VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

IMPLEMENTACE JEDNOTKY PRO OBSLUHU BOOTOVÁNÍ INTEL FPGA

IMPLEMENTATION OF A BOOT CONTROLLER FOR INTEL FPGAS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ HAK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MATOUŠEK JIŘÍ, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Hak Tomáš**
Program: Informační technologie
Název: **Implementace jednotky pro obsluhu bootování Intel FPGA**
Implementation of a Boot Controller for Intel FPGAs
Kategorie: Návrh číslicových systémů

Zadání:

1. Seznamte se s moderními FPGA čipy Intel Stratix 10 a Agilex, zejména si nastudujte způsoby jejich konfigurace.
2. Navrhněte jednotku pro FPGA čip umožňující komunikaci s konfigurační pamětí FPGA čipu a nahrání nového bitstreamu.
3. Navrhněte uživatelskou aplikaci pro práci s navrženou jednotkou v systému Linux.
4. Navrženou jednotku implementujte v jazyce VHDL; navržený software implementujte v jazyce C.
5. S využitím implementované jednotky a vytvořeného software ověřte schopnost aktualizace bitstreamu.
6. Zhodnoťte dosažené výsledky a diskutujte možnosti dalších rozšíření.

Literatura:

- Podle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Matoušek Jiří, Ing., Ph.D.**
Konzultant: Cabal Jakub, Ing., CESNET
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 29. října 2021

Abstrakt

Tato práce se dotýká využití technologie FPGA v oblasti počítačových sítí, konkrétně pro hardwarovou akceleraci zpracování síťového provozu na síťové kartě vyvíjené sdružením CESNET, z. s. p. o. Technologie FPGA je oblíbená zejména díky možnosti snadno rekonfigurovat čip a opravit tak případné chyby či aktualizovat firmware. Práce nejprve pojednává o návrhu a implementaci nové jednotky pro Intel FPGA, která bude schopná komunikovat s externí konfigurační flash pamětí čipu osazeného na výše zmiňované kartě. Dále pak řeší návrh a implementaci softwarového nástroje, který bude umožňovat skrze nově implementovanou firmwarovou jednotku nahrát do flash paměti nová konfigurační data a vynutit si rekonfigurování FPGA čipu pomocí těchto nově nahraných dat. Ke konci práce je funkcionalita nově implementovaného systému otestována v praxi.

Abstract

This thesis touches the topic of using FPGA technology in the field of computer networks, specifically for hardware acceleration of network traffic processing on a network card developed by the CESNET association. FPGA technology is popular mainly due to the possibility to easily reconfigure the chip and fix any errors or update the firmware. The thesis first discusses the design and implementation of a new unit for Intel FPGA, which will be able to communicate with the external configuration flash memory of the chip featured on the card mentioned above. It then goes on to address the design and implementation of a software tool that will allow, via the newly implemented firmware unit, to load new configuration data into the flash memory and force reconfiguration of the FPGA chip using this newly loaded data. Towards the end of the thesis, the functionality of the newly implemented system is tested in practice.

Klíčová slova

FPGA, konfigurace, Intel, Stratix 10, Agilex, Active Serial, QSPI, CESNET, bitstream, SDM, SDM Client, RSU, Mailbox

Keywords

FPGA, configuration, Intel, Stratix 10, Agilex, Active Serial, QSPI, CESNET, bitstream, SDM, SDM Client, RSU, Mailbox

Citace

HAK, Tomáš. *Implementace jednotky pro obsluhu bootování Intel FPGA*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Matoušek Jiří, Ph.D.

Implementace jednotky pro obsluhu bootování Intel FPGA

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doktora Jiřího Matouška. Další informace mi v roli konzultanta poskytl inženýr Jakub Cabal ze sdružení CESNET. V seznamu literatury jsem uvedl všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Hak
9. května 2022

Poděkování

Moje poděkování patří oběma výše zmiňovaným, doktoru J. Matouškovi za mé zapojení do výzkumu v rámci sdružení CESNET, odborné vedení této práce a pomoc při vypracování její textové podoby a inženýru J. Cabalovi za rady a průběžné konzultace při vypracovávání její praktické části a pomoc při dokončování její finální verze.

Obsah

Kapitola 1

Úvod

Zejména díky Internetu se prosazuje přenos čím dál většího množství dat v elektronické podobě. Aby bylo možné všechna tato data beze ztrát a co nejrychleji zpracovávat a aby byla zajištěna dostatečná bezpečnost, vybraná zařízení v síti jsou vybavena technologiemi, které umožňují celou komunikaci hardwarově akcelerovat. Jednou z hojně používaných technologií pro hardwarovou akceleraci jsou čipy FPGA (Field-Programmable Gate Array).

Největší výhodou čipů FPGA je zejména možnost jejich snadné rekonfigurace a aktualizace firmware. Díky této vlastnosti se staly oblíbenými v širokém spektru aplikací. Konfigurace probíhá při procesu bootování čipu, resp. samotné síťové karty, na níž je osazen, a jeho výsledkem je nahrání konfigurace do interních SRAM pamětí na čipu. Konfigurační data jsou u různých čipů uložena různými způsoby. Některé čipy mají přímo v sobě konfigurační paměť, do jiných (což je i případ této práce) jsou konfigurační data nahrávána z externí flash paměti. Výsledkem konfigurace je, že čip vykonává činnost podle designu, který do něho byl nahrán. Tato práce je z většiny zaměřena na jeden konkrétní způsob konfigurace (Active Serial, který pro svou činnost používá právě externí konfigurační QSPI flash paměť).

Tato bakalářská práce svým obsahem spadá do oblasti využití těchto čipů při zpracovávání síťového provozu na akcelerační síťové kartě. Kromě samotné hardwarové akcelerace zpracování dat je na této úrovni také možné data monitorovat či filtrovat škodlivý síťový provoz. Mimo tuto oblast nachází čipy FPGA své uplatnění také např. u vestavěných zařízení nebo pro potřebu rychlého prototypování hardwarových aplikací.

Sdružení CESNET se ve spolupráci s některými českými univerzitami věnuje mj. vývoji akceleračních síťových karet, které na sobě mají osazen FPGA čip. Takovýchto karet již v rámci projektu Liberouter vyvinulo několik. K dnešnímu datu poslední taková karta nese interní označení 400GE a podporuje propustnost až kolem 400Gb/s. Tato karta je osazena čipem Intel Agilex a je jednou z cílových platforem, pro kterou jsou vyvíjeny nástroje diskutované v této práci.

Přínosem tohoto řešení bude možnost vzdáleně konfigurovat čip na kartě za použití vlastního softwarového nástroje, a to zápisem konfiguračního obrazu do konfigurační flash paměti a následného vynucení rekonfigurace z právě zapsaného obrazu. Takto je možné uchovávat v závislosti na kapacitě konfigurační paměti jeden tovární obraz a několik aplikačních obrazů současně vedle sebe a přepínat mezi nimi podle potřeby. Tento přístup umožňuje distribuci vylepšení a opravu chyb v designu bez nutnosti stahování produktu od klienta, rychlejší nasazení produktu na trh a obecně prodlužuje životnost produktu [9, 10].

Pro možnost takto konfigurovat daný čip je potřeba vytvořit pro něj design s hardwarovou jednotkou, která bude schopná komunikovat s vestavěným Hard IP[1] čipu, který je předpřipraven výrobcem a samotnou konfiguraci ovládá. Na druhé straně bude tato jednotka komunikovat se softwarovou aplikaci, která se postará o samotné nahrání nové konfigurace ze souboru skrze tuto jednotku do externí konfigurační paměti a vynutí rekonfiguraci čipu z daného obrazu uloženého v dané konfigurační paměti.

Veškeré teoretické souvislosti včetně pojednání o čipech od firmy Intel a jejich konfiguraci jsou uvedeny v kapitole 2. Kapitola 3 pojednává o hardwarové části této práce, jejíž výstupem je vytvoření návrhu a implementace nové jednotky pro FPGA, která umožňuje komunikaci s konfigurační flash pamětí a nahrání nového bitstreamu. Návrh a implementace softwarového nástroje a přidání podpory pro práci s navrženou jednotkou z kapitoly 3 je předmětem kapitoly 4. Kapitola 5 se zabývá ověřením funkcionality nově implementovaného systému. Závěrečná kapitola 6 zhodnocuje dosažené výsledky a jsou v ní uvedeny návrhy na další rozšíření.

[1]IP (z angl. "Intellectual Property") je logický blok či kus kódu poskytovaný výrobcem jako součást produktu

Kapitola 2

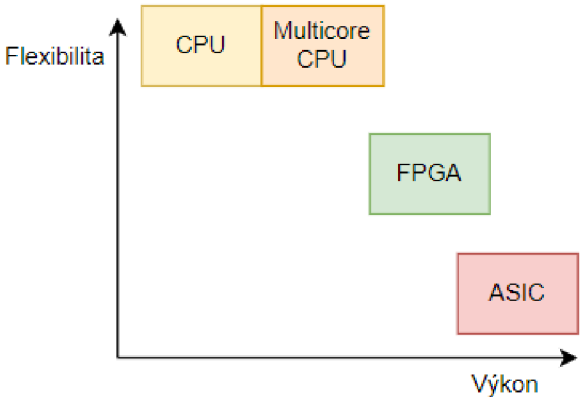
Teoretický přehled

2.1 Technologie pro hardwarovou akceleraci

Jak již bylo zmíněno v úvodu této práce, technologií umožňující paralelizování výpočtů na úrovni hardware, která je použita u nové 400GE karty, je FPGA z rodiny Agilex od firmy Intel. Mluvíme-li o technologii FPGA (z angl. "field-programmable gate array"), máme na mysli integrovaný obvod (nebo také čip), který je výrobcem navržen tak, aby byl naprogramován až koncovým uživatelem, jemuž nabízí možnost přeprogramování čipu pouhým nahráním nové konfigurace do konfigurační paměti v podobě binárních dat (tzv. bitstreamu). Princip činnosti FPGA je blíže popsán v sekci 2.2. Další technologií hojně používanou pro účel hardwarové akcelerace je ASIC (z angl. "application-specific integrated circuit"). U technologie ASIC je funkcionalita čipu dána už z výroby a důraz je kladen na maximální využití zdrojů a tedy maximální rychlost výpočtů. Oproti technologii ASIC nabízí FPGA jistou míru flexibility při vývoji aplikací, neboť u FPGA není potřeba měnit fyzické rozložení komponent na čipu pro změnu funkcionality čipu, ale stačí do něho pouze nahrát nový bitstream. Co se týče ceny za jednotku, jsou FPGA sice dražší než ASIC, ale případné chyby v designu jsou oproti technologii ASIC snadno opravitelné (nahrání nového bitstreamu). Technologie ASIC se vyplatí, pokud je vyráběna ve velkém množství, protože má velmi vysoké náklady na návrh masky čipu a s tím spojenou verifikaci, avšak pokud jsou případné chyby v návrhu objeveny příliš pozdě, znamená to nemalé ztráty. FPGA tedy představuje kompromis mezi flexibilitou a výkonem (viz obrázek 2.1) a také návrh aplikace je pomocí této technologie mnohem jednodušší. Díky tomu nachází hojné uplatnění u vestavěných a jiných zařízení, kde se během času mohou měnit požadavky na funkcionalitu zařízení, dále u menších projektů využívajících paralelní výpočty v hardware, kde se nevyplatí sériová výroba ASIC čipů, a obecně při vytváření prototypů aplikací, kdy je důraz kladen na rychlejší vývoj, průběžné testování a ladění a rychlejší uvedení produktu na trh.

