení neodesílá dostatek bytů pro naplnění celé šíře sběrnice, jsou platné byty označeny v masce `TKEEP[n-1 downto 0]`, ve které každý bit odpovídá platnosti příslušného bytu v `TDATA`. Kromě ukončování streamu může být signál použit i pro snížení šířky sběrnice, čehož bylo využito v raných fázích vývoje.

Pro řízení přenosu slouží signály `TREADY`, který je řízen zařízením *slave* a značí jeho připravenost přijímat data, a `TVALID`, který je řízen zařízením *master* a značí platnost dat na `TDATA`. Protokol mezi signály `TVALID` a `TREADY` je typu show–ahead, kde `TREADY` slouží spíše jako potvrzení transakce, než signalizace pro možnost zápisu. *Master* totiž nesmí na signál `TREADY` čekat, ale musí data vystavit na sběrnici a nastavit signál `TVALID`. *Slave* může (ale nemusí) počkat na signál `TVALID`, po jeho nastavení přečíst hodnotu `TDATA` a nastavit `TREADY`, čímž transakci potvrdí.

Posledním použitým signálem je `TLAST`, který značí poslední zapsané slovo v paketu. Pakety odděluje, signalizuje tím propojovací logice, že v daném paketu nepřijdou žádná další data a je možné odeslat data z vyrovnávacích pamětí. Dle specifikace je možné odeslat prázdné slovo (všechny bity `TKEEP` nastaveny na 0) s nastaveným signálem `TLAST`, čímž se daný paket ukončí i po odeslání všech dat.

4.6 RSoC

RSoC je komerční framework pro přenos dat mezi FPGA a systémem GNU/Linux, který vznikl jako diplomová práce na VUT FIT [12]. Další informace lze nalézt na stránkách

projektu [11]. Na straně FPGA jsou jeho rozhraním akcelerátory, s rozhraním AXI–Stream master pro přenos z GNU/Linuxu do FPGA, AXI–Master pro opačný směr a AXI–Lite pro uživatelsky definované registry. Mezi AXI–Lite a registry je k dispozici adaptér, který řeší režii spojenou s přenosem a adresováním. Výrazně zjednodušena je i práce se streamy, protože framework řeší veškerou režii spojenou s přenosy DMA.

Pro stranu GNU/Linuxu obsahuje modul, který dvojici rozhraní AXI–Stream propojí se souborem v `/dev` a uživatelské registry umožní namapovat do paměti. Po nezbytné inicializaci jsou pak data přenášena čtením ze (resp. zápisem do) souboru.

Z vlastních rozhraní AXI–Stream framework podporuje podmnožinu. Dle specifikace je nekompletní slovo povoleno pouze při ukončení paketu signálem `TLAST` a musí být zarovnáno směrem k LSB (stejně tak odpovídající `TKEEP`). Dále jsou s každým slovem s nastaveným signálem `TLAST` vyprázdněny interní vyrovnávací paměti, proto na první pohled výhodná technika popsaná ve specifikaci (nastavení `TLAST` s každým slovem) vede k výraznému zhoršení výkonu. Tato omezení musela být vzata při návrhu přenosu dat.

Součástí frameworku jsou i ukázkové aplikace, z nichž jedna je určena na měření přenosové rychlosti mezi FPGA a GNU/Linuxem při použití jednoho kanálu. Při použití 100 MHz hodin na straně FPGA dosahuje přes 210 MB/s, což je pro tuto aplikaci více než dostatečné. K dispozici je i testbench a projekt pro Modelsim, který umožňuje uživatelskou komponentu propojit s virtuálním frameworkem a otestovat její kompatibilitu.

4.7 Buildroot

Buildroot je systém pro cross–kompilaci linuxových systémů na vestavěná zařízení. Jeho manuál je k dispozici na [3]. Jedná se o sadu makefile a skriptů, která po konfiguraci stáhne všechny potřebné zdrojové soubory, sestaví toolchain, přeloží a sestaví kernel, kořenový filesystém, a další potřebné komponenty. Pro některé vývojové kity, mezi kterými je i Arrow SoCKit, obsahuje předdefinovanou konfiguraci a patche. Ačkoliv je s Buildrootem možné sestavit plnohodnotný systém včetně GUI, nejsou podporované některé pokročilé funkce, jako balíčky nebo kompilace toolchainu běžícího na cílovém systému. Pro sestavení lehkotonážního a kompaktního systému pro vestavěné zařízení je však ideální.

Kapitola 5

Návrh akcelerátoru snímání

Akcelerátor snímání slouží k provedení jedné fáze snímání a odeslání sejmutých snímků do paměti RAM. Specifikace požadavků na jeho funkcionalitu a zevrubný návrh byl proveden v kapitole 3. V této kapitole je popsán podrobný návrh a implementace komponenty akcelerátoru a její napojení na framework RSoC.

Rozhraní akcelerátoru je tvořeno rozhraním AXI Stream Master pro přenos dat, AXI Lite pro práci s registry, signály pro řízení LED a sběrnicemi pro konfiguraci kamer a čtení jimi generovaných dat.

Akcelerátor je aktivován zápisem do určeného registru. Je provedeno vyprázdnění vyrovnávacích pamětí a aktivován řadič, který je zodpovědný za přesné spouštění PWM a kamer. Snímky z kamer jsou synchronizovány do hodinové domény akcelerátoru, převedeny na 8 b hloubku, ořezány a zmenšeny. Pro přenos tří obrazových kanálů jedním kanálem RSoC jsou data rozdělena do paketů a jako jeden stream vkládána do výstupní fronty. Z výstupní fronty jsou přeposlána do frameworku RSoC. Dokončení provedení fáze je signalizováno hodnotou registru.

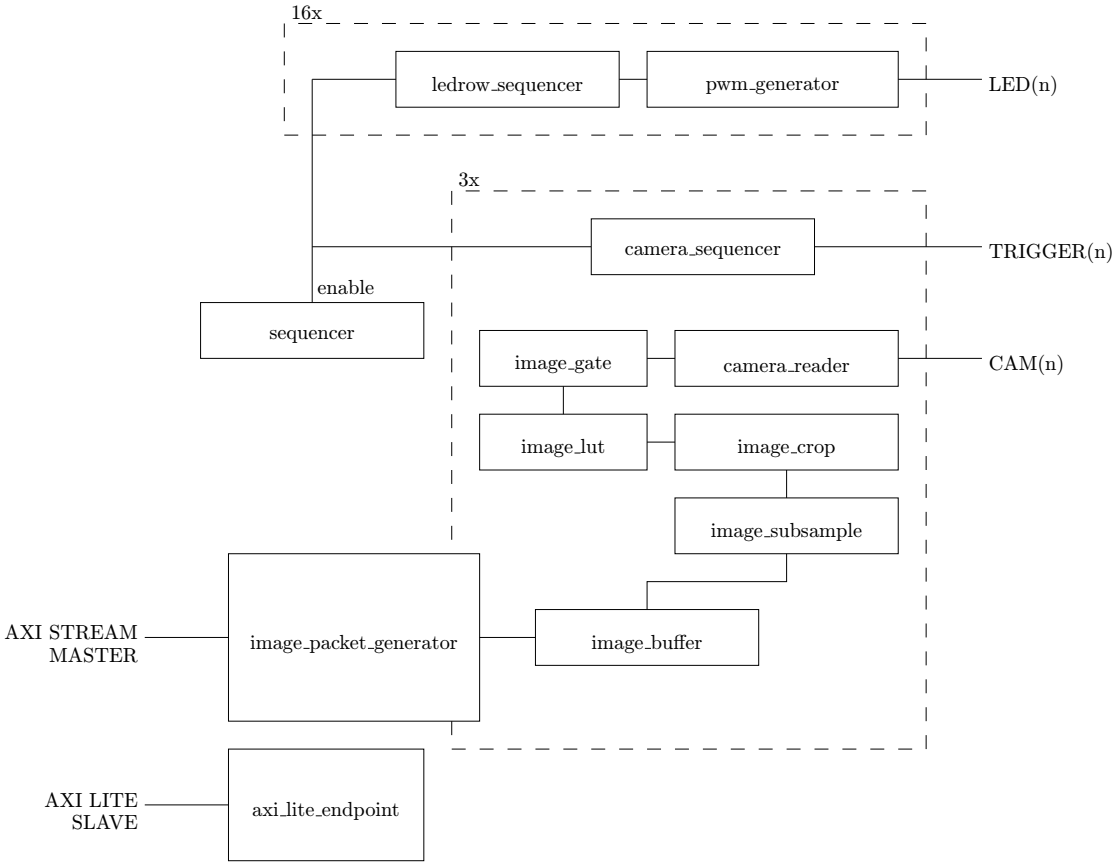
Komponenty zpracování obrazu jsou propojeny jednoduchou sběrnicí, která vychází z rozhraní kamery. Obsahuje deset datových signálů (po zmenšení hloubky jsou dva signály nevyužity), signál `LINE_VALID`, který je nastaven při přenosu jednoho řádku pixelů, `FRAME_VALID`, který spojuje řádky do obrazu, a `BUS_VALID`. Sběrnice neobsahuje žádný ekvivalent signálu `READY` — je určena pro přenos obrazu řadou filtrů a jeho následný zápis do vyrovnávací FIFO, ze které se přesune do paměti procesoru. Pokud se vyrovnávací FIFO naplní, jedná se o kritickou chybu akcelerátoru.

Převod streamu na pakety před odesláním do RAM je nutný, protože kvůli zdrojům v FPGA nejsou k dispozici tři akcelerátory, ale pouze jeden. Řízení senzoru tedy musí přečíst hlavičku paketu, alokovat potřebné množství paměti, přečíst data a opakovat cyklus. Hlavička paketu musí obsahovat identifikátor kamery a velikost dat, která budou následovat.

Při vytváření obrazových paketů jsou řádky snímku sjednoceny do jednoho proudu a jejich hranice ignorovány. Pokud by z každého řádku byl jeden obrazový paket, některé extrémnější optimalizace (např. dvoupixelové řádky pro detekci prstu) by generovaly několikanásobně více dat přenosem hlaviček než vlastních dat.

Signály ovládání kamer musí být synchronizovány do domény hodin generovaných pro kamery (nikoliv hodin z kamer přijatých). Akcelerátor měl být původně synchronizován hodinami s frekvencí 100 MHz, pro zjednodušení této synchronizace však budou mít hodiny frekvenci 96 MHz a hodiny pro kamery budou generovány pouhým vydělením dvěma. Synchronizace takového přechodu je triviální.