2.2 Architektura FPGA

Hlavní součástí FPGA čipu je matice programovatelných logických bloků (dále jen PLB) a konfigurovatelná síť propojů, která tyto bloky navzájem spojuje. Dále se zde nachází věstavěné DSP (Digital Signal Processing) bloky pro efektivnější provádění některých operací (násobičky, práce s floating point čísly), blokové paměti a často také speciální obvody pro rychlou sériovou komunikaci (transcievery). Součástí PLB je i tzv. funkční generátor, který může pracovat buď jako posuvný registr, RAM paměť nebo tzv. LUT (z angl. "Look-Up



Obrázek 2.1: Poměr výkon-flexibilita u vybraných technologií

Table"). Právě díky možnosti konfigurovat tyto generátory jako LUT se výpočty v FPGA (oproti klasické hardwarové realizaci obvodů) několikanásobně urychlí. LUT má několik vstupů a právě jeden výstup. Při běžné hardwarové realizaci logické funkce by bylo potřeba z hradel sestavit obvod, který by tuto funkci realizoval. Naopak LUT funguje jako paměť, do které jsou „předpočítány“ výsledky této logické funkce pro všechny vstupy, takže mizí zpoždění jednotlivých hradel a obvod může díky tomu fungovat na vyšších frekvencích. Podle kombinace vstupů je pak jako výstup vybrán odpovídající řádek tabulky, který je svou hodnotou ekvivalentní výsledku dané logické funkce pro poskytnuté vstupy. LUT se dají navzájem propojovat a skládat do větších celků pomocí multiplexorů, aby bylo možné realizovat funkce o více vstupech. Pro účely této práce se v této sekci jedná pouze o stručné nastínění celé problematiky návrhu FPGA, více na toto téma může být nalezeno např. na `https://vyvoj.hw.cz/zaklady-fpga-co-je-fpga-a-proc-je-pouzit.html`.

2.3 Intel FPGA

Mezi největší výrobce čipů FPGA celosvětově patří firmy Xilinx, nedávno koupená firmou AMD[1], a Intel. Tato sekce je zaměřená na stručné seznámení s čipy od firmy Intel (Stratix 10 a Agilex), zejména pak se způsoby jejich konfigurace, které se od sebe u obou čipů výrazně neliší.

Architektura obou čipů je navržena podobně. U obou čipů je hlavním prvkem v centru umístěné FPGA, které je externě připojeno k několika dalším technologiím voleným podle potřeb zákazníka. Toto externí propojení je realizováno skrze technologii EMIB (z angl. "Embedded Multi-Die Interconnect Bridge"). Mezi připojované technologie patří různé paměti (HBM, DDR), transcievery či PCIe.

Intel Stratix 10 je starší a staví na 14 nm technologii, zatímco novější Intel Agilex je postaven na 10 nm technologii. Agilex dále také poskytuje možnost připojení např. DDR5 pamětí, PCIe 5.0 nebo 112G transcieverů. Oproti předchozí technologii nabízí o 40% vyšší výkonnost, o 40% nižší spotřebu nebo až 40 TFLOPS, co se týče výkonnosti DSP bloků. Pro více informací o čipu Intel Agilex viz např. článek na adrese `https://www.anandtech.com/show/14149/intel-agilex-10nm-fpgas-with-pcie-50-ddr5-and-cxl`.

[1] tisková zpráva je k dispozici na adrese `https://www.amd.com/en/press-releases/2022-02-14-amd-completes-acquisition-xilinx`

2.3.1 Konfigurace

Konfigurace FPGA čipů probíhá při zapnutí nahráním firmware z externího zdroje do interních konfiguračních pamětí v FPGA. Externím zdrojem pro konfiguraci bývá zpravidla konfigurační flash paměť. Obě výše jmenovaná zařízení v sobě zahrnují tzv. Secure Device Manager (dále jen SDM), jenž je podrobněji rozebrán v sekci 2.3.2, který se stará o konfiguraci FPGA a o jeho zabezpečení. Samotná konfigurace těchto zařízení pak může probíhat podle jednoho ze dvou schémat, která se od sebe navzájem liší rolí, ve které FPGA vystupuje. Jedná-li se o aktivní schéma, FPGA vystupuje jako master a externí pamět jako slave zařízení[2]. U pasivních schémat je proces řízen z vnějšku externím zařízením a FPGA vystupuje jako slave. Dále obě zařízení podporují částečnou rekonfiguraci[3] a u aktivního schématu konfigurace také Remote System Update (RSU), který je detailně popsán v sekci 3.4. Jaké schéma bude při konfiguraci zařízení použito určuje vektor MSEL. Příklad toho, jak se MSEL nastavuje, je uveden v sekci 3.2.

Aktivní schémata

- *Active Serial* (AS), v jednom ze dvou módů `normal` (MSEL = 011) nebo `fast` (MSEL = 001), je režim, při němž se SDM nakonfiguruje ze své boot ROM a pro konfiguraci zbytku zařízení se používá externí QSPI flash paměť (QSPI rozhraní je popsáno v sekci 2.4), kde je tato konfigurace uložená. Režim `fast` se liší od prvního zmiňovaného režimu pouze v tom, že se na počátku konfigurace nečeká 10ms. Tato práce se zabývá převážně tímto konfiguračním schématem v režimu `normal`.

Pasivní schémata (bližší informace jsou k nalezení v [9, 10])

- *Avalon Streaming* (Avalon-ST) je nejrychlejší konfigurační schéma. Podporuje 3 režimy – x8 (MSEL = 110) s využitím SDM I/O pinů, dále x16 (MSEL = 101) a x32 (MSEL = 000) s využitím GPIO pinů.
- *JTAG* využívá dedikované JTAG piny. Kromě konfigurace nabízí možnost využití debugování pomocí systémové konzole nebo pomocí nástroje Signal Tap pro sledování průběhů signálů. JTAG port má nejvyšší prioritu, pokud je povolený a zapojený, ignoruje se nastavení MSEL (i pokud není nastaveno jako JTAG, jinak pro JTAG je MSEL = 111).
- *Configuration via Protocol* (CVP) je konfigurace přes PCIe linku. Na začátku používá externí QSPI flash pamět v režimu Active Serial pro zprovoznění CvP rozhraní nahráním designu obsahujícího PCIe IP, přes které následně externí host konfiguruje zbytek zařízení.

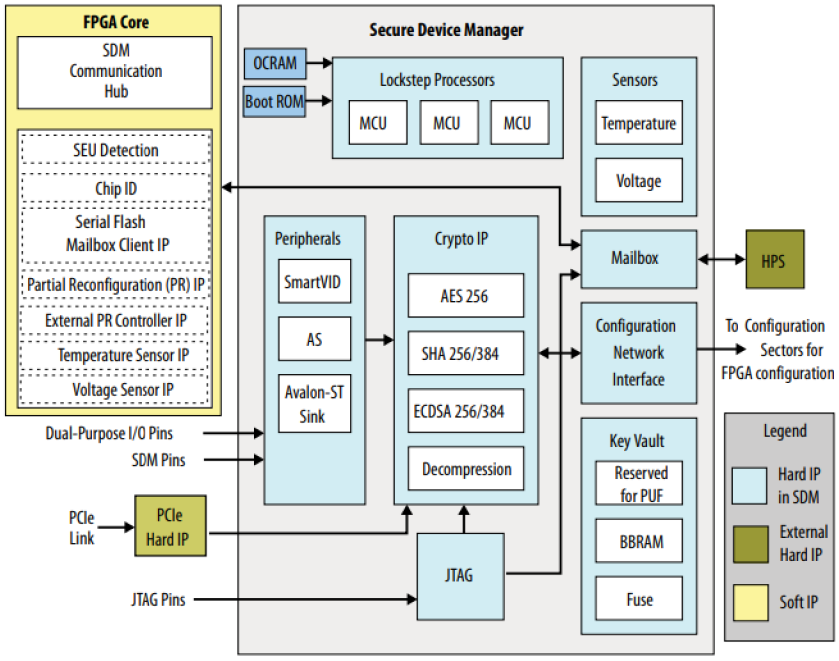
2.3.2 Secure Device Manager (SDM)

Secure Device Manager (viz obrázek 2.2) je modul na bázi procesoru zodpovědný za konfiguraci FPGA a autentizaci konfiguračních bitstreamů. Pro distribuci bitstreamu a konfiguraci zařízení využívá konfigurační síť složenou z tzv. Local Sector Managers (LSMs), kdy každý

[2] V modelu master-slave jedno zařízení či proces (master) řídí činnost druhého (nebo více) zařízení nebo procesu (slave).

[3] Částečná rekonfigurace umožňuje, aby byla část zařízení konfigurována, zatímco zbytek zařízení funguje v běžném režimu.

LSM je mikroprocesor zpracovávající jemu určenou část bitstreamu, pomocí níž pak komponenty ve svém sektoru konfiguruje. Tato konfigurační síť není z vnějšku přístupná. Kromě rozhraní konfigurační sítě, periferií pro jednotlivé režimy konfigurace a modulů pro kryptografii a zabezpečení v sobě SDM zahrnuje řadu dalších součástí. Mezi tyto součásti patří např. teplotní a napětový senzor, boot ROM, obvody pro RSU nebo Mailbox, přes který se dá za použití Mailbox Client Intel FPGA IP s SDM komunikovat (popis a použití tohoto IP je k nalezení v sekci 3.3.2).

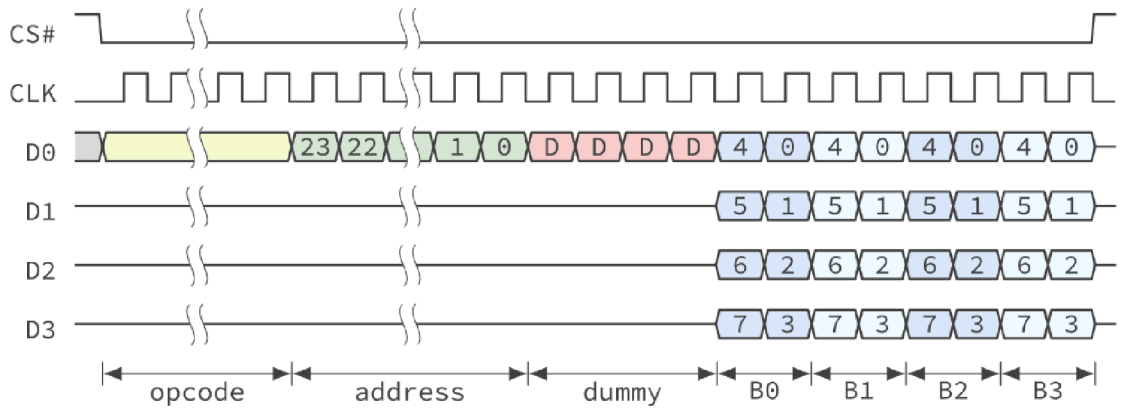


Obrázek 2.2: komponenta Secure Device Manager [9, 10]

2.4 SPI a Multi I/O SPI

SPI (z angl. "serial peripheral interface") je synchronní rozhraní pro sériovou komunikaci mezi zařízeními na principu master-slave. Komunikace na tomto rozhraní je plně duplexní[4]. Pro komunikaci používá signály SCLK (clock), MISO (master in/slave out), MOSI (master out/slave in) a SS (slave select), kde poslední jmenovaný může být pro účely komunikace s více zařízeními zastoupen vícekrát. Pro zvýšení propustnosti rozhraní jednoho ze zařízení se používají rozšířené varianty SPI, konkrétně Dual SPI nebo Quad SPI. U těchto se ale jedná pouze o komunikaci typu half-duplex[5]. Místo signálů MISO a MOSI jsou zde obecné signály IO, a to dva u Dual SPI a čtyři u Quad SPI. Propustnost se tedy při jejich použiti zvýší 2x, resp. 4x. Multi I/O SPI naleznou využití obzvláště tam, kde je potřeba intenzivně komunikovat s pamětí. Oproti dřívějšímu používání paralelních rozhraní (8-32bitových) nabízí možnost redukovat velikost externích pamětí díky zmenšenému počtu pinů. [4, 1]

[4]V režimu full-duplex jsou zařízení schopna současně data posílat i přijímat.
[5]V režimu half-duplex zařízení sice komunikují oběma směry, ale ne současně.



Obrázek 2.3: Přenos dat přes rozhraní QSPI [1]. Signály D0-D3 odpovídají obecným signálům IO.

2.5 Sběrnice MI32

MI (z angl. "Memory Interface") je sběrnice, kterou používá sdružení CESNET pro softwarovou komunikaci s firmwarovými komponentami. Jedinou používanou šířkou přenášených dat i adres je 32 bitů, od toho se odvíjí její časté ozačení MI32. [2]

2.5.1 Rozhraní

V tabulce 2.1 je znázorněno rozhraní MI pro slave komponentu. Signály fungují následujícím způsobem:

- `ADDR` slouží pro vystavení adresy pro zápis či čtení
- `DWR` představuje vystavená data, která mají být zapsána
- `MWR` jsou nepovinná metadata[6]
- `BE` slouží pro určení konkrétních validních bytů signálu `DWR` nebo `DRD`, se kterými má být pracováno
- `WR` značí požadavek na zápis dat (`DWR`) na adresu `ADDR`
- `RD` značí požadavek na čtení dat (`DRD`) z adresy `ADDR`
- `ARDY` se používá pro signalizaci, že je slave komponenta připravena přijímat další čtecí nebo zápisový požadavek
- `DRD` představuje vyčítaná data
- `DRDY` se používá pro signalizaci, že jsou na výstupu slave komponenty připravena platná data

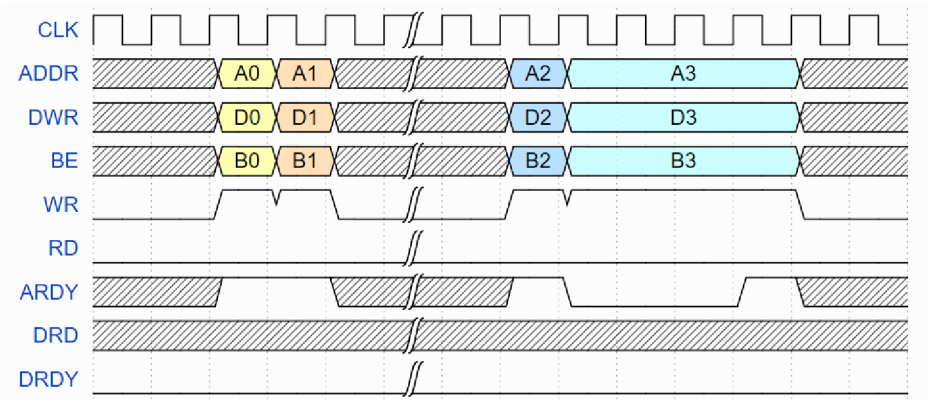
[6]Metadata v rámci MI sběrnice jsou zamýšlena pro potřebu zasílání informací sloužících pro identifikaci konkrétního virtuálního stroje běžícího na serveru (na jednom serveru bývá zpravidla spuštěno několik virtuálních strojů), kdy z tohoto virtuálního stroje byla posílána transakce do síťové karty, jenž je do serveru zapojena. Metadata mohou jinak sloužit pro přeposílání jakýchkoliv dat, které je potřeba přenášet nad rámec MI transakce.

Signál	I/O	Šířka (v bitech)	Název
master to slave			
ADDR	in	32	Address
DWR	in	32	Data write
MWR	in	libovolná	Metadata write
BE	in	4	Byte enable
WR	in	1	Write
RD	in	1	Read
ARDY	out	1	Address ready
slave to master			
DRD	out	32	Data read
DRDY	out	1	Data ready

Tabulka 2.1: Popis signálů rozhraní MI32 komponenty [2]

2.5.2 Časové diagramy

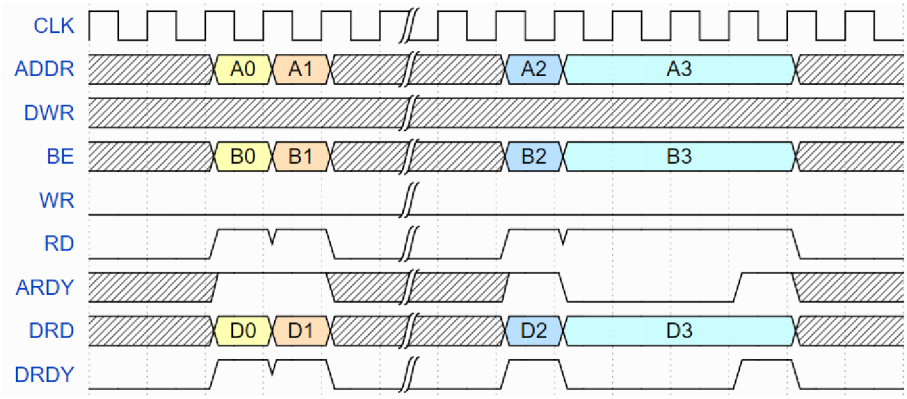
Pro ilustraci jsou v této podkapitole přiloženy dva obrázky popisující průběhy signálů na rozhraní MI při požadavku na zápis dat (obrázek 2.4) a při požadavku na čtení dat (obrázek 2.5).



Obrázek 2.4: Časový průběh signálů při požadavku na zápis na MI sběrnici [2]. Pro zápis musí být vystavena platná data na DWR a adresa na ADDR (BE slouží pro vybrání konkrétních bytů zapisovaných dat, se kterými se má dále pracovat). Pouze pokud jsou tyto nastaveny je možné nastavit požadavek na zapsání dat WR. Signál ARDY určuje, zda je slave komponenta připravena přijmout nový požadavek. Můžeme si tedy všimnout, že v případě signálu ARDY v log. 0 se na sběrnici nemění hodnoty signálů a čeká se, až budou data komponentou odebrána a ARDY bude opět nastaveno do log. 1.

2.6 Sběrnice Avalon Memory-Mapped

Firma Intel poskytuje mj. i množinu rozhraní a sběrnic, které je možné využít pro propojení komponent na jejích FPGA čipech. Tato rozhraní nesou souhrnný název Avalon. Avalon Memory-Mapped (dále jen Avalon-MM nebo AVMM) je sběrnice z této množiny zprostředkovávající rozhraní pro komponenty, které pracují na bazí čtení či zápisu na ad-



Obrázek 2.5: Časový průběh signálů při požadavku na čtení na MI sběrnici [2]. Situace na tomto obrázku je podobná průběhu signálů na obrázku 2.4. Logika pro reakci na nastavený ARDY signál je stejná. Místo požadavku na zápis WR je zde požadavek na čtení RD, který je ale, co se týče průběhu, svým chováním ekvivalentní signálu WR z předchozího obrázku. Rozdílem zde je, že na DRD se objevují vyčtená data, která validuje signál DRDY. Tato vyčtená data se mohou na sběrnici objevit už ve stejném taktu, ve kterém byl vznesen požadavek na jejich čtení, ale obecně tomu tak být nemusí.

resy. Komponenty jsou mezi sebou ve vztahu Host-Agent, což je de facto ekvivalentní vztah jako u modelu Master-Slave. [8]

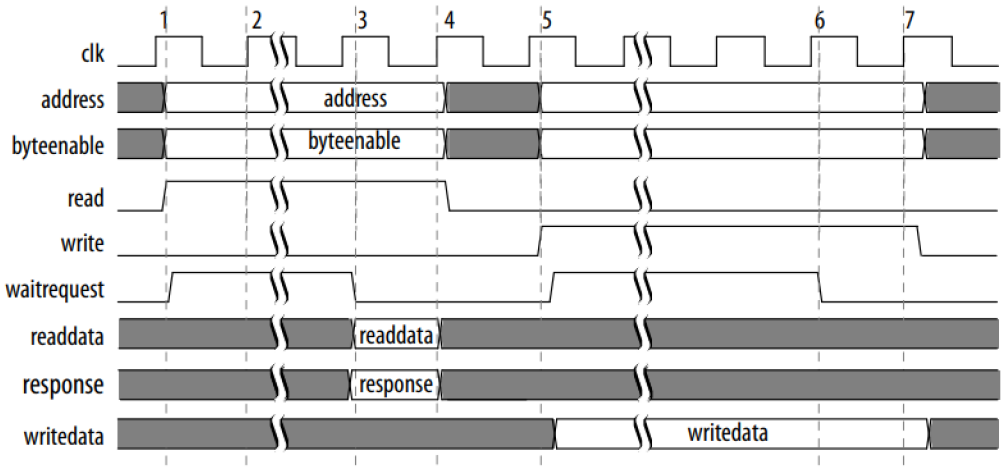
V tabulce 2.2 je znázorněno rozhraní Avalon-MM pro agent komponentu. Na obrázku 2.6 je pak časový průběh signálů ilustrující reálnou situaci Některé ze signálů mají i variantu pro signál aktivní v log. 0 (signal_n). Rozhraní podporuje základní signály, čekací signály, dále signály pro zřetězené operace a pro tzv. "burst"režim. Ani jeden ze všech signálů není vyžadován ve všech případech použití rozhraní. Signály fungují následujícím způsobem (burst režim není v této práci využíván, pro získání více informací o této skupině signálů viz dokumentaci [8]):