Architektura akcelerátoru je na obrázku 5.1. Akcelerátor se skládá z komponent, jejichž



Obrázek 5.1: Blokové schéma akcelerátoru snímání. Signály registrů vedoucí od komponenty `axi_lite_endpoint` do všech ostatních komponent zobrazeny nejsou.

přehled a detailní popis je v následujících sekcích.

- `ledrow_sequencer` je ovladač PWM jedné skupiny LED. Vstupními signály jsou časy začátku a konce osvětlení, signál startu fáze a reset. Výstupní signál je určen pro ENABLE vstup řízeného PWM.
- `camera_sequencer` je ovladač kamery. Vstupními signály jsou čas startu snímání, signál startu fáze a reset. Výstupním signálem je signál TRIGGER pro kameru.
- `sequencer` řídí dílčí ovladače kamer a skupin LED.
- `pwm_generator` generuje PWM pro jednu skupinu LED. Vstupem je střída a ENABLE.
- `camera_reader` je komponenta pro čtení dat z paralelní sběrnice kamery, a jejich synchronizaci do hodin systému.
- `image_gate` blokuje nevyžádané obrazy z kamery (například po neočekávaném resetu kamery).
- `image_lut` implementuje obecnou bodovou funkci pro převod desetibitové hloubky na osmibitovou.

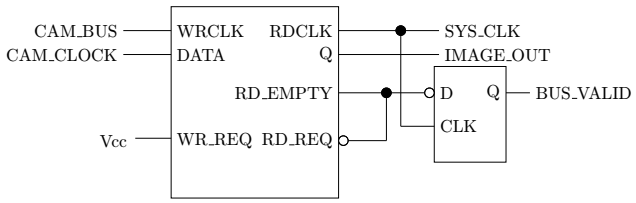
- `image_crop` provádí ořezání obrazu.
- `image_subsample` provádí podvzorkování obrazu.
- `health_monitor` monitoruje správný průběh snímání.
- `image_buffer` uchovává obrazová data z kamery do vytvoření paketu.
- `packet_generator` dle naplnění vyrovnávacích pamětí převádí streamy dat na pakety a ty odesílá do RAM.
- `rsoc_packet_generator` generuje pakety optimalizované pro RSoC.

5.1 Komponenty akcelerátoru snímání

5.1.1 Camera Reader

Komponenta `camera_reader` synchronizuje signály z hodinové domény kamery na doménu systémových hodin. Na výběr jsou dvě možnosti. První je použití asynchronní FIFO. Druhou možností by bylo synchronizovat signál PIXCLK (např. pomocí několikanásobného registrování) a na jeho sestupné hraně číst signály z kamer. V porovnání s FIFO je tato možnost úspornější, lze ji však použít pouze pokud je frekvence systémových hodin několikrát vyšší než frekvence hodin kamer.

Protože je frekvence systémových hodin jen o málo vyšší než frekvence hodin kamer, komponenta je postavena na asynchronní FIFO. Šířka slova ve FIFO je nastavena na šířku sběrnice pro přenos obrazu bez signálu `BUS_VALID`, který je použit jako příkaz zápisu do FIFO `WR_REQ`. Počet slov je nastaven tak, aby při okamžitém čtení dat, která jsou k dispozici, nedocházelo k přetečení FIFO. Okamžité čtení je zajištěno přivedením negace výstupního signálu `RD_EMPTY` na vstupní signál `RD_REQ`. O jeden takt zpožděná negace `RD_EMPTY` je také použita jako signál `BUS_VALID` výstupní obrazové sběrnice. Zapojení FIFO pro účely čtení dat kamery je na obrázku 5.2.



Obrázek 5.2: Schéma zapojení DC FIFO pro synchronizaci signálů mezi hodinovými doménami kamery a systému. Hodnota na sběrnici `CAM_BUS` je přijata každý cyklus hodin `CAM_CLOCK`. Po synchronizaci do hodin `SYS_CLK` FIFO zruší příznak `RD_EMPTY`, čímž se vyžádá přečtení uloženého slova. To je k dispozici příští takt, kdy je opět nastaven příznak `RD_EMPTY`, proto je signál `BUS_VALID` o jeden takt spožděn.

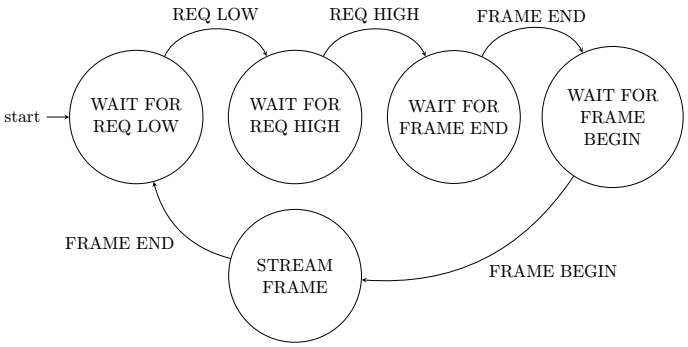
5.1.2 Image Gate

Některé komponenty akcelerátoru jsou postaveny na faktu, že kamera odesílá obrazová data pouze tehdy, je–li k tomu vyzvána signálem `TRIGGER`. Tento *triggered mód* je však nejdříve potřeba zapnout v konfiguraci kamery. K tomu, aby se vyrovnávací paměti akcelerátoru

neplnily nechtěnými daty, slouží právě komponenta `image_gate`. Ta po resetu zahazuje všechna obrazová data, dokud nedetekuje náběžnou hranu signálu `IMG_REQ`. Poté počká na konec aktuálního snímku (sestupná hrana `FRAME_VALID`), a následující snímek již přepošle na výstup. Po přeposlání jednoho snímku opět blokuje.

Komponenta byla užitečná při vývoji, kdy ještě nebyla k dispozici konfigurace kamery, aby bylo možné přečíst data z právě jednoho snímku. Při inicializaci systému ošetří data odeslaná kamerou od startu po vstup to triggered módu. Navíc při neočekávaném resetu konfigurace kamery či nechtěném zrušení triggered módu je díky ní možné upozornit řízení senzoru na tento (nechtěný) stav.

Stavový stroj komponenty je na obrázku 5.3.



Obrázek 5.3: Stavový stroj komponenty `image_gate`. Snímek je přeposlán pouze ve stavu `STREAM FRAME`, v ostatních stavech je zablokován.

5.1.3 Look Up Table

Účelem LUT je převod z 10 b hloubky obrazu, který poskytuje kamera, na 8 b hloubku, která je použita v uživatelské aplikaci, pomocí libovolné bodové funkce. Implementace LUT je přímočará. Obsahuje pole 1024 slov šířky 8 b hodnota na vstupní sběrnici je použita jako index, na kterém leží hodnota zapsaná na výstup. Pro úsporu zdrojů je použita pouze jedna LUT pro všechny tři kanály.

V praxi LUT ukazuje spíše jako experimentální záležitost a její inicializace je zbytečnou zátěží klientské aplikace. Proto je ve stávajícím firmware komponenta `lut_bypass`, která LUT obchází a hloubku redukuje ořezáním dvou nejméně významných bitů. Pro zjednodušení návrhu FPGA firmware není tato komponenta implementována, její funkcionalita je nahrazena procedurou v řízení senzoru.

5.1.4 Health Monitor

Komponenta `health_monitor` slouží pro diagnostiku akcelerátoru. Jejími vstupy jsou chybové signály z ostatních komponent, které komponenta ukládá do registru a vystavuje do paměťového prostoru akcelerátoru. Kromě toho monitoruje některé další podmínky, které musí platit během snímání — například po náběžné hraně signálu TRIGGER musí kamera do 50 ms začít odesílat obrazová data.

5.1.5 Mezipaměť obrazových dat

Protože v paketu odesílaném do RAM jsou vždy data jen z jedné kamery, je potřeba vyrovnávací paměť pro uchování dat ostatních kamer před jejich odesláním. Generátor paketů pak při odesílání dat mezipaměti střídá. Implmentaci této paměti zajistí komponenta `image_reader_queue`. Při návrhu komponent se ukázalo, že toto místo je zároveň vhodné pro převod z 8bitové šířky sběrnice kamery na šířku datové sběrnice frameworku RSoC. Protože Altera poskytuje FIFO s rozdílnou šířkou vstupu a výstupu, Image Reader Queue měl být založen právě na této komponentě. Ukázalo se však, že návrh bude zkomplikován hlavně režií zarovnávání dat do FIFO, proto byla komponenta navržena jinak a výsledná architektura je rozsáhlejší, nicméně jednodušší.

Původní návrh

V původním návrhu byla zapisována procesem `writer`, který signalizuje každý zapsaný byte procesu `reader` a po přijetí všech dat snímku (signál `FRAME_VALID`) zarovná počet zapsaných bajtů na násobek šířky výstupního portu FIFO. To je nezbytné, protože částečně vyplněná slova se z FIFO číst nedají.

Proces `reader` měl na požadavek signálem `RD_READY` číst data dostupná z FIFO, na základě čtení a signalizace zápisu spravovat počitadlo bytů ve FIFO (výstup `RD_SIZE`) a na základě počtu pak určit masku platnosti bytů výstupního slova (`RD_KEEP`). Vlastní počitadlo bylo výhodné kvůli zpoždění výstupních signálů FIFO.

Čtecí komponenta, která čte data z `image_reader_queue` a balí je do paketů, měla přečíst hodnotu `RD_SIZE`, z té určit velikost paketu a následně z přečíst odpovídající počet slov. Když `RD_SIZE` není násobkem šířky výstupní sběrnice v bajtech, platné bajty posledního přečteného slova jsou označeny maskou `RD_KEEP`, zbylé jsou neplatné. Pokud by však čtecí komponenta takovou velikost přečetla ještě před přijetím všech zbývajících dat obrazu, FIFO neplatné bajty vyplní novými a čtecí komponenta dostane více dat, než odpovídá hodnotě `RD_SIZE`. Ošetření této situace by na straně čtecí komponenty znamenalo rozdělení slova a odeslání nového paketu. Proto měl být případ ošetřen na straně `image_reader_queue`, která, dokud nebyl ukončen přenos snímku signálem `FRAME_VALID`, měla velikost dat zaokrouhlit na nejbližší nižší násobek šířky výstupní sběrnice v bajtech.