- `address` je adresa, na které má být prováděn zápis nebo čtení, kdy agent komponenta adresuje svuj paměťový prostor po 4bytových slovech
- `byteenable(_n)` se používá pro určení, se kterými byty datového slova se má pracovat
- `read(_n)` reprezentuje požadavek na čtení
- `readdata` jsou vyčtená slova
- `response` je nepovinný signál zpřístupňující informaci o tom, jak čtecí nebo zápisová operace dopadla
- `write(_n)` reprezentuje požadavek na zápis
- `writedata` jsou data určená pro zapsání
- `lock` je nepovinný signál pro hosta sloužící pro získání exkluzivního přístupu k danému agentovi
- `waitrequest(_n)` značí, že agent není schopný zpracovávat další čtecí nebo zápisové požadavky

- `readdatavalid(_n)` indikuje validitu signálu `readdata` (na rozdíl od rozhraní MI, u rozhraní Avalon-MM musí být přítomno zpoždění alespoň jeden hodinový takt mezi iniciováním čtecí operace pomocí signálu `read` a vyčtením platných dat za přítomnosti aktivního signálu `readdatavalid`)
- `writeresponsevalid` je nepovinný signál, který se používá pro aktivování zasílání stavových informací agentem v reakci na zápisové operace iniciované hostem prostřednictvím signálu `response`

Signál	I/O	Šířka (v bitech)
`address`	in	32
`byteenable(_n)`	in	4
`read(_n)`	in	1
`readdata`	out	32
`response`	out	2
`write(_n)`	in	1
`writedata`	in	32
`lock`	in	1
`waitrequest(_n)`	out	1
`readdatavalid(_n)`	out	1
`writeresponsevalid`	out	1

Tabulka 2.2: Popis signálů rozhraní Avalon-MM komponenty pro data i adresu o šířce 32 bitů [8]



Obrázek 2.6: Časový průběh signálů při čtecím a zápisovém požadavku za použití signálu `waitrequest` [8].

2.7 Software CESNET

Pro práci se síťovými kartami využívá sdružení CESNET rozsáhlý software[7], který zahrnuje mj. i skripty pro překlad ze zdrojových souborů nebo skripty pro sestavení tzv. Device Tree (viz podkapitolu 4.1), který se dále využívá pro softwarový přístup k hardwarovým komponentám zapojeným v designu pro FPGA. Dále tento software zahrnuje modul pro operační systém Linux implementující ovladače pro karty, knihovnu pro psaní uživatelských aplikací a samotné uživatelské aplikace. Mezi již v minulosti implementovanými aplikacemi se nachází i nástroj `nfb-boot` pro proces bootování karet. Náplní této práce v programové části bylo přidání podpory pro Intel FPGA (konkrétně již zmiňované čipy Stratix 10 a Agilex) do tohoto stávajícího software.

2.8 Konfigurace FPGA

V této sekci je přiblíženo, jak celý proces konfigurace FPGA čipu ze software funguje. Popsaný proces je ilustrován na obrázku 2.7.

2.8.1 Vytvoření bitstreamu

Nejprve je nutné vytvořit zdrojové soubory, ve kterých jsou jednotlivé hardwarové komponenty implementovány (nejčastěji ve VHDL či ve Verilogu). K implementaci je možné využít IP cores, které poskytuje výrobce čipu osazeného na kartě. Tyto jednotlivé komponenty jsou pak instanciovány a vzájemně propojeny v rámci celkového designu v `fpga.vhd` tak, aby bylo dosaženo požadované funkcionality.

Ke komponentám, které implementují funkcionalitu specifickou pro konkrétní čip (např. komunikace s Hard IP, jenž se na různých čipech liší), se navíc přidává soubor `DevTree.dts`, který se při sestavování designu pro kartu použije pro vytvoření celkového popisu zapojených komponent. Tento celkový popis designu se využívá v software pro vybrání správného kódu implementujícího ovladač příslušný k dané kartě (konkrétní příklad je uvedený v sekci 4.1).

Dalším krokem je překlad těchto zdrojových souborů, který provadí tzv. syntézní nástroj. Tento nástroj může být poskytovaný buď výrobcem FPGA čipu (pro Intel je integrován v Intel Quartus Prime – dále jen Quartus) nebo se může jednat o nástroj poskytovaný třetí stranou. Syntézní nástroj přímo od výrobce konkrétního FPGA čipu je pak potřeba pro vygenerování binárního souboru (bitstreamu) pro programování tohoto čipu. Důvodem je znalost organizace čipu potřebná k namapování komponent zapojených v našem designu do konkrétního použitého čipu. Vygenerovaný design je ve formátu `.sof` (SRAM Object File).

Pro tuto práci jsou důležité soubory dvou dalších formátů, konkrétně `.jic` (JTAG Indirect Configuration File) a `.rpd` (Raw Programming Data File). Pro jejich vygenerování se použije nástroj Programming File Generator (také součást nástroje Quartus), jemuž se na vstup předá v předchozím kroku vygenerovaný `.sof` soubor.

Soubor ve formátu `.jic` je používán nástrojem Intel Quartus Prime Programmer pro nahrání konfiguračního obrazu do QSPI flash konfigurační paměti přes rozhraní JTAG. Kromě samotných konfiguračních dat určených pro uložení do flash paměti obsahuje minimalistický pomocný design pro FPGA, přes který je nástroj následně schopný konfigurační data do flash paměti nahrát.

[7]Software včetně knihovny, modulu pro ovladače a uživatelských aplikací je volně dostupný na adrese `https://github.com/CESNET/ndk-sw`.

Soubor ve formátu `.rpd` obsahuje pouze data určená pro nahrání do flash paměti. Tento soubor se využívá v případě, kdy je nahrávání konfigurace do flash paměti řízeno externím nástrojem (ne v rámci software Quartus, ale například právě aplikací pro bootování karet od sdružení CESNET). Tento formát je možné využít v plné míře namísto výše zmiňovaného `.jic` formátu.

Pro účely implementace podpory pro Intel FPGA a její přidání do stávajícího software má význam zaměřit se zejména na formát `.rpd`. Soubor ve formátu `.jic` byl využit v této práci převážně na začátku pro ověření správné softwarové komunikace s implementovanou hardwarovou jednotkou a její správné funkcionality v rámci provádění jednoduchých čtení (resp. zápisů a mazání sektorů) QSPI flash konfigurační paměti.

2.8.2 Nahrání bitstreamu

Náplní softwarové části této práce je přidání podpory pro konfigurování Intel FPGA do stávajícího software, tedy jinými slovy přidání možnosti konfigurování FPGA daty ve formátu `.rpd`, protože právě pro tento formát dat platí, že je používán v případě, kdy je konfigurace Intel FPGA řízena externím nástrojem. Tento proces zahrnuje přidání podpory pro tento formát dat do uživatelské aplikace a do knihovny, která je používána při psaní uživatelských aplikací, a přidání nízkoúrovňových funkcí, které implementují dříve popisovaný protokol ze sekce 3.3.4, do ovladačů ke kartám.

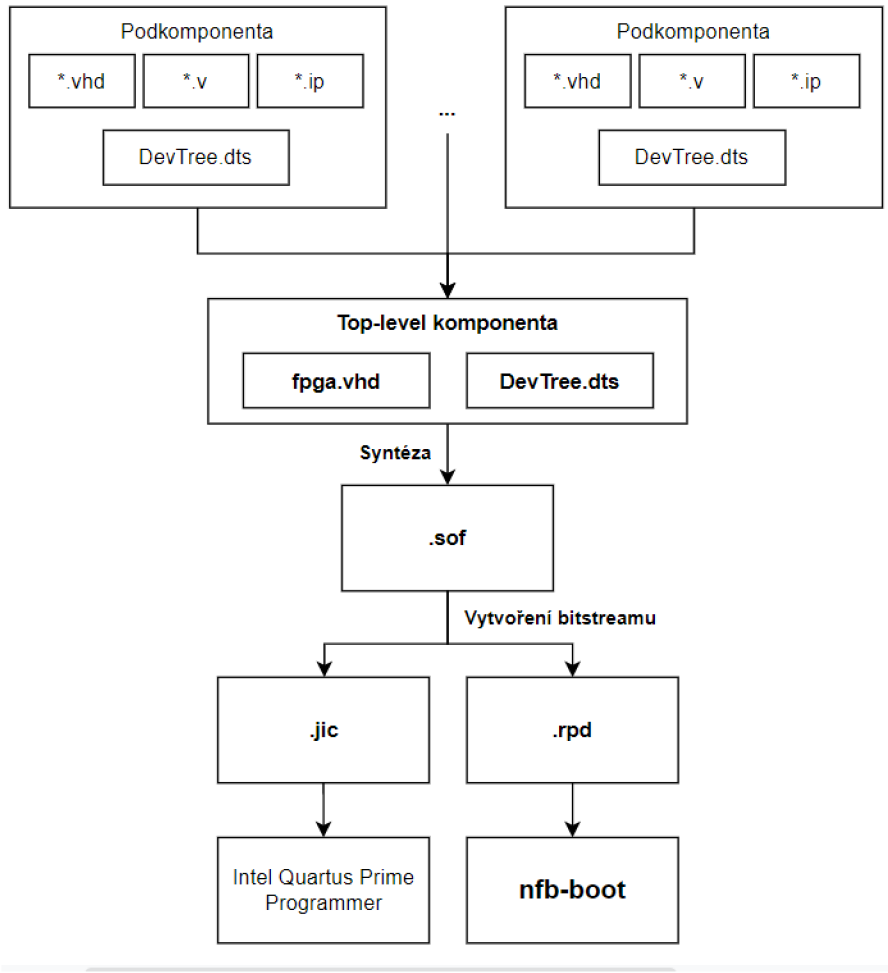
Pro nahrání konfiguračních dat ve formátu `.jic` se využije nástroj Intel Quartus Prime Programmer, který celý proces konfigurace plně zastřešuje. Nevýhodou použití tohoto způsobu konfigurace je, že je prováděn přes JTAG kabel, a tedy karta musí být fyzicky připojena speciálním kabelem k hostitelskému počítači. Konfigurační soubor ve formátu `.jic` je také díky pomocnému designu pro FPGA několikanásobně větší než soubor ve formátu `.rpd` obsahující pouze samotná konfigurační data. Podrobný postup, jak za pomoci `.jic` souboru nahrát konfiguraci do flash paměti, je popsán v sekci 5.6.3. v odpovídajících dokumentacích [9, 10].

2.9 Uživatelská aplikace a knihovna

Bitstream je společně se souborem obsahujícím Device Tree předán uživatelské aplikaci `nfb-boot` (viz sekci 4.2), která realizuje proces konfigurování FPGA čipu na kartě. Tato aplikace pro svou činnost používá funkce z knihovny `libnfb` (viz sekci 4.3), které v sobě ve výsledku zapouzdřují systémová volání operačního systému. Na základě informací o přítomné hardwarové jednotce pro ovládání konfigurace, které jsou získány z Device Tree (konkrétně typ této jednotky a adresa v designu karty, na které je jednotka namapována), je zkontrolován správný formát poskytnutého bitstreamu podporovaný danou jednotkou. V dalším kroku jsou volány funkce knihovny `libnfb` pro zápis bitstreamu (do speciálního souboru reprezentujícího v Linuxu zařízení), které v sobě zapouzdřují zmiňovaná systémová volání. Pro obsluhu systémových volání je v tuto chvíli řízení programu předáno jádru operačního systému.

2.10 Ovladače zařízení v Linuxu

Operační systém Linux podporuje možnost do svého jádra za jeho běhu dynamicky zavádět rozšiřující kusy kódu v podobě modulů, které zprostředkovávají mj. implementaci ovladačů



Obrázek 2.7: Proces vytvoření bitstreamu

pro zařízení v systému. Při zavádění takového modulu je modul registrován a je mu přiřazeno v rámci operačního systému unikátní číslo (Major number). U speciálních souborů reprezentujících v systému Linux zařízení (složka `/dev`) je toto číslo uvedeno, aby bylo jádro schopné při práci s takovým souborem identifikovat příslušný ovladač, kterému je nutné předat řízení pro docílení požadované funkcionality. S těmito speciálními soubory je manipulováno přes stejné rozhraní jako s klasickými soubory, konkrétně pomocí systémových volání jako jsou např. `lseek`, `write` nebo `read`. [12, 5]

Každý modul pro ovladač zařízení implementuje vybranou množinu funkcí z tohoto rozhraní a při jeho zavádění jsou tyto funkce v interních strukturách jádra namapovány (s využitím ukazatelů na funkce) na příslušná systémová volání. Pokud je tedy prováděn např. zápis pomocí `write` do speciálního souboru, jádro operačního systému předá řízení ovladači, který je registrován pod daným unikátním číslem uvedeným u tohoto souboru, a to voláním funkce, kterou samotný ovladač implementuje a registroval ji při svém zavádění jako obslužnou funkci pro systémové volání `write`. Slabinou tohoto přístupu je skutečnost, že kód ovladače běží v privilegovaném režimu jádra, kdy případná chyba v ovladači může vyústit v pád zařízení, a obecně ladění tohoto kódu je velmi problematické.

Tato práce přidává vybranou množinu funkcí, které budou obsluhovat systémová volání při práci s jednotkou SDM Client. Detailní pojednání o těchto funkcích se nachází v sekci 4.4.

Kapitola 3

Hardware

3.1 400GE karta

Označení 400GE nese nová akcelerační síťová karta, kterou vyvinulo sdružení CESNET ve spolupráci s francouzskou společností Reflex CES. Tato karta byla jedním z primárních cílových prostředků pro nově vytvářenou hardwarovou jednotku. Bohužel nakonec nebylo možné výslednou jednotku na této platformě otestovat (blíže vysvětleno v podkapitole 5.1). Místo této karty byl pro vývoj využit prostředek Intel Stratix 10 GX Development Kit (více v sekci 3.2).

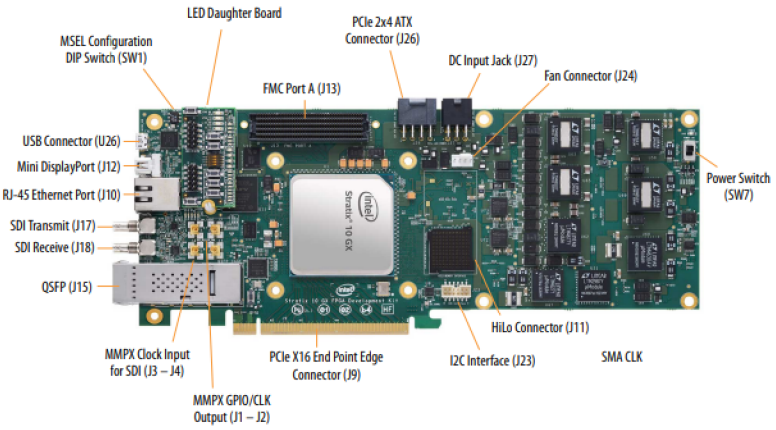
Tato nová karta (viz obrázek 3.1) na sobě nese FPGA Intel Agilex a podporuje nejnovější standard PCI 5.0, přičemž ale i v případě použití PCIe 4.0 sběrnic umožňuje díky druhému rozšiřujícímu PCIe konektoru dosažení rychlosti přenosu dat 400Gb/s. Karta je primárně určena pro monitorování sítě a pro aplikace pro bezpečnost počítačové sítě. Pro celou tiskovou zprávu viz článek [3].

Obrázek 3.1: 400GE karta [3]

3.2 Intel Stratix 10 GX Development Kit (verze ES)

Primární vývojovou platformou pro tuto bakalářskou práci byl prostředek Intel Stratix 10 GX Development Kit (dále jen DevKit). K dispozici byl ve verzi ES[1].

Na prostředku je osazeno FPGA Intel Stratix 10 GX a mnoho dalších komponent, které ale přímo nesouvisí s touto prací [7]. Pro účely konfigurace je na desce sada manuálních přepínačů, kterými se ovládá signál MSEL, viz obrázek 3.2 v levém horním rohu. Karta dále podporuje také všechna ostatní konfigurační schémata, která byla popsána v sekci 2.3.1.



Obrázek 3.2: Intel Stratix 10 GX Development Kit – přední strana [7]

3.2.1 Konfigurační QSPI paměť

Jako konfigurační QSPI flash paměť byla použita paměť EPCQ1024L od firmy Intel. Tato paměť byla osazená na DevKitu již ve výchozím stavu. Jelikož se nejedná o produkt třetí strany, je plně kompatibilní s FPGA čipy od firmy Intel a zároveň přirozeně podporuje konfigurační schéma Active Serial.

Paměť je vyrobena technologií NOR (více o technologii NOR je k nalezení např. v článku na adrese `https://www.utmel.com/blog/categories/memory%20chip/nor-flash-working-structure-and-applications`) a má kapacitu 1Gb. Pro práci s pamětí je nutné komunikovat skrze SDM. Pro tento účel se používá řada příkazů, které jsou blíže popsány v sekci 3.3.5.

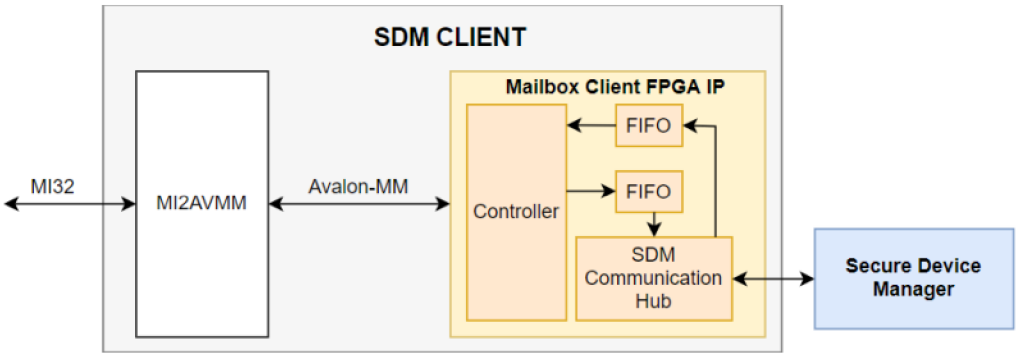
Paměť obsahuje celou řadu registrů. Pro čtení a zápis těchto registrů se používají operační kódy. V této práci jsou používány zejména operační kódy pro čtení identifikačního čísla zařízení (0x9E nebo 0x9F) čtení stavového registru (0x05) a povolení/zakázání zápisů (0x06/0x04) [6]. Co se týče práce s registry paměti, z nejasných důvodů komunikace v tomto ohledu neprobíhala podle specifikace (bližší informace jsou k nalezení v sekci 5.3).

3.3 Implementovaná jednotka SDM Client

SDM Client je výsledná jednotka pro FPGA sloužící pro komunikaci s externí QSPI flash konfigurační pamětí dle bodu 2 zadání. Cílem návrhu bylo, aby samotná hardwarová jednotka byla co nejjednodušší a ve výsledku se tak dalo přesunout veškeré řešení konfigurační

[1]ES (z angl. "Engineering Sample") značí beta verzi integrovaného obvodu, která slouží jako demonstrační prostředek, takže nemusí podporovat veškerou funkcionalitu, kterou podporuje plná verze.

logiky do softwarové části navrhovaného systému. Aby bylo možné pracovat se zmiňovanou flash pamětí, musí komunikace probíhat skrze SDM, který v sobě zahrnuje mj. i rozhraní QSPI pro konfigurační režim Active Serial (blok AS v sekci Peripherals, viz obrázek 2.2). Z manuálu pro konfiguraci FPGA [9, 10] bylo zjištěno, že pro komunikaci designu a SDM je možné využít IP core poskytovaný firmou Intel, konkrétně Mailbox Client Intel FPGA IP (detailně jsou popis tohoto IP a práce s ním rozebrány v sekci 3.3.2). Toto IP má na vstupu rozhraní Avalon-MM (viz podkapitolu 2.6), pro které bylo nutné přidat převodník na MI rozhraní (pro více informací o sběrnici MI viz sekci 2.5). Princip tohoto převodníku a odlišnosti obou rozhraní jsou blíže popsány v sekci 3.3.1. Výsledná jednotka SDM Client, která je k vidění na obrázku 3.3, je tedy připojená na MI sběrnici, interně převádí rozhraní MI na rozhraní Avalon-MM a instanciuje v sobě komponentu Mailbox Client IP, přes kterou je realizována komunikace s SDM. Přes rozhraní MI jsou poté ze software do jednotky zasílány příkazy pro SDM a taktéž z ní vyčítány odpovědi od SDM. Rozhraní jednotky SDM Client odpovídá rozhraní MI slave komponenty v tabulce 2.1.
Implementace jednotky SDM Client v jazyce VHDL je volně dostupná na adrese `https://github.com/CESNET/ofm/tree/main/comp/ctrls/sdm_client`.