Nový návrh

Nový návrh je rozdělen do dvou komponent, což usnadnilo testování, a využívá frontu se stejnou šířkou vstupu a výstupu. Ačkoliv je potřeba šířku převádět ručně, výsledkem je průhlednější a robustnější logika.

První komponenta zajišťuje převod z osmibitové na 64 bitovou sběrnici. Jejím vstupem je obrazová sběrnice, výstupem signály `DATA`, `KEEP`, `LAST` a `VALID`. Konečný automat vstupními daty postupně naplňuje buffer pro výstupní data a registr pro výstup `KEEP`. Pokud je snímek ukončen, jsou ihned odeslány hodnoty `DATA` a `KEEP` s nastaveným signálem `LAST`. Pokud snímek ukončen není, automat s odesláním čeká, dokud není buffer naplněn a přijata další hodnota. Tím je zajištěno, že komponenta během přenosu snímku vždy obsahuje alespoň jeden byte dat a je schopná ukončit přenos neprázdným výstupem. Stavový automat komponenty je na obrázku 5.4.

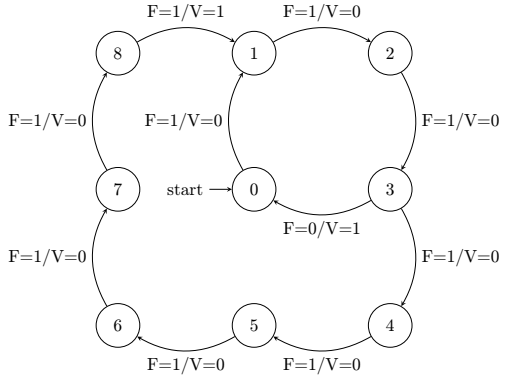
Rozšířením na 64 b se ztrácí informace o struktuře obrazu a snímek je přenášen již jen jako datový proud. Pro generování posledních paketů je však nutné vědět, že byl přijat

poslední byte snímku, a je tedy možné vytvořit paket s libovolně malou velikostí. Proto komponenta ukládá a generátoru paketů poskytuje příznak posledního přijatého bytu.

Vlastní data jsou již standardně uložena do FIFO, ze které jsou postupně čtena a sdružována do paketů komponentou pro generování paketů.

Při přijetí posledního slova je přijat i údaj o počtu platných bytů tohoto slova (numerické vyjádření masky `KEEP`). Údaj je uložen a poskytnut generátoru paketů. Tento systém je zvolen proto, že je jednodušší poslat zaokrouhlená slova a případné neplatné byty zahodit na straně služby řízení. Mimo jiné se výrazně zvýší kapacita výstupní vyrovnávací paměti, která by masku `KEEP` musela uchovávat pro každé slovo.

V novém návrhu není potřeba ručně počítat počet slov ve FIFO (díky použité variantě FIFO je údaj na jejím počitadle dostupný v taktu následujícím po zápisu) a zarovnávat její obsah (nekompletní slovo je uloženo v registru posledního slova). Proto je varianta i přes mírně větší rozsah vybrána jednodušší a byla vybrána do finálního návrhu.



Obrázek 5.4: Stavový stroj komponenty pro rozšíření sběrnice. Číslo stavu odpovídá aktuálnímu zaplnění vyrovnávacího bufferu, `F=X` je hodnota vstupního signálu `FRAME_VALID`, `V=X` je hodnota výstupního signálu `VALID`. Uvnitř snímku (`F=1`) je postupně naplňována kapacita bufferu. V cyklu, kdy buffer přeteče (`8 -> 1`), je aktuální slovo odesláno do výstupu (`V=1`). Po ukončení snímku (`F=0`) je v kterémkoliv stavu odesláno aktuální slovo s nastaveným příznakem přijetí posledního slova a automat přejde do stavu `0`, jak je ukázáno na stavu `3` (na ostatních stavech kvůli přehlednosti ukázáno není). Tak je zajištěno, že příznak posledního slova přijde vždy s alespoň jedním bytem dat.

5.1.6 Image Packet Generator

Image Packet Generator měl původně být rozdělen na dvě komponenty, jednu pro vytváření paketů, a jednu pro střídavé odesílání paketů ze třech kamer. Toto rozdělení mělo být výhodné pro vývoj a testování. Komponenty by však mezi sebou musely mít kromě klasického páru signálů `VALID` a `READY` obdobnou signalizaci pro vyžádání a potvrzení paketu. Jednodušší možností by bylo, aby byty počítala i komponenta pro odesílání. Tím však komponenta pro vytváření paketů ztrácí smysl - jedinou její úlohou je zápis hlavičky, což je triviální.

Během příjmu snímku (signál `FRAME_VALID`) Image Packet Generator sleduje streamy a čeká, dokud počet dostupných bytů v některém z nich nedosáhne nastavitelného prahu. Pokud dosáhne, z informací poskytnutých vyrovnávací pamětí vybrané kamery je vygenerována hlavička, která se skládá z indexu kanálu, počtu přenesených bytů a počtu neplatných bytů na konci paketu. Následně jsou do služby řízení odeslána data paketu. Pokud všechny

vyrovnávací paměti signalizují přijetí posledního slova snímku, generátor přečte všechna dostupná data ze všech kamer, nezávisle velikosti paketu. V extrémním případě může být odeslán poslední paket s jedním bytem dat, to ale jednou, při ukončení, nevadí.

Je žádoucí, aby z trojice streamů byl vždy vybrán ten s největší kapacitou, aby byl systém odolnější proti přetečení vyrovnávací paměti. Jednoduchá simulace však ukázala, že stačí vhodně definovat dva prahy, nejdříve kontrolovat vyšší a pokud nebyl překročen v žádném ze streamů, kontrolovat nižší. Přístup má výhodu i v tom, že sám od sebe zajišťuje minimální velikost prahů.

Kvůli protokolu sběrnice AXI–Stream musí plánovač na výstup umístit hlavičku a počkat, až druhá strana potvrdí příjem. To pro tuto aplikaci není vhodné, protože se může začít plnit jiný kanál, který by bylo potřeba odeslat přednostně. Pro plánování by bylo výhodné, kdyby byl připravený paket odeslán naráz. Proto může být mezi plánovač a rozhraní RSoC umístěna další vyrovnávací paměť. Plánovač pak může umístit paket do výstupní vyrovnávací paměti pouze pokud je v ní dostatek volného místa a nebude se muset čekat na vlastní odeslání paketu.

Výhodou výstupní fronty by byl i fakt, že její paměť je dostupná pro všechny kanály. Díky tomu jsou přenosy stabilnější a odolnější vůči výkyvům rychlosti přenosu, zvláště na začátku nebo konci snímání. Dále se na obsah výstupní

Poté, co jsou odeslána data trojice snímků, je konec přenosu signalizován prázdným paketem — hlavičkou bez čísla kamery a následných dat. Prázdný paket se ukázal výhodný jak pro řízení senzoru, tak pro samotný akcelerátor, protože se zjednodušila implementace generování příznaku `LAST` při dokončení přenosu.

Použitá simulace byla velmi jednoduchá, jejím účelem nebyla přesná předpověď chování, ale hrubý odhad zaplnění front při daných rychlostech zápisu a čtení metodě výběru fronty. Jedná se o jednoduchý skript v jazyce Python, který vyrovnávací paměti reprezentuje jejich kapacitou a obsahuje procesy pro zápis a čtení. Výsledky simulace zobrazí pomocí systému matplotlib.

5.1.7 Ledrow Sequencer

Ledrow Sequencer je jednoduchá komponenta pro časování jedné skupiny LED. Obsahuje počitadlo, které je na vzestupné hraně signálu `ENABLE` nastaveno na 0. Po dosažení hodnoty na signálech `LED_START` je výstup `LED_ENABLE` nastaven na 1, po dosažení hodnoty na `LED_END` je výstup nastaven na 0 a počitadlo zastaveno. Počitadlo je zastaveno a nastaveno na 0 i na sestupné hraně signálu `ENABLE`.

5.1.8 Camera Sequencer

Camera Sequencer funguje podobně jako Ledrow Sequencer, jen místo obecného pulzu generuje krátký pulz s daným zpožděním, jehož náběžná hrana spustí sejmutí snímku.

5.1.9 Sequencer

Sequencer je komponenta pro řízení provedení celé fáze. Jejím úkolem je na vzestupnou hranu signálu `TRIGGER` povolit ovladače kamer a LED a signál `TRIGGER` zakázat. Poté čeká na přenesení dat ze všech kamer, které je signalizováno sestupnou hranou signálu `FRAME_VALID`. Ovladače jsou deaktivovány, sequencer počká na vyprázdnění výstupních front a opět povolí signál `TRIGGER`.

5.1.10 PWM Generator

Generátor PWM je, vzhledem ke své aplikaci, velmi jednoduchý. Vstupem je signál `ENABLE`, na jehož náběžné hraně je resetováno 8 b počitadlo a odstartována jeho funkce. Když je počitadlo na hodnotě 0, výstup je nastaven na 1. Když počitadlo dosáhne hodnoty střídy (vstup `DUTY_CYCLE`), hodnota výstupu je nastavena na 0. Při dosažení hodnoty 255 je počitadlo ihned vyresetováno; nastavením střídy na 255 bude díky tomu výstup nastaven stále.

5.1.11 RSoC Packet Generator

Nastavení signálu `TLAST` pouze při posledním odeslaném bytu snímku vede k přetečení fronty. Obrazová data totiž začnou s příjmem snímku zaplňovat vyrovnávací paměti frameworku. Po jejich naplnění jsou odeslány do procesoru, což však trvá příliš dlouho a přetečou vyrovnávací paměti akcelerátoru. Rozměry řádků nejsou k dispozici (a jejich použití není ani pro tento účel vhodné). Obrazové pakety vytvořené generátorem paketů by bylo před odesláním do frameworku potřeba parsovat (mezi generátorem a frameworkem je FIFO), navíc jsou jejich délky optimalizovány pro generátor, nikoliv pro framework. Proto je signál `TLAST` aktivován uměle tak, aby byly optimálně využity vyrovnávací paměti frameworku a jejich vyprazdňování netrvalo příliš dlouho, nezávisle na struktuře proudu obrazových dat.