Obrázek 3.3: Jednotka pro obsluhu konfigurování FPGA

3.3.1 MI2AVMM

MI2AVMM je obecný převodník rozhraní MI (viz podkapitolu 2.5) na rozhraní Avalon-MM (viz podkapitolu 2.6). Obě rozhraní jsou navzájem téměř kompatibilní. Rozdíl v komunikaci nastává u ARDY signálu MI sběrnice, kdy master strana počítá s tím, že pokud je signál aktivní (log. 1), slave zařízení je připraveno přijmout další čtecí/zápisový požadavek. Na druhé straně u Avalon-MM rozhraní aktivní signál (log. 1) značí, že je slave zařízení zaneprázdněno a požaduje čekání před zasíláním dalších požadavků – AVMM_WAITREQUEST. Toto je u převodníku řešeno negací signálu.

3.3.2 Mailbox Client IP

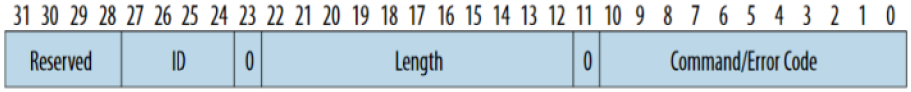
Konkrétní Soft IP, které se dá použít pro komunikaci s SDM, je Mailbox Client Intel FPGA IP. Tato komponenta funguje jako prostředník mezi SDM a designem v FPGA, kde na straně SDM používá rozhraní Avalon-ST (více viz dokumentaci [8]) a na straně FPGA rozhraní Avalon-MM (Avalon Memory-Mapped). Komunikace pak probíhá skrze zasílání paketů s příkazy a vyčítání paketů s odpovědmi od SDM přes vstupní a výstupní FIFO paměti této komponenty. Zasílání příkazů a čtení odpovědí se řídí definovaným protokolem,

který říká, co se v jakém pořadí zapisuje na jaké adresy adresního prostoru komponenty (viz podkapitolu 3.3.4).

Mailbox Client IP pracuje s 32bitovými datovými slovy. Jemu zasílané příkazy pak mohou sloužit pro celou řadu úkonů. Dají se využít např. pro vyčítání Chip ID, hodnot teplotního a napětového senzoru v SDM či konfiguračních informací. Hlavním cílem v této práci ale bude používat ho pro čtení a zejména pak zápisy do externí quad SPI (QSPI) flash paměti a provádění Remote System Update, což se využije pro zápis bitstreamu do externí QSPI flash paměti a následné vynucení rekonfigurace zařízení pomocí tohoto zapsaného bitstreamu.

3.3.3 Formát příkazu a odpovědi

Každý příkaz a každá odpověd začíná hlavičkou (viz obrázek 3.4), která obsahuje ID příkazu/odpovědi (u odpovědi určuje, ke kterému předchozímu příkazu se váže), délku (počet zapisovaných/vyčítaných 32bitových slov) a kód příkazu (chyby u odpovědi). Následuje sekvence argumentů, kdy se jejich význam a jejich počet odvíjí od použitého příkazu.



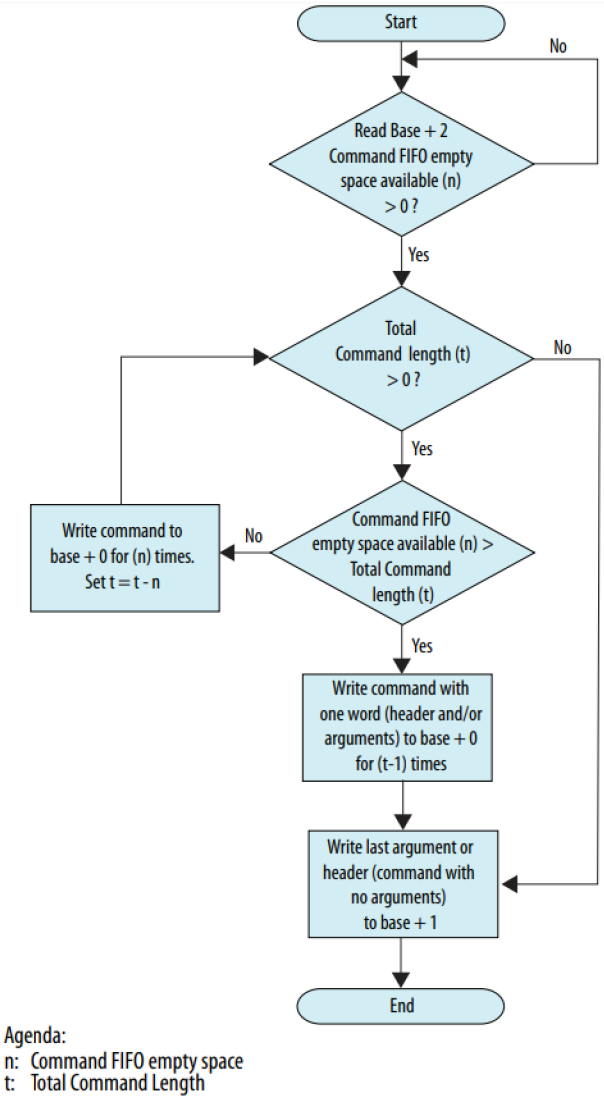
Obrázek 3.4: Formát hlavičky paketu [11]

Offset (32bit)	R/W	31	30:2	1	0
BA + 0	W	Command			
BA + 1	W	Command last word (eop)			
BA + 2	R	Command FIFO empty space			
BA + 3	N/A	Reserved			
BA + 4	N/A	Reserved			
BA + 5	R	Response data			
BA + 6	R	Response FIFO fill level		EOP	SOP
BA + 7	R/W	Interrupt enable register (IER)			
BA + 8	R	Interrupt status register (ISR)			
BA + 9	R/W	Timer 1 enable	Timer 1 period		
BA + 10	R/W	Timer 2 enable	Timer 2 period		

Tabulka 3.1: Adresní prostor Mailbox Client Intel FPGA IP [11]. BA (Base Address) je počáteční adresa, na které je komponenta Mailbox Client Intel FPGA IP namapována.

3.3.4 Protokol

Zaslání příkazu začíná zápisem slova obsahujícího hlavičku do vstupní FIFO paměti, která je namapována v adresním prostoru komponenty na adrese BA + 0 (viz tabulku 3.1). Dále následuje zapisování argumentů (kromě posledního slova) daného příkazu sekvenčně na tutéž adresu. Poslední slovo je poté zapsáno na adresu BA + 1, což určuje konec příkazu. V průběhu zápisu jednotlivých slov je nutné kontrolovat zaplnění vstupní FIFO paměti, a to čtením z adresy BA + 2, kde je informace o její volné kapacitě. Diagram detailně znázorňující logiku zasílání příkazů je na obrázku 3.5.



Obrázek 3.5: Zaslání příkazu do Mailbox Client IP [11]

Pro čtení odpovědi musí být nastaven nejnižší bit registru ISR (BA + 8) do log. 1, což indikuje, že se ve výstupní FIFO paměti nachází validní data. Dále je potřeba vyčíst zaplnění výstupní FIFO paměti z adresy BA + 6 a současně kontrolovat SOP (začátek paketu, z angl. "start of packet") a EOP (konec paketu, z angl. "end of packet") ohraničující daný paket s odpovědí. Samotná odpověď je pak vyčtena z adresy BA + 5, kde je namapována výstupní FIFO paměť komponenty. Při čtení odpovědi je také vhodné kontrolovat celkový počet slov odpovědi získaný při čtení hlavičky odpovědi, tj. prvního slova (viz podkapitolu 3.3.3). Diagram detailně znázorňující proces získání odpovědí je na obrázku 3.6.

3.3.5 Příkazy pro práci s flash pamětí a konfiguraci zařízení

V této podkapitole se nachází přehled příkazů, pomocí kterých je možné skrze SDM komunikovat s konfigurační QSPI flash pamětí. U každého z příkazů je uveden jeho operační kód (hexadecimálně). Je nutné myslet na to, že do délky příkazu uváděné v hlavičce se ne-

započítává samotná hlavička, ale pouze slova, která za ní následují. Detailní přehled těchto příkazů je k nalezení v dokumentaci k Mailbox Client Intel FPGA IP [11].

- QSPI_OPEN (0x32) – získání exkluzivního přístupu k flash paměti
- QSPI_CLOSE (0x33) – uzavření exkluzivního přístupu k flash paměti
- QSPI_SET_CS (0x34) – vybrání jednoho z připojených QSPI zařízení (můžou být až 4, ale pro konfiguraci FPGA čipu se vždy používá pouze to, které je přístupné přes hodnotu `00`, ostatní slouží pouze jako úložiště)
- QSPI_READ (0x3A) – požadavek na čtení z externí paměti
- QSPI_WRITE (0x39) – požadavek na zápis do externí paměti
- QSPI_ERASE (0x38) – vymazání sektoru v paměti
- QSPI_READ_DEVICE_REG (0x35) – čtení registru externí flash paměti
- QSPI_WRITE_DEVICE_REG (0x36) – zápis do registru externí flash paměti
- RSU_IMAGE_UPDATE (0x5C) – požadavek na spuštění rekonfigurace zařízení ze zvoleného aplikačního nebo továrního obrazu

Pro zajištění správné komunikace s flash pamětí, musí být při práci s ní dodržena následující sekvence příkazů:

- QSPI_OPEN
- QSPI_SET_CS
- libovolné příkazy pro zápis, čtení a mazání flash paměti a jejích registrů
- QSPI_CLOSE

3.3.6 Implementační detaily

Při práci se samotným Mailbox Client IP je nutné vzít v potaz několik skutečností:

- MI sběrnice adresuje po bytech, zatímco sběrnice Avalon-MM v případě agentní komponenty Mailbox Client IP adresuje po 4bytových slovech. V jednotce SDM Client je tedy potřeba pracovat s adresami zarovnanými na 4 byty. Spodní 2 bity adresy vystavené na MI sběrnici jsou tedy ignorovány, neboť celkově je adresa v jednotce SDM Client bitově posunuta o tyto 2 bity doprava a dále se pracuje pouze s touto bitově posunutou adresou. V jednotce je použita šířka dat i adresy 32 bitů.
- Mailbox Client IP nemá signál AVMM_WAITREQUEST, který je součástí standardního Avalon-MM rozhraní a určuje (v log. 1), že je komponenta zaneprázdněná a vyžaduje čekání. Kontrola dostupného místa ve vstupní FIFO paměti komponenty (a tedy i schopnost přijímat další požadavky) se provádí čtením z adresy BA+2 v jejím adresním prostoru (viz tabulku 3.1, postup komunikace s komponentou je blíže popsán v sekci 3.3.4). Tento signál je tedy v top-level komponentě trvale připojen na log. 0, což značí, že je komponenta Mailbox Client připravena přijímat požadavky vždy. Případná kontrola kapacity pak musí být provedena uvedeným způsobem.

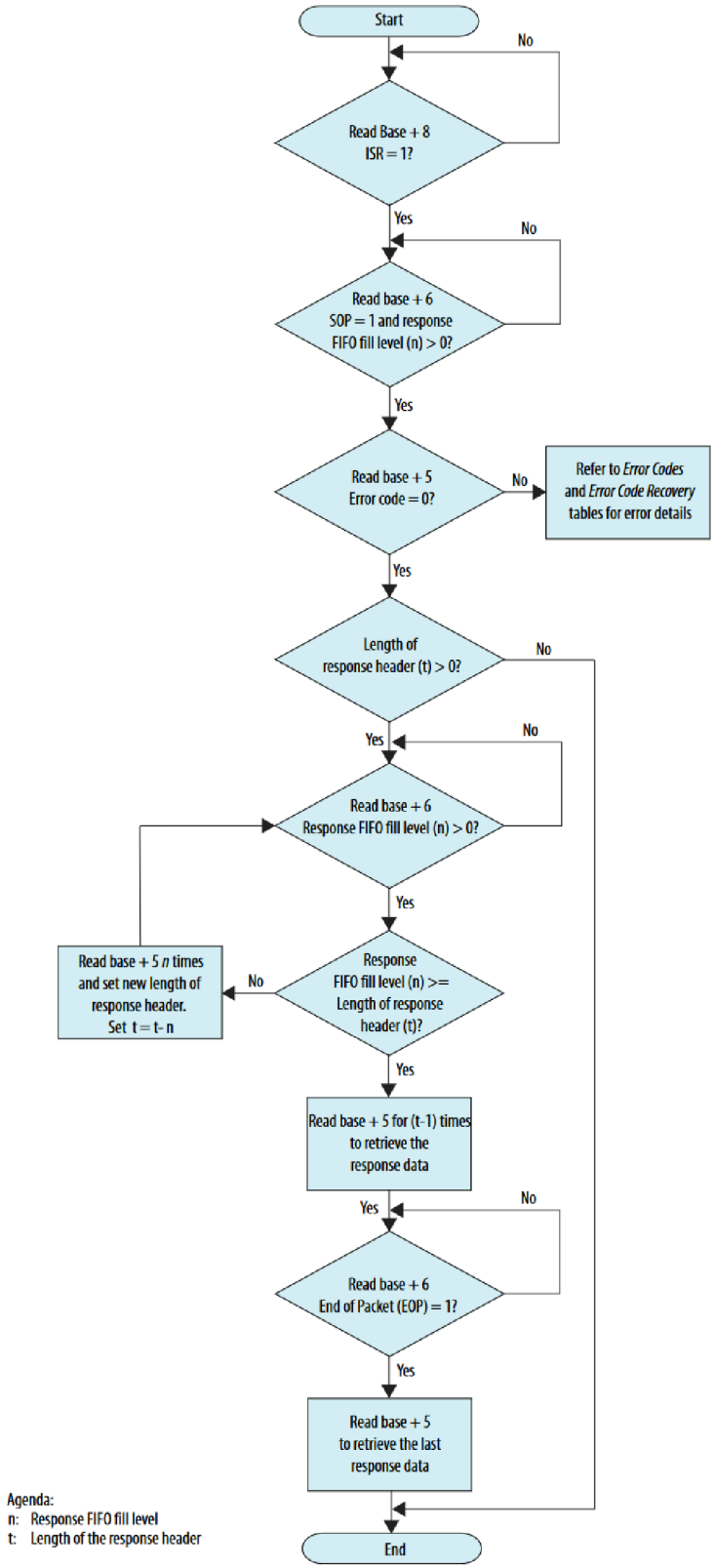
- Mailbox Client na rozdíl od MI sběrnice podporuje signál IRQ pro signalizování přerušení (mj. např. v případě dostupných dat ve výstupní FIFO paměti komponenty). Že se ve výstupní FIFO paměti nalézají platná data se ale dá zjistit přes registr ISR na adrese BA+8 (viz tabulku 3.1), kdy je v tomto případě jeho nejnižší bit nastaven do log. 1, takže absence tohoto signálu na rozhraní MI není velkou překážkou.
- Mailbox Client IP nepodporuje signál BE z rozhraní MI (resp. Avalon-MM). V příchozích transakcích je tedy BE ignorován a každý byte datového slova je považován za platný.

3.4 Remote System Update

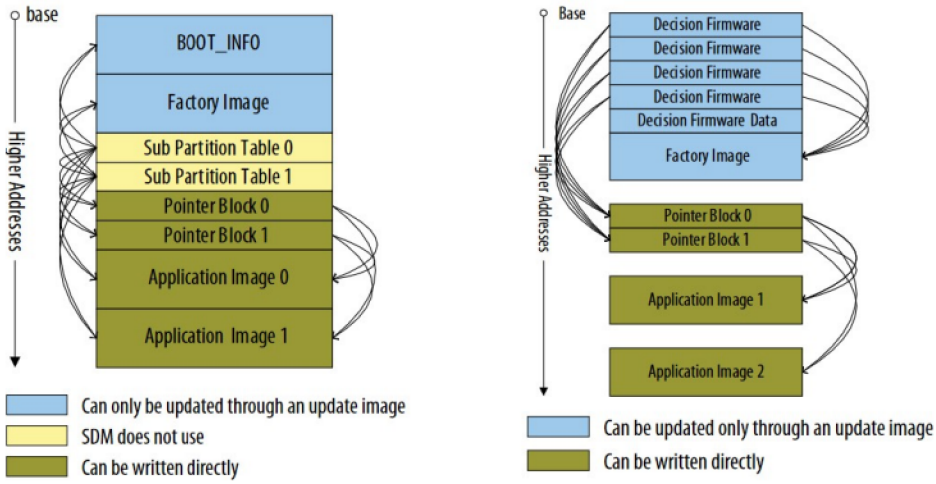
Pro spuštění rekonfigurace zařízení pomocí zaslání příkazu skrze Mailbox Client IP se ve zvolených FPGA od firmy Intel dá použít tzv. Remote System Update (RSU), kdy logika, která ho implementuje, je součástí SDM. RSU je běžně dostupný při konfiguraci v režimu Active Serial, neboť pracuje s externí QSPI flash pamětí. RSU umožňuje uchovávat v konfigurační paměti jeden tovární obraz a několik aplikačních obrazů paralelně vedle sebe. Pomocí Mailbox Client IP je možné nahrát do paměti nový aplikační obraz (QSPI_WRITE) a pomocí RSU lze vynutit rekonfiguraci zařízení za použití tohoto obrazu (RSU_IMAGE_UPDATE). Podstatou tohoto systému tedy je, že umožňuje na dálku jednoduše opravit chybný design či změnit funkcionalitu zařízení. SDM se vždy snaží nakonfigurovat zařízení nejprve z aplikačního obrazu s nejvyšší prioritou (nejvyšší prioritu má nastavenou nově nahraný obraz). Pokud se mu to nepovede, pokračuje konfigurováním z ostatních aplikačních obrazů dle jejich priority. Pokud selže u všech aplikačních obrazů, nakonfiguruje zařízení z továrního obrazu.

3.4.1 Organizace QSPI flash paměti za použití RSU

Pro využívání RSU je na začátku nutné vytvořit počáteční RSU obraz pomocí nástroje `quartus_pfg`, který v externí paměti vytvoří strukturu podobnou té, která je zobrazená na obrázku 3.7. Pro tento počáteční RSU obraz je nutné poskytnout tovární obraz a alespoň jeden aplikační obraz.



Obrázek 3.6: Vyčtení odpovědi z Mailbox Client IP [11]



Obrázek 3.7: Adresní prostor konfigurační flash paměti [9, 10]. Na obrázku vlevo je znázorněno rozložení paměti, jak ho vidí uživatel. Pro přehled o obsahu paměti a řízení alokace zdrojů se používá Sub Partition Table (SPT), která odkazuje na veškerý obsah paměti. Na obrázku vpravo je znázorněn úhel pohledu SDM, kde je vidět provázání Decision Firmware s ukazateli na jednotlivé aplikační obrazy. V obou případech je vidět, že je možné přímo zapsat nový aplikační obraz a také přímo modifikovat Configuration Pointer Block (CPB) přidáním ukazatele na tento nový aplikační obraz. Pro vyšší spolehlivost vytváří nástroj SPT i CPB ve dvou kopiích.

Kapitola 4

Software

Softwarová rozšíření pro Intel FPGA, která jsou popisována v této kapitole, nejsou zatím dostupná v aktuální verzi na githubu[1] (v6.16.0), neboť tam se jedná pouze o stabilní verzi a interní vývoj se nachází v soukromém repozitáři na gitlabu. Rozšíření ale budou zveřejněna na githubu v rámci některé z příštích verzí.

4.1 Device Tree

Jak již bylo zmíněno v teoretické části, v průběhu překladu designu je současně také skládána hierarchická reprezentace speciálních komponent zapojených v daném designu pro FPGA (tzv. Device Tree). Jelikož pro proces konfigurování Intel FPGA v režimu Active Serial bude jako řadič používána nově implementovaná hardwarová jednotka SDM Client (viz podkapitolu 3.3), která je odlišná od již dříve implementovaných řadičů pro jiná FPGA, v Device Tree je pro ni nutno vytvořit vlastní záznam, aby bylo v software možné vybírat nově implementované funkce pracující s touto jednotkou, pokud bude tato jednotka v designu zapojena. V software je rozhodováno na základě atributu `compatible`, kdy tento nový řadič pro Intel FPGA čipy je v Device Tree identifikován pod označením "netcope,intel_sdm_controller". V Device Tree se také nachází informace o adrese, na níž je řadič (resp. Mailbox Client IP) v designu namapován. Tato adresa je tedy počáteční adresou komponenty (BA v tabulce 3.1). V příloze A je k vidění reálný kód, který tuto funkci zajišťuje.