5.2 Řešení selhání akcelerátoru

V akcelerátoru může dojít k několika situacím, při kterých nelze dokončit snímání obrazu. Nejtypičtějším příkladem je přetečení front způsobené pomalým čtením ze strany služby řízení. To způsobí ztrátu částí snímku. Může dojít k chybě hodin generovaných kamerami a následným potížím při synchronizaci signálů kamer. Nejjednodušším řešením je v takových případech reset, který může být generovaný ze strany akcelerátoru nebo ze strany služby řízení.

Reset sebe sama se zdá výhodný z pohledu služby řízení, která je díky tomu zdánlivě jednodušší. Kvůli ztrátě obrazových dat však služba řízení upozorněna být musí a musí obsahovat procedury pro řešení nastalé situace. Přidání resetu akcelerátoru tyto procedury příliš nezkomplikuje. Reset ze strany služby řízení navíc odpovídá filozofii architektury senzoru, v rámci které je v akcelerátoru implementováno pouze snímání, a veškeré řízení je zajištěno službou.

Kapitola 6

Návrh řízení senzoru

Jak bylo navrženo v kapitole 3, řízení senzoru je rozděleno do tří částí. Pro řízení periferií je použit systém GNU/Linux, komunikaci s FPGA zajišťuje framework RSoC. Proto se tato kapitola zaměřuje na detailní návrh třetí části, služby řízení a služby aktualizace.

6.1 Spouštění služeb

Po startu GNU/Linuxu je automaticky spuštěna služba aktualizace. Ta musí zkontrolovat, či se systém nenachází uprostřed přerušené aktualizace, a vyřešit případné problémy. Při jejím úspěšném ukončení je spuštěna služba řízení. Po ukončení či pádu služby řízení je služba řízení. Speciálním návratovým kódem může služba řízení vynutit spuštění služby aktualizací po svém ukončení.

6.2 Protokol komunikace s uživatelským zařízením

Jak bylo uvedeno v návrhu, na sběrnici USB budou pro komunikaci k dispozici endpointy typu bulk a control. Při komunikaci je potřeba přenášet obrazová data, konfiguraci a stavové informace. Ve stávající generaci je pro konfiguraci a stavové informace použit typ control, pro obrazová data typ bulk. Použití typu control je výhodné ve strukturovaném přenosu dat, kdy je každou stranou přijata či odeslána vždy celá zpráva, nevýhodou menší propustnost sběrnice (která pro tento účel není na překážku). Výhodou typu bulk je právě vyšší propustnost, které je využito pro přenos obrazových dat.

Při implementaci USB zařízení přímo v jádře je tento přístup výhodný, nicméně při použití některých existujících implementací by bylo vhodnější komunikovat jen přes endpointy typu bulk. Příkladem je USB Serial Gadget, který pro komunikaci s hostitelským zařízením implementuje protokol sériové linky. Z pohledu aplikačního protokolu by podobným řešením byla komunikace přes TCP/IP, pro kterou by se mohl znovupoužít kód pro dvojici endpointů typu bulk. Dalším argumentem pro dvojici endpointů typu bulk (TCP/IP nemá ekvivalent endpointu typu control). Komunikace přes dvojici streamů dat však na aplikační úrovni prakticky vždy musí být rozdělena do paketů (ve smyslu objektů, které obsahují hlavičku s délkou zprávy). Dva typy paketů, jeden pro data a druhý pro řídicí zprávy, pak funkcionalitu kontrolního endpointu nahradí.

Protokol komunikace by tak závisel spíše na způsobu komunikace s uživatelským zařízením, od kterého by ve službách mělo být abstrahováno. Proto by musel být navržen tak, aby byl vhodný pro všechny možnosti komunikace. Provoz by byl rozdělen na řídicí

kanál (konfigurace a stavové informace) a datový kanál (odesílání obrazových dat). Kanály odpovídají endpointům typu control a bulk. Z toho plynou omezení a datové formáty obou kanálů. Řídicí kanál přenáší strukturované zprávy, data je možné číst i zapisovat, a maximální velikost je 64 B dat na jeden požadavek. Datový kanál je jednosměrný, neformátovaný a lze jím přenést libovolné množství dat.

Pokud však nebude použit řídicí endpoint a komunikace bude probíhat přes dva endpointy typu bulk (analogicky s Ethernetem), pakety mohou být zcela obecné (formát nezávisí na formátu paketu řídicího endpointu) a sjednotit oba kanály. Výhodou i nevýhodou je nutnost definice vlastního protokolu. Jedná se o jistou režii navíc, nicméně protokol bude společný pro Ethernet i USB.

Použit by mohl být i existující protokol, např. pro RPC. To by však znamenalo nezanedbatelnou režii při přenosu, která je ve světle jednoduchosti dále popsaného protokolu komunikace zcela zbytečná.

Komunikaci se senzorem vždy začíná uživatelské zařízení. Odesílá 32 b příkaz (momentálně je použit pouze jeden bit pro značení zápisu či čtení, šířka 32 b je použita kvůli zarovnání a budoucím rozšířením), 32 b adresy v adresovém prostoru senzoru, 16 b velikost slova dat a 16 b počet slov. Následují data, jejichž velikost je součinem velikosti slova a počtu slov. Taková definice umožňuje přenos jak řady 32 b registrů FPGA, tak velkého objemu obrazových dat po 16 kB stránkách.

V případě čtení senzor odpovídá 32 b návratovou hodnotou. Záporné hodnoty znamenají chybu, kladné hodnoty pak velikost dat, která za návratovou hodnotou následují. Čtená data mohou být rozdělena do více paketů — v základní implementaci je např. při přečtení pěti 4 b registrů vráceno 5 paketů se 4 byty dat —, což je důležité v případě, že po přečtení části požadovaných dat dojde v senzoru k chybě.

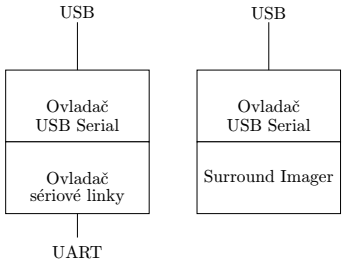
V případě zápisu senzor neodpovídá. Je–li potřeba zapsanou hodnotu verifikovat, je většinou možné ji po zápisu přečíst. Nelze–li hodnotu přečíst, zápis typicky vyvolává akci, která nastavuje stavový registr.

6.3 Komunikace po USB

Ve stávající implementaci je rozhraní navrženo zcela nezávisle na existujících třídách USB. V novém návrhu by se mělo jednat o poslední možnost, jelikož je potřeba psát vlastní driver na obou stranách komunikace, což může vést ke zbytečným chybám. Proto byly jako první vzaty v potaz existující třídy zařízení.

První možností je již zmíněný protokol sériové linky implementovaný modulem USB Serial Gadget. Nejedná se zde o přenos přes UART (viz. obrázek 6.1). Při použití je (jako obvykle) vytvořen soubor v /dev. Data zapsaná do tohoto souboru jsou přenesena do uživatelského zařízení a naopak. Výhodou řešení je jednoduchost. Nevýhodou je malá přenosová rychlost — experimenty bylo zjištěno, že s použitým hardwarem lze přenést cca 3.5 MB/s. To je dostatečné pro kontrolní zprávy, živý náhled ze všech kamer však s takovou rychlostí reálný není.

Druhou možností je kompozitní zařízení, které by kontrolní zprávy přenášelo USB Serial Gadgetem a obrazová data přes USB Mass Storage Gadget. Ten implementuje protokol používaný pro velkokapacitní úložná zařízení. Jako úložiště by byl použit soubor v operační paměti zařízení, do kterého by se přenášel vyžádaný obraz. Ačkoliv řešení na první pohled vypadá použitelně, skrývá několik problémů. Při paralelním provádění snímání, které znamená vysokokapacitní přenos z FPGA do RAM, a přenosu do uživatelského zařízení, které znamená přenos z RAM do souborového systému přes jeho ovladače, by mohlo již



Obrázek 6.1: Použití modulu usb_serial pro komunikaci s uživatelským zařízením. Omezení ryhlosti přenosu je za normálních okolností dané rychlostí sériové linky (stejně tak jsou pro modul sériové linky určena nastavení parametrů komunikace), zde však data pochází přímo ze zařízení a nic nebrání rychlejšímu přenosu dat. Testy však ukázaly, že ani zvýšená rychlost není pro účely senzoru dostatečná.

začít docházet k zahlcování sběrnic. Navíc se jedná o tak nestandardní použití, že se ztrácí výhoda nízké chybovosti díky standardní implementaci. Například kvůli ochraně dat bude muset zařízení být nakonfigurováno jako read-only. Operační systém uživatelského zařízení si při čtení může umístit data do svých vyrovnávacích pamětí a místo nových snímků budou čteny staré.

Třetí možností je vlastní ovladač. Je třeba si uvědomit, že existující implementace nebyly vybírány ze standardizovaných tříd USB, ale ze tříd implementovaných v Linuxu. Ovladač lze psát jako modul pro jádro, výhodnější pro vývoj aplikace a stabilitu systému je však implementace v uživatelském prostoru. To umožňuje modul jádra GadgetFS, který endpointy USB zařízení vystaví jako uzly v `/dev`. Výhodou tohoto přístupu je relativně malý overhead (komunikuje se přímo s ovladačem USB řadiče) a jednoduchost architektury. Klasickou výhodou aplikací v uživatelském prostoru je stabilita — pokud služba havaruje, systém jako takový funguje dále. Jistou nevýhodou by mohla být potřeba přenášet data mezi uživatelským prostorem a prostorem jádra a nutnost přepínání mezi prostory při obsluze požadavků USB. Ve srovnání s dobou potřebnou pro přenos dat po USB je však tato režie zanedbatelná.

GadgetFS bude zkonfigurován pro dva endpointy typu bulk, každý pro jeden směr komunikace, podobně jako adaptér mezi USB a sériovou linkou.

6.4 Komunikace po TCP/IP

Pro komunikaci přes TCP/IP je potřeba řešit zcela jiné problémy, než pro komunikaci přes USB. Ovladače i rozhraní jsou zde standardní, řešit je potřeba zejména aplikační protokol. Oproti komunikaci přes USB je TCP/IP jednodušší. Zařízení musí obsahovat klienta DHCP a fungovat jako server, ke kterému se připojí klient (uživatelské zařízení). Na zařízení se v jeden okamžik může připojit pouze jeden klient, což zjednoduší návrh. Jak bylo zmíněno v návrhu, zavedení UDP pro přenos obrazových dat režimu preview nepřinese výrazné výhody.

6.5 Služba řízení

Služba řízení je zodpovědná za správu nastavení senzoru, správu získaných snímků a jejich případné úpravy. Při svém startu také inicializuje komunikaci s FPGA. Je rozdělena na

několika vrstev. Obecně platí, že efektů vrstvy lze dosáhnout i použitím vrstvy o úroveň níže, ale pomaleji. Díky tomu jsou nižší vrstvy použitelné například pro experimentování se senzorem nebo ladění chyb.

Nejníže je vrstva pro komunikaci s hardwarem. Obsahuje například funkce pro práci se sběrnicí I^2C, komunikaci s FPGA nebo inicializaci spojení mezi FPGA a GNU/Linuxem.

Tuto vrstvu využívá vrstva registrů zařízení. Definuje protokoly, kterými se po sběrnicích komunikuje s komponentami senzoru, například FPGA, kamerami či EEPROM. Kromě použití vyššími vrstvami je jako první dostupná pro uživatelské zařízení, například pro iniciální nastavení registrů, které není vhodné kopírovat do všech fází.

První vrstvou určenou pro běžný provoz je vrstva nastavení fáze. Nastavení senzoru registr po registru by bylo příliš pomalé, proto jsou definovány vysokoúrovňové příkazy, které nastavují větší počet registrů zároveň. Vrstva nastavení fáze tyto příkazy rozdělí na dílčí nastavení. Zároveň umožňuje nastavení nové fáze během provádění stávající a jednorázové odeslání do vrstvy registrů zařízení.

Nejvyšší vrstvou služby řízení je vrstva správy fází. Z pohledu služby řízení je fáze sadou nastavení pro FPGA, kamery a službu samotnou. Při použití této vrstvy uživatelské zařízení přenese několik fází do nastavení senzoru a následně mezi nimi rychle přepíná.

Služba mohla definovat i vyšší vrstvy, počínaje vrstvou pro nastavení profilu. Profil definuje sekvence fází režimu capture. Vrstva nastavení profilu by pracovala ekvivalentně k vrstvě nastavení fáze — profil by přijala jako jeden příkaz a postupně snímala fáze. Přidaná hodnota takové vrstvy je však téměř nulová, protože novou fázi lze jedním příkazem vybrat na pozadí, zatímco se snímá fáze stávající. Proto vyšší úrovně ve službě zahrnuty nejsou.

Otázkou byl požadavek na uložení alespoň 16 fází. To je potřeba, protože snímky musí být kvůli pohodlí uživatele zhotoveny rychle za sebou a pro sejmutí nového se nesmí se čekat na přenos starého do uživatelského zařízení. Toho však dosáhnout lze díky vrstvám správy a nastavení fází. Do vrstvy správy fází budou fáze nahrány při inicializaci senzoru a vrstva nastavení fáze umožní jejich přepínání již během snímání. V senzoru mohou být snímky uloženy až do dokončení snímání.

6.5.1 Adresový prostor

Všechny snímky, nastavení vrstev a registry pro komunikaci s nimi jsou umístěny v adresovém prostoru služby řízení, který je adresován 32bitovou adresou. Nejvýznamnějších 8 bitů adresy jsou použity pro rozdělení prostoru do segmentů jednotlivých funkcí. Toto rozdělení napomáhá ladění chyb, kdy není možné např. přepsat všechna nastavení špatně nakonfigurovanou adresou pro zápis snímku.

Po rozdělení adresového prostoru je k dispozici 256 segmentů, každý o velikosti 16 MB. To však nestačí pro segment snímků, který vyžaduje minimálně 64 MB. Proto je horní polovina segmentů spojena do skupin po šestnácti a vedle 128 16 MB segmentů tak vzniká 8 segmentů o velikosti 256 MB.

Vlastní adresy jsou v některých případech namapované do polí v operační paměti (např. čtení snímků), v jiných případech (např. práce s EEPROM) jsou zcela virtuální.

Prvním segmentem je segment služby. Jako jediný segment je přítomen i ve službě aktualizací, i když s odlišnou definicí registrů. Obsahuje informace o senzoru, např. verzi firwmare a některé příkazy, které se netýkají snímání, např. přepnutí do služby aktualizací. Aby se předešlo chybám, adresy registrů ve službě aktualizací a službě snímání jsou navzájem disjunktní.

Dalším segmentem je segment registrů zařízení, který je rozhraním pro službu registrů

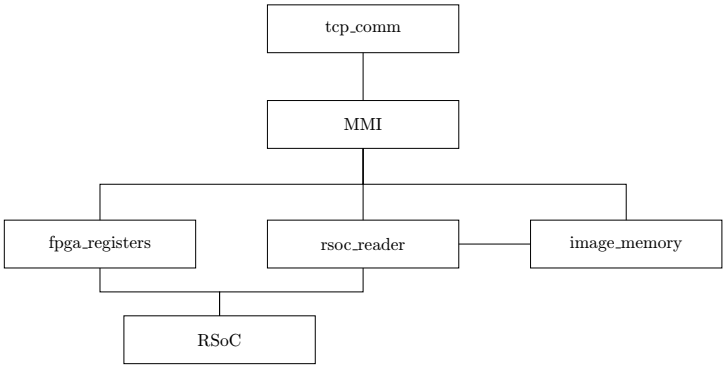
zařízení. Ve svém adresovém prostoru obsahuje všechny registry FPGA a kamer a je do něj namapována uživatelská část EEPROM.

Vrstva správy fází a vrstva nastavení fáze sdílí segment fází. Ten v první řadě definuje rozložení fází tak, aby bylo fázi možné zapsat co nejmenším počtem příkazů. Dále obsahuje pole pro uložení 32 fází a registry pro zápis vybrané fáze, kterými lze fázi jak pouze zapsat, zapsat a spustit nebo jen spustit. V poli je první položka mapovaná přímo na vrstvu registrů zařízení, což odpovídá vrstvě nastavení fáze a umožňuje vybrat fázi, upravit ji a následně spustit. To je důležité například pro kalibraci senzoru.

Všechny snímky a výsledky případných funkce zpracování obrazu jsou uloženy v segmentu snímků. Do tohoto segmentu by měly zapisovat všechny komponenty, které mají uživatelsky definovatelnou adresu pro výstup, aby se předešlo chybám při špatné konfiguraci adresy.

Vrstva pro komunikaci s hardwarem svůj segment nemá. Jako jediná by vyžadovala opakované čtení či zápis z adresy, tudíž rozšíření protokolu, který tuto funkcionalitu neumožňuje. Protože by vrstva byla užitečná pouze v případě, že se k senzoru připojí rozšiřující hardware, a tato možnost zatím nebyla uvažována. Proto segment definován není.

Implementačně je adresový prostor rozdělen do dvou úrovní. Vyšší úrovní je struktura `si_mmi` (Surround Imager Memory Mapped Interface), pro kterou jsou definovány funkce pro čtení a zápis dat a správu modulů. Funkce dle čísla segmentu vyberou segment, se kterým se má pracovat, podle adresy v segmentu vyberou příslušný modul a požadavek mu předají. Moduly jsou uloženy ve strukturách typu `si_mmi_module`, které obsahují ukazatele na kontext modulu, funkce čtení a zápisu definici jeho adresového prostoru. Část architektury služby je na obrázku 6.2.



Obrázek 6.2: Archiktektura služby řízení založená na modulech mapovaných do paměti. Zobrazeny jsou pouze části zodpovědné za přenos obrazu mezi frameworkem RSoC a uživatelským zařízením. Na příkaz uživatelského zařízení (přijat přes `tcp_comm`) modul `rsoc_reader` přečte obrazová data a uloží je do modulu `image_memory`. Z něj jsou přečtena uživatelskou aplikací. Mapování modulů do adresového prostoru zajišťuje modul `MMI` (Memory Mapped Interface).

Každému implementovanému modulu je pro jednoduchost a rychlost přístupu k paměti přiřazen jeden segment. Pokud jsou jako adresa segmentu použito 8 nejvýznamnějších bitů 32 b adresy, je k dispozici 256 segmentů o velikosti 16 MB. Taková velikost však nestačí pro uložení obrazových dat. Použití více modulů paměti v sousedních registrech je kvůli přechodům mezi nimi nepraktické. Zbývá tak redukovat počet segmentů. Použití 4 bitů pro adresu segmentu problém s velikostí vyřeší, zmenší však počet segmentů na 16, což je málo.

Řešením je rozdělení prostoru na dvě poloviny podle nejvyššího bitu adresy. Polovina s hodnotou 1 je rozdělena na 8 segmentů velikosti 128 MB. Ty jsou určeny na ukládání obrazových dat, komunikaci s EEPROM (pokud její velikost přesáhne 16 MB) atd. Spodní polovina adresového prostoru je rozdělena na 128 segmentů po 16 MB, což je dostatečné pro všechny ostatní účely (mapování registrů, ukládání konfigurace, atd.).

6.5.2 Modul preview

Zatímco režim capture je řízen uživatelským zařízením, modul preview musí být řízen ze senzoru. Pokud by nebyl, uživatelské zařízení by muselo neustále komunikovat se senzorem pro řízení detekce prstu. Řízení je realizováno v modulu preview.

Pro vlastní řízení jsou dvě možnosti. Může obsahovat dvojitý buffer a zapisovat do bufferu, který není čten. Aby však mohlo dojít k vystřídání bufferů, čtení musí vždy počkat na dokončení zápisu aktuálního snímku. Nejhorším případem je, že čtení bude o málo pomalejší než zápis, pak dojde ke snížení frekvence nových snímků na uživatelském zařízení na polovinu. Jedná se případ čtení jednoho snímku využitím plné rychlosti full-speed USB, kde odečtením režie od ideální rychlosti vychází právě rychlost o něco menší než čtení obrazových dat.

Výhodnější, a proto použitou, možností je použití tří bufferů pro každou kameru, jejichž přepínání je definováno dvěma pravidly:

1. Zápis nového snímku přepíše starší ze snímků, které nejsou čteny.
2. Čtení vybírá novější ze snímků, které nejsou zapisovány.

Díky pravidlům se budou při čtení jednoho snímku zbylé dva střídat pro zápis, a čekat na nový snímek může být potřeba pouze v případě, že čtení je rychlejší než zápis. Toho lze dosáhnout jen ve speciálních případech, proto čekání implementováno není a při příliš rychlém čtení bude snímek přečten vícekrát.

Řazení snímků v trojici bufferů je realizováno pomocí trojice čítačů. Ty by teoreticky mohly při použití 32,b šířky během několika měsíců přetéct, proto je použita šířka 64 b. Dalšími možnostmi by bylo například použití front nebo udržování čítačů v určitém malém rozsahu. To by však vedlo k lehce, nicméně zcela zbytečně komplikovanějšímu kódu.

Užitečnou funkcí by mohlo být sdružování čtení snímků. Stávající přístup (i stávající senzor) negarantuje, že při přečtení preview snímků ze všech kamer budou tyto snímky z jediné iterace režimu preview. Proto by modul preview mohl obsahovat funkci, která na začátku čtení snímku tento uzamkne pro čtení spolu se snímky zbylých dvou kamer. Po přečtení snímků všech kamer dojde k jejich odemknutí a při dalším čtení bude již zamknuta jiná trojice. Funkce nesmí být aktivována neustále, protože by nebylo možné např. číst pouze střední snímek.

Určitou optimalizací, která však přinesla zjednodušení návrhu, je odeslání dat všech snímků naráz při aktivovaném sdružování čtení snímků. Uživatelské zařízení si pak nemusí každý snímek vyžadovat zvlášť. To však není výraznou výhodou. Tou je zjednodušení v případě surového ukládání snímků, kdy proud dat z akcelerátoru není rozdělen do snímků podle hlaviček, ale pouze uložen do paměti senzoru. Modul preview by musel buď hlavičky sám parsovat, nebo místo modulu čtení adresového prostoru používat modul čtení paketů. Při aktivaci sdruženého čtení snímků se však modul čtení adresového prostoru použít může, data jsou do uživatelského zařízení přenesena včetně hlaviček a do snímků rozdělena až v ovladači na straně uživatelského zařízení.

6.5.3 Modul čtení adresového prostoru

Čtení obrazových dat uživatelskou aplikací na zajišťuje na straně zařízení modul čtení adresového prostoru. Tomu je předána adresa, na které se nachází vybraný snímek, velikost snímku, a následně aktivováno čtení. Čtená data jsou odeslána datovým kanálem. Modul k paměti přistupuje pouze pro čtení, proto adresy nejsou omezeny jen na segment snímků a modul lze použít i při ladění chyb pro kontrolu zapsaných nastavení. Adresový prostor lze v omezené míře číst i přímo kontrolním kanálem, což lze použít například pro získání výsledků funkcí pro zpracování obrazu.

Modul přesouvá obsah paměťového prostoru do datového kanálu, může proto běžet na pozadí a hlavní vlákno tak pokračuje ve zpracování řídicích zpráv od uživatelského zařízení. Hrozí riziko že modul přečte právě měněná data. Problém by se dal řešit mutexy, což by znamenalo složitou synchronizaci (pokud by se zamykaly oblasti paměti) nebo výrazné snížení rychlosti přenosu dat (pokud by se zamykala celá paměť). Při správném použití však čtení a zápis budou vždy probíhat v různých částech paměti a synchronizace není potřeba. Proto mutexy použity nejsou a služba spoléhá na správnou konfiguraci.

6.5.4 Modul čtení paketů

Kvůli problémům při implementaci modulu čtení akcelerátoru (a případnému zvýšení jeho výkonu, pokud by to bylo potřeba) byl vyvinut modul čtení paketů, který na čtené adrese očekává první paket přenosu a podle hlaviček vybírá data, která se mají číst pro přenos vybraného snímku. Režim dává smysl i v běžném provozu — při přesunu dat mezi akcelerátorem snímání a RAM je potřeba co nejvyšší výkon, proto se zpracování paketů může přesunout až na přenos dat mezi RAM a uživatelským zařízením, který je výrazně pomalejší.

Problém však může nastat s funkcemi pro zpracování obrazu, které by musely znát strukturu paketů. To by šlo vyřešit vystavením obrazových dat od adresového prostoru. Kvůli rychlému přístupu k pixelům vybraného snímku by modul čtení paketů musel vytvořit index paketů rozdělených dle snímku a seřazených dle části snímku, kterou obsahují.

6.5.5 Modul čtení akcelerátoru

Přenos snímků z akcelerátoru do paměti zajišťuje modul čtení akcelerátoru. Ten je nakonfigurován třemi adresami v segmentu snímků, do kterých po začátku snímání data zapíše. Adresy jsou konfigurovány nastavením fáze, proto modul kontroluje, zda odkazují do správného segmentu, a pokud ne, snímky nezapíše.

Obsahuje tři ukazatele do prostoru snímků. Po odstartování snímání jsou ukazatele nastaveny na cílové adresy. Při vlastním čtení modul do interního bufferu přečte hlavičku, ze které určí cílový snímek a počet slov čtených dat. Protože jsou data vždy zarovnána na celá slova, hlavička obsahuje délku případného dorovnání. Po dokončení čtení paketu modul posune ukazatel o daný počet bytů zpět.

Mechanizmus řízení čtení hlavičkami paketů lze i vypnout a pakety ukládat přímo do RAM, což mírně zvýší výkon přenosu. Problémem je však jeho ukončení, kdy není k dispozici ukončovací hlavička. Proto režim surového ukládání na nová data čeká maximálně 1 ms a pokud do té doby akcelerátor nová data nepošle, přenos je považován za ukončený.

Protože se čtení musí provádět na pozadí, modul obsahuje vlastní vlákno. Opět hrozí riziko, že modul čtení akcelerátoru zapíše do právě čtených dat, a opět je jeho řešení ponecháno na konfiguraci z uživatelského zařízení.

6.6 Služba aktualizace

Služba aktualizace slouží primárně pro příjem nových verzí služby řízení a firmware FPGA, jejich autentizaci a zápis do úložiště senzoru. Je spouštěna operačním systémem, pokud neběží služba řízení a existuje soubor `/tmp/start_si_update`. Po spuštění čeká na příkazy uživatelského zařízení. V základní verzi definuje čtyři příkazy: aktualizace služby řízení, aktualizace firmware FPGA, restart systému a ukončení procesu aktualizace.

Aktualizace procesu řízení a firmware FPGA probíhá podobně. Aktualizace je započata příkazem řídicího kanálu. Datovým kanálem je odeslán podepsaný soubor. Po úspěšném ověření podpisu je soubor uložen na disk místo původní verze. Novou službu řízení lze spustit ihned (příkaz ukončení procesu aktualizace), pro zavedení nového firmware FPGA je třeba restart. Pro okamžité zavedení by byl potřeba mechanizmus, který odstraní modul RSoC z jádra, deaktivuje propojení procesoru a FPGA, nahraje nový firmware, aktivuje propojení a znovu zavede modul. Kromě okamžitého zavedení by tento mechanizmus umožňoval online aktualizaci firwmare FPGA.

Jak bylo uvedeno v návrhu, podpis souborů s aktualizacemi komplikuje pokus o vzdálenou aktualizaci upraveným souborem, určeným například pro krádeže otisků a jejich odesílání třetí straně. Vzhledem k povaze asymetrické kryptografie je však nyní nutné řešit i vypršení platnosti veřejného klíče. První možností by bylo problém ignorovat a spolehnout se, že za dobu životnosti zařízení nedojde k prolomení klíče o zvolené délce. Délka může být výrazně vyšší než je použitelné v jiných aplikacích, jelikož se jedná o použití v jednorázovém procesu, který není časově kritický. Druhá možnost je mechanizmus aktualizace klíče, který by fungoval stejně jako aktualizace procesu řízení a firmware FPGA. Při vhodně zvolené délce klíče bude jednou či dvakrát během života zařízení potřeba klíč aktualizovat, jinak bude (při vypršení platnosti stávajícího klíče) vzdálená aktualizace vyřazena z provozu. Zkušenosti se starou verzí ukazují, že většinu problémů se senzorem lze za jistou cenu (např. občasné falešné odmítnutí při selhání snímání) řešit i z uživatelské aplikace. Takový scénář není tak špatný, jako senzor infikovaný škodlivým kódem.

Zvážena byla i možnost symetrické kryptografie a tajného klíče uloženého na zařízení. Scénáře útoků však počítají i se situací, kdy útočník získá senzor a je schopen přečíst jeho paměťové zařízení. V takovém případě by byl tajný klíč vyzrazen a zabezpečení pozbylo smyslu. Unikátní klíč v každém zařízení nepřipadá v úvahu kvůli režii spojené s vydáním speciálního aktualizačního souboru pro každý kus senzoru.

6.6.1 Online aktualizace

V praxi je snaha o co nejmenší zásahy do firmware procesoru. Obyčejně je jednodušší a bezpečnější aktualizovat software na uživatelském zařízení. Při aktualizaci senzoru může dojít k výpadku proudu v nevhodný okamžik, poškození souborového systému úložiště či jiné kritické chybě, která způsobí zablokování senzoru na straně zákazníka. Proto je navržen i mechanizmus online aktualizace, kdy nedochází k zápisu aktualizovaných souborů do úložiště senzoru, nýbrž pouze do jeho RAM. Výhodou takového přístupu je stabilita, protože nevadí kritická selhání — při chybě stačí senzor restartovat. Nevýhodou přístupu je rychlost, všechny aktualizační soubory se do senzoru musí nahrát při každé jeho inicializaci.

Vlastní mechanizmus může být součástí služby aktualizací a využívat okamžitého zavedení. Rozdíl oproti standardní aktualizaci je v úložišti (online aktualizace ukládá do RAM). K dispozici by měl být i mechanizmus, který aktualizaci ze senzoru smaže.

6.7 Funkce pro zpracování obrazu

Díky mikrokontroléru lze do senzoru vložit i některé funkce pro zpracování obrazu. Jak bylo uvedeno v návrhu prototypu, je potřeba zvážit, které funkce dává smysl v senzoru implementovat, a zatím jedinou takovou funkcí je detekce prstu.

6.7.1 Detekce prstu v senzoru

Detekce prstu v senzoru je převzata ze stávající verze zařízení. Praxe prokázala, že její velmi jednoduchý princip funguje dostatečně dobře ve všech situacích. Detekce se provádí periodicky. V každé periodě jsou sejmuty všechny tři snímky a z každého vybrán jeden sloupec pixelů, který se prahuje. Dle počtu pixelů s nadprahovou hodnotou je rozhodnuto, zda je dle daného snímku do senzoru vložen prst. Pouze pokud je prst detekován v každém snímku, je uživatelské zařízení upozorněno nastavením hodnoty registru výsledku.

Pro samotné zpracování obrazu lze nakonfigurovat akcelerátor — podporuje jak ořez obrazu až na jeden pixel šířky, tak prahování pomocí LUT. Úlohou procedury detekce je tak jen spočítat nenulové pixely a rozhodnutí uložit do registru, odkud jej přečte modul čtení adresového prostoru.

6.8 Ladění a monitorování

Ladění a monitorování senzoru lze rozdělit do dvou oblastí, krátkodobé a dlouhodobé. Krátkodobé sledování je použito při ladění známých chyb či testování při vývoji. Je použito připojení přes JTAG a k dispozici jsou všechny získatelné údaje. Tento přístup je však nevýhodný pro dlouhodobé sledování, například při snaze o replikaci vzácně se vyskytující chyby. V takovém případě je sledováno několik senzorů po dobu řádově dnů či týdnů. Proto senzor obsahuje ladicí modul s rozhraním dostupným přes UART. Při problémech v ostrém nasazení je navíc výhodné mít možnost získat informace i bez připojení k UART či JTAG, proto je rozhraní dostupné i přes adresový prostor.

Ačkoliv logování do úložiště senzoru by bylo nepraktické, to samé nelze říct o statickém kruhovém bufferu definovaném v paměti služby. Jakoukoliv chybu či podezřelé chování může kterýkoliv modul do bufferu zapsat a sledující aplikace přečíst. Kruhový buffer je možné jedním příkazem atomicky přečíst a vyprázdnit, aby nedocházelo k duplikacím zpráv ani jejich ztrátám.

V adresovém prostoru je rozhraní ladicího modulu a jeho kruhový buffer dostupný v segmentu služby a umožňuje čtení uživatelským zařízením nebo přes UART. Rozhraní definuje kromě příkazů pro práci s kruhovým bufferem i příkazy pro čtení adresového prostoru či jeho změny. Změny v adresovém prostoru umožňují větší možnosti při ladění přes UART, nicméně v ostrém provozu představují bezpečnostní riziko, proto tato funkcionalita není do ostrých verzí kompilována.

Kapitola 7

Realizace a testování

Pro ověření funkčnosti navrženého řešení není nutná implementace všech vlastností, postačí pouze kritické části, potřebné pro sejmutí trojice snímků, jejich uložení v paměti senzoru a přenos do uživatelského zařízení. V této kapitole je popsána realizace kritických částí senzoru, způsoby jejich testování a získané výsledky. Na závěr je nastíněn směr dalšího vývoje.

7.1 Realizace služby řízení

Ze služby řízení byly implementovány byly vrstva pro komunikaci s hardwarem, vrstva registrů zařízení a vrstva nastavení fáze. S uživatelským zařízením senzor komunikuje přes TCP/IP, které typicky dosahuje nižších rychlostí než USB, čímž se zvýší náročnost testování.

Návrh služby aktualizací bude muset být upraven dle výsledné realizace hardware. Proto byla služba aktualizací implementována pouze částečně, jako součást služby řízení, která umožňuje její aktualizaci. Jedná se opět o modul mapovaný do adresového prostoru senzoru, který obsahuje instrukce pro smazání aktualizačního souboru, přípis dat k aktualizačnímu souboru a ukončení služby s definovaným návratovým kódem. Jednoduchý skript, přes který je služba řízení spuštěna, pak při ukončení s daným návratovým kódem přepíše soubor služby aktualizací aktualizačním souborem. I přes jednoduchost přístupu lze tímto způsobem aktualizovat téměř všechny součásti senzoru, a aktualizace je tak bezpečná, jak bezpečný je souborový systém.

Při realizaci se ukázalo, že nebude potřeba implementace navrženého modulu čtení adresového prostoru. Ve všech případech, které nejsou kritické na časování (např. ladění), lze libovolný úsek paměti přečíst standardním komunikačním protokolem. Kritické je čtení obrazových dat, protože implementace protokolu vždy kopíruje čtená data z paměťového prostoru do interního bufferu. Za normálních okolností je kopie dat zanedbatelná, v případě paralelního čtení z akcelerátoru do paměti a z paměti do uživatelského rozhraní by však mohlo dojít k zahlcení sběrnic. Modul čtení adresového prostoru měl data číst přímo z modulů pamětí senzoru. Alternativou se však ukazují funkce pro čtení, které by v modulech, pro které to dává smysl, místo kopie dat pouze vrátily ukazatel do interní paměti. Z té pak mohou být data odeslána přímo.

7.2 Testování služby řízení

Při testování služby řízení bylo potřeba ověřit hlavně správnou funkčnost a rychlost protokolu komunikace s uživatelským zařízením. Ideální je spouštění služby přímo na vývojářském PC, které však neobsahuje FPGA. Proto pro počáteční testování služba ve výchozím stavu nekomunikuje s FPGA, a komunikaci je třeba při spouštění aktivovat parametrem.

Čtení obrazových dat snímků paket po paketu (přečtení hlavičky, přečtení dat, opakování dokud není neprázdná hlavička) bylo extrémně pomalé, přenos trojice snímků trval přes 30 s. Proto byla uživatelská aplikace upravena pro přečtení bloku paměti, ve kterém se pakety nachází, a rekonstrukci snímků na straně uživatelského zařízení. To snížilo dobu přenosu na cca 120 ms. Měření doby přenosu a její srovnání s ostatními částmi snímání je na obrázku 7.2.

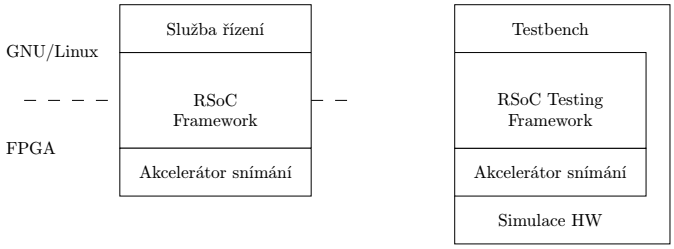
Při testování byla ověřena správná funkce protokolu komunikace, čtení a zápisu registrů a modulu aktualizace. Komunikace s FPGA musela být testována na kitu, což za normálních okolností znamená vysokou režii při vývoji, díky funkčnímu procesu aktualizace se však nejednalo problém.

7.3 Realizace akcelerátoru snímání

Z akcelerátoru snímání byl jako nejkritičtější část implementován a otestován přenos obrazových dat z kamer do paměti senzoru. Jedná se o synchronizaci mezi hodinovými doménami kamer a akcelerátoru, mezipaměti obrazových dat a komponentu pro generování paketů.

7.4 Testování akcelerátoru na simulátoru

Pro testování akcelerátoru byl použit testovací projekt frameworku RSoC pro simulační prostředí Altera Modelsim. Namísto kamer byly použity generátory dat nastavitelné pro různé velikosti snímků. Účelem testován bylo ověřit výkon přenosu, konzistenci přenášených dat a správné tvoření paketů. Achitektura testovacího frameworku je na obrázku 7.1.



Obrázek 7.1: Srovnání architektury reálného provozu (vlevo) a testování. Akcelerátor může být testován beze změn, je však potřeba implementovat jak testbench, tak simulaci hardwarových komponent.

Pro testování generátoru paketů byla vytvořena komponenta, která kontroluje jejich konzistenci. Pracuje vždy když je `TVALID` a `TREADY` zároveň aktivní a dochází tak k přenosu dat. V cyklu přečte hlavičku a odpočítá příslušný počet slov. Pokud je hlavička větší než limit nastavený v modulu, permanentně nastaví příznak nekonzistentního paketu. To umožní

jak notifikaci instance SignalTap, tak jednoduché vyhledání problematického místa ve výsledcích simulace. Provedené simulace pak pak prokázaly jak konzistenci dat, tak správnost tvoření paketů.

Simulátor měl ověřit i výkon akcelerátoru. Pokud by režie vytváření paketů a jiných částí akcelerátoru byla příliš vysoká, došlo by k přetečení vyrovnávacích pamětí akcelerátoru. Při simulaci však k tomuto případu nedocházelo.

7.5 Testování akcelerátoru na reálném hardware

Pro otestování v reálném provozu byl použit vývojový kit SoCKit a rozšiřující deska pro GPIO. Byl zhotoven adaptér pro kameru, který umožňuje její zapojení do rozšiřující desky. V průběhu vývoje vyšlo najevo, že na rozšiřující desku nebude možné připojit všechny tři kamery, proto je použita jedna kamera pro reálný obraz a její výstup kopírován na vstupy pro zbývající dvě kamery. Účelem testování pak bylo hlavně ověřit rychlost a stabilitu přenosu obrazových dat mezi FPGA a RAM.

Testování proběhlo ve třech fázích. V první byl akcelerátor do zařízení vložen bez frameworku RSoC, připojený na komponentu, která kontrolovala konzistenci paketů. Cílem bylo ověřit, jak bude akcelerátor fungovat na reálném hardware. Ukázalo se, že akcelerátor funguje správně. Zachycením odesílaných dat instancí SignalTap a jejich ručním extrahováním do formátu PGM byla získána i část obrazu.

Poté byl akcelerátor připojen na framework RSoC a spuštěna služba řízení senzoru. Po odladění počátečních problémů pak bylo možné na příkaz z uživatelského zařízení uložit snímek z kamery do paměti senzoru a následně ho přečíst.

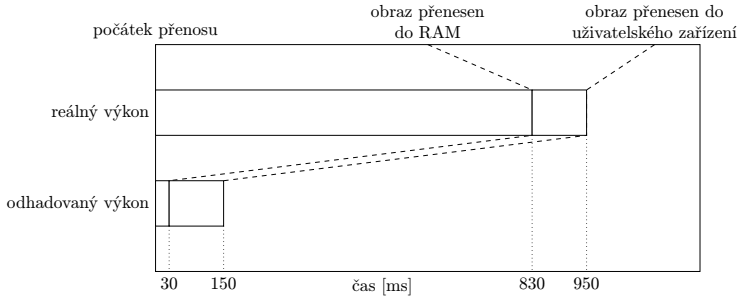
Poslední fází bylo testování výkonu přenosu. Ve službě řízení bylo implementováno měření doby od dokončení přenosu první dávky dat až po přenos celého obrazu. Tím se eliminovala doba mezi spuštěním snímání a začátkem přenosu. Čas strávený v jiných úlohách eliminován nebyl, služba řízení je jedinou úlohou běžící na zařízení (mimo systémové) a i výsledky měření se ukázaly jako konstantní. Výsledky měření pak byly ukládány do registru přístupného z paměťového prostoru.

Při vlastním testování se objevil vážný problém s výkonem. Přenos dat z FPGA do paměti RAM místo očekávaných cca 90 ms trval cca 850 ms. Tak dlouhá doba přenosu však bez ztráty nebyla možná — při datovém toku kamer 50 MB/s by téměř celý obraz musel být uložen někde mezi akcelerátorem a službou řízení.

S použitím externího logického analyzátoru vyšel najevo problém s hodinovým taktem generovaným z HPS. Ukázalo se, že ačkoliv je požadovaná frekvence nastavená na 50 MHz, reálná frekvence jsou necelé 2 MHz. Datový tok z kamer tak byl mnohem nižší, než by měl být. Tento problém se zatím nepodařilo vyřešit, protože při zvýšení frekvence jiným způsobem (např. PLL) dojde k nesplnění omezení časování na straně frameworku RSoC a je získán silně porušený obraz. Dopad tohoto problému na výkon snímání je na obrázku 7.2.

7.6 Testování USB

Posledním krokem ověření vhodnosti platformy bylo otestování funkce USB. Pokud by hardware v čipu nebyl podporován operačním systémem, znamenalo by to přepracování celého návrhu. Jelikož byl prototyp postaven nad sběrnicí Ethernet, testování USB muselo být provedeno zvlášť.



Obrázek 7.2: Srovnání reálného výkonu přenosu trojice snímků ze senzoru do uživatelského zařízení a odhadovaného výkonu téhož přenosu po vyřešení problému s hodinami. Omezujícím faktorem výkonu je právě frekvence hodin kamer, proto byl výkon po opravení odhadnut z poměru požadované a reálné frekvence.

Pro testování byl použit modul Linuxu `g_mass_storage`, který implementuje funkci velkokapacitního úložiště. Při zavedení do jádra mu je jako parametr předán soubor, jehož obsah je zviditelněn uživatelskému zařízení. Toho bylo využito v raných fázích testování akcelerátoru snímání, kdy neexistoval pohodlnější způsob přenosu obrazu mezi senzorem a PC. Sejmutá data byla uložena do souboru, ten byl použit jako úložiště a po připojení k vývojovému PC přečten programem `dd`.

Tímto způsobem byla ověřena funkčnost USB. Programem `dd`, byla zároveň změřena i dosažitelná (ne nutně maximální dosažitelná) rychlost přenosu dat mezi endpointy typu bulk. Ta činí necelých 12 MB/s. Ve výsledném senzoru bude rychlost vyšší, protože nebude zatížena režií spojenou s třídou USB. Dále bude ovlivněna čipem použitým pro fyzickou vrstvu USB. Požadavek na dostatečně výkonnou komponentu tedy bude muset být zahrnut do návrhu desky.

7.7 Další vývoj

V rámci dalšího vývoje bude potřeba dokončit nekritické části akcelerátoru snímání a, ideálně na kitu s cílovým čipem (např. Novpek CVLite), ověřit dostupnost potřebného množství zdrojů a možnost splnění omezení časování. Poté bude možné přejít k návrhu vlastní desky a implementaci zbylých částí služby řízení.

Framework RSoC mimo jednoduchého rozhraní v podobě souboru pro čtení, zápis a mapování do paměti obsahuje ještě další rozhraní na nižší úrovni, která by mohla umožnit čtení více proudů dat jiným způsobem než pomocí paketů. Generátor paketů by se tak stal zbytečným a ušetřily by se zdroje FPGA.

Bude potřeba rozhodnout, zda implementovat navrženou službu aktualizace, nebo aktualizaci ponechat ve zjednodušené podobě, v jaké byla realizována. Rozhodnutí bude záviset zejména na způsobu uložení kořenového souborového systému. Pokud bude uložen jako soubor na oddílu booloaderu, zjednodušená podoba bude vůči výpadkům odolná tak, jako je odolný souborový systém na tomto oddílu. Pokud bude kořenový souborový systém uložen na klasickém oddílu, aktualizaci musí provádět zavaděč a bude potřeba navržená varianta.

I přes optimalizace čtení z adresového prostoru jsou odesílaná obrazová data kopírována z uživatelského prostoru do prostoru jádra. Bude muset být provedena profilová analýza, která vyloučí zahlcení sběrnic při paralelním čtení z akcelerátoru do paměti a z paměti do uživatelského zařízení. Pokud k zahlcování docházet bude, bude potřeba najít způsob, jak se tomuto zahlcení vyhnout.

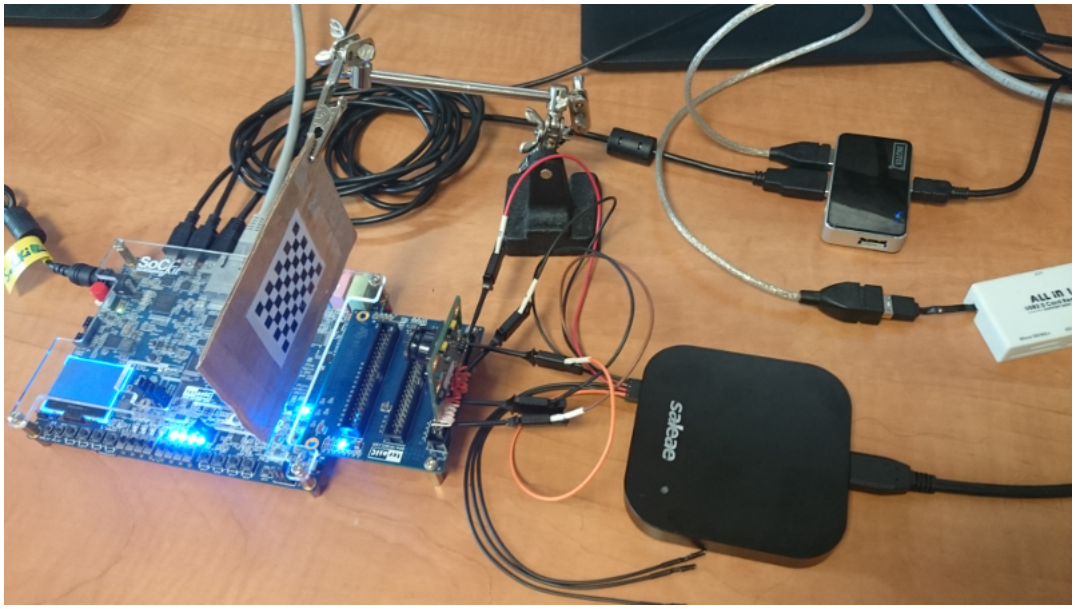
Kapitola 8

Závěr

Tato práce obsahuje popis existující platformy pro snímání biometrických vlastností lidského prstu, jejích výhod, nevýhod a prostředků pro její vývoj. Na základě těchto poznatků bylo navrženo několik možností pro vylepšení této platformy a z nich vybrána varianta postavená na kombinaci FPGA a mikrokontroléru ARM na zařízení Cyclone V. Byl navržen jak firmware pro FPGA, tak systém pro ARM v podobě programu pro systém GNU/Linux. Pro ověření konceptu byly na vývojovém kitu Arrow SoCKit implementovány kritické části senzoru a poté byla testována jejich funkčnost. Při vývoji bylo použito frameworku RSoC, který vznikl jako diplomová práce [12].

Výsledkem práce je implementace základních funkcí senzoru a jeho rozhraní v podobě modulů mapovaných do adresového prostoru. Dále byl navržen protokol pro komunikaci mezi senzorem a uživatelským zařízením, který lze použít jak pro připojení přes USB, tak pro připojení přes TCP/IP. Tento protokol byl implementován ve vrstvě firmware senzoru i knihovně pro uživatelská zařízení.

Navržený koncept může být použit jak pro novou generaci senzoru, tak pro jiné projekty, které vyžadují přesnou časovou synchronizaci osvětlení objektu, jeho snímání více kamerami a uchování snímků v senzoru, např. stereovizi. Kameru umístěnou na vývojovém kitu lze navíc použít pro experimenty s kamerou jako takovou, případně testování jejích optických vlastností. Fotografie zhotovené verze zařízení je na obrázku 8.1.



Obrázek 8.1: Fotografie výsledného zařízení, v tomto případě použitého v rámci jiného projektu pro kalibraci kamery a určení jejího zorného pole.

Literatura

[1] AMBA 4 AXI4-Stream Protocol Specification [online]. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0051a/index.html, 2010 [cit. 2015-02-13].

[2] libusb-1.0 API Reference [online]. http://libusb.org/static/api-1.0/, 2015-02-03 [cit. 2015-03-13].

[3] The Buildroot user manual [online]. http://buildroot.uclibc.org/downloads/manual/manual.html, 2015-03-01 [cit. 2015-01-24].

[4] Altera Corporation: Design Debugging Using the SignalTap II Logic Analyzer [online]. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/qts/qts_qii53009.pdf, 2013 [cit. 2015-04-28].

[5] Altera Corporation: SCFIFO and DCFIFO IP Cores User Guide [online]. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_fifo.pdf, 2015 [cit. 2015-04-28].

[6] Aptina Imaging Corporation: 1/2-Inch Megapixel CMOS Digital Image Sensor Features [online]. https://www.aptina.com/assets/downloadDocument.do?id=851, 2004 [cit. 2015-03-15].

[7] Aptina Imaging Corporation: MT9M001 Register Reference [online]. https://www.aptina.com/assets/downloadDocument.do?id=863, 2004 [cit. 2015-03-15].

[8] Drahanský, M.; Orság, F.; kolektiv: *Biometrie*. Computer Press, první vydání, 2011, ISBN 9788025489796.

[9] Inc., T.: SoCKit Specification [online]. http://sockit_support.terasic.com, 2013 [cit. 2014-12-10].

[10] Peacock, C.: USB in a NutShell [online]. http://www.beyondlogic.org/usbnutshell/usb1.shtml, 2010 [cit. 2015-04-02].

[11] RehiveTech: RSoC Description [online]. 2015 [cit. 2015-05-04], http://rsoc-framework.com/description/.

[12] Viktorín, J.: *HW/SW Codesign for the Xilinx Zynq Platform*. Diplomová práce, Brno, FIT VUT v Brně, 2013.

[13] Yaghmour, K.; Masters, J.; Ben, G.: *Building Embedded Linux Systems, 2Nd Edition*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., druhé vydání, 2008, ISBN 9780596529680.