4.2 `nfb-boot`

Aplikace `nfb-boot` slouží uživateli pro ovládání konfigurování zvoleného `nfb` (Netcope FPGA board) zařízení. Zajišťuje především nahrávání bitstreamu do konfigurační paměti a bootování zařízení (a libovolnou kombinaci těchto funkcí v závislosti na použitých přepínačích). Bitstream a Device Tree soubor jsou aplikaci předávány v archivu `.tar`.

Aplikace `nfb-boot` pracuje s tzv. sloty, které reprezentují prostor pro uložení jednotlivých obrazů v konfigurační flash paměti. Tyto sloty jsou pro každé zařízení specifikovány přímo ve zdrojovém kódu ovladačů[2]. Klasicky jsou u předchozích karet podporovány pouze

[1] `https://github.com/CESNET/ndk-sw`

[2] konkrétní příklad volání funkce `nfb_fdt_create_binary_slot` pro vytvoření takového slotu pro kartu NFB-200G2QL je k nahlédnutí na adrese `https://github.com/CESNET/ndk-sw/blob/main/drivers/kernel/drivers/nfb/pci.c#L182`

dva sloty, kdy první obsahuje tovární obraz, kterým je FPGA čip konfigurován při bootování, a do druhého (pro aplikační obraz) může uživatel nahrát svůj obraz s vlastním designem a vynutit si z něho rekonfiguraci zařízení. Aplikace dále umožňuje pro dané zařízení vypsat seznam dostupných konfiguračních slotů.

Podrobnější popis aplikace je uveden v dokumentaci, viz `https://cesnet.github.io/ndk-sw/tools/nfb-boot.html`. Zdrojový kód aplikace je k nalezení na adrese `https://github.com/CESNET/ndk-sw/blob/main/tools/boot/boot.c`.

RSU (viz podkapitolu 3.4) u Intel FPGA čipů ale umožňuje pracovat hned s několika aplikačními obrazy, kdy jejich počet je limitován kapacitou konfigurační flash paměti a velikostí těchto obrazů. Práce s konkrétním slotem (ať už pro aplikační nebo tovární obraz) poté stejně jako v původní verzi programu zahrnuje zapsaní nového konfiguračního obrazu do tohoto slotu a nakonfigurování čipu obrazem z daného slotu. V závěru této práce je diskutováno rozšíření software o dynamickou správu obrazů v konfigurační flash paměti, díky které by nebylo potřeba mít pro toto konfigurační schéma záznamy o dostupných slotech pro dané zařízení přímo ve zdrojovém kódu ovladačů (funkcionalita je uvedena jako možné rozšíření a není přímo implementována z důvodů uvedených v sekci 5.1).

Pro přidání podpory pro Intel FPGA bylo nutné přidat podmínku, kdy v případě, že se formát bitstreamu liší od toho již dříve podporovaného, zkontroluje program, zda mu nebyl poskytnut nově podporovaný formát `.rpd` pro konfigurování Intel FPGA v režimu Active Serial. Pokud ano, předává dále bitstream v tomto formátu knihovním funkcím. Z důvodů uvedených v sekci 5.1 se vztahuje implementované rozšíření pouze na možnost zápisu bitstreamu ve formátu `.rpd` do zvoleného slotu, nikoli však na možnost vynucení rekonfigurace zařízení (více viz přílohu B).

4.3 `libnfb`

Tato knihovna v sobě implementuje funkce pro práci s `nfb` zařízeními, mj. funkce pro zpracování bitstreamu a funkce pro zápis a čtení adresního prostoru jednotlivých komponent, které jsou v designu FPGA připojené na MI sběrnici, a tedy softwarově dostupné. Stručný tutoriál seznamující s knihovnou a s prací s ní je k nalezení na `https://cesnet.github.io/ndk-sw/`.

Do knihovny `libnfb`[3] byla přidána podpora pro nový formát konfiguračních dat `.rpd`. V případě těchto dat je nutné si dát pozor na fakt, že při nahrávání bitstreamu do konfigurační paměti je pro každý byte dat nejprve zapisován nejméně významný bit, tedy je potřeba každý byte zapisovaných dat bitově překlopit (např. 0xAF → 0xF5). Pokud je z Device Tree designu zjištěno, že je v něm zapojena jednotka SDM Client (atribut `compatible` je nastaven na "netcope,intel_sdm_controller"), je tento nový `.rpd` formát odpovídajícím způsobem zpracován (implementace viz přílohu C).

[3] zdrojový kód je k dispozici na `https://github.com/CESNET/ndk-sw/tree/main/libnfb`

4.4 Ovladač

Poslední a zároveň nejvetší implementační částí je přidání sady nízkoúrovňových funkcí implementujících příkazy ze sekce 3.3.5 a komunikační protokol ze sekce 3.3.4, které tvoří ovladač pro nově vytvořenou hardwarovou jednotku SDM Client. Stejně jako v případě uživatelské aplikace a knihovny se funkcionalita v kódu ovladačů řídí podle sestaveného Device Tree.

4.4.1 Začlenění jednotky do stávajícího software

Pro proces bootování je v kódu ovladačů přítomna struktura typu `nfb_boot`, do které byly v rámci přidání podpory pro Intel FPGA přidány atributy (viz kód 4.1), které se nastaví při procházení Device Tree obsahujícího SDM Client (viz kód 4.2). Struktura `nfb_boot` v sobě také obsahuje ukazatele na strukturu představující komponentu pro obsluhu konfigurace (`struct nfb_comp *comp`) a struktury reprezentující konfigurační flash paměť (`struct spi_nor *nor` a `struct mtd_info **mtd`).

V případě, že byly dříve zmiňované nově přidané atributy nastaveny, tedy je v designu zapojena jednotka SDM Client, dojde k namapování nově implementovaných nízkoúrovňových funkcí na funkce z rozhraní této flash paměti (viz kód 4.3) a to vyústí ve volání těchto nově implementovaných funkcí v případě práce se zmiňovanou flash pamětí (`nor`), které budou ve výsledku zapisovat data do komponenty `comp` (jednotky SDM Client).

```
struct nfb_boot {
    struct nfb_comp *comp;
    ...
    struct spi_nor *nor;
    struct mtd_info **mtd;
    ...
    int sdm_controller;
    unsigned sdm_cmd_id;
    unsigned sdm_empty_space;
    ...
}
```

Výpis 4.1: drivers/kernel/drivers/nfb/boot/boot.h: Přidáné atributy do struktury `nfb_boot` (`sdm_controller` slouží pro identifikaci, že se v designu nachází jednotka SDM Client, `sdm_cmd_id` určuje ID příkazu, viz obrázek 3.4, `sdm_empty_space` slouží pro uchování informace o volné kapacitě vstupní FIFO paměti)

```
int nfb_boot_attach(struct nfb_device *nfb, ...) {
    ...
    struct nfb_boot *boot;
    ...
    fdt_offset = fdt_node_offset_by_compatible(nfb->fdt, -1, \
        "netcope,intel_sdm_controller");
    if (fdt_offset < 0) {
        fdt_offset = fdt_node_offset_by_compatible(nfb->fdt, -1, \
            "netcope,boot_controller");
        ...
    } else {
        boot->sdm_controller = 1;
        boot->sdm_cmd_id = 0;
        boot->sdm_empty_space = 0;
    }
    ...
}
```

Výpis 4.2: drivers/kernel/drivers/nfb/boot/boot.c: Útržek kódu znázorňující nastavení atributů struktury v případě nalezení jednotky SDM Client (pokud není nalezena v designu jednotka SDM Client, hledá se jiný řadič pro ovládání konfigurace – "netcope,boot_controller"). `nfb_device` je struktura reprezentující zařízení (kartu), která v sobě mj. zahrnuje i ukazatel `fdt` (Flat Device Tree) na Device Tree designu.

Rozhraní pro práci s jednotkou SDM Client

Pro potřebu namapování nově vytvořených funkcí na ukazatele poskytované rozhraním pro práci s flash pamětí je nutné, aby se tyto funkce řídily danými předpisy. Ovladač implementuje následující množinu funkcí z rozhraní (detailněji popsány v podkapitole 4.6), kde tyto funkce odpovídají příkazům pro Mailbox Client IP v jednotce SDM Client pro práci s konfigurační pamětí (viz sekce 3.3.5):

- `sdm_qspi_read` (QSPI_READ)
- `sdm_qspi_read_reg` (QSPI_READ_DEVICE_REG)
- `sdm_qspi_write` (QSPI_WRITE)
- `sdm_qspi_write_reg` (QSPI_WRITE_DEVICE_REG)
- `sdm_qspi_erase` (QSPI_ERASE)

```
int nfb_boot_mtd_init(struct nfb_boot *nfb_boot) {
    ...
    struct spi_nor *nor;
    ...
    if (nfb_boot->spi || nfb_boot->sdm_controller) {
        ...
        nor->priv = nfb_boot;
```

```
        if (nfb_boot->sdm_controller) {
            nor->read = sdm_qspi_read;
            nor->read_reg = sdm_qspi_read_reg;
            nor->write = sdm_qspi_write;
            nor->write_reg = sdm_qspi_write_reg;
            nor->erase = sdm_qspi_erase;
            nor->mtd.name = "sdm_qspi_nor";
        } else {
            ...
        }
    }
}
```

Výpis 4.3: drivers/kernel/drivers/nfb/boot/flash.c: Na tomto kusu kódu je k vidění namapování nízkoúrovňových funkcí pro obsluhu systémových volání směřovaných na konfigurační flash paměť. Přes ukazatel `nor->priv` je ve funkcích přistupováno ke struktuře `nfb_boot`. Pro práci s flash pamětí se pak používá mapování skrze strukturu `mtd_info` (Memory Technology Device, více viz např. `http://www.linux-mtd.infradead.org/doc/general.html`)

4.5 Pomocné funkce pro ovladač

Tato a následující sekce podrobně rozebírají rozhraní a implementaci nových funkcí pro ovladač k jednotce SDM Client. V kódu je přidáno pět pomocných funkcí, které jsou poté využívány funkcemi z rozhraní ovladače (viz následující sekce 4.6). Tyto jsou:

- `sdm_send_data` se stará o zaslání dat jednotce
- `sdm_get_data` vyčítá data z jednotky
- `sdm_send_header` se používá pro vytvoření a odeslání hlavičky příkazu
- `sdm_qspi_open_set_cs` je spojením příkazů QSPI_OPEN a QSPI_SET_CS ze sekce 3.3.5
- `sdm_qspi_close` implementuje funkcionalitu příkazu QSPI_CLOSE ze sekce 3.3.5

Tyto funkce a funkce z rozhraní ovladače (soubory `sdm.c` a `sdm.h`) přímo implementují protokol ze sekce 3.3.4 a příkazy ze sekce 3.3.5 (kromě RSU_IMAGE_UPDATE, pro blížší vysvětlení viz podkapitolu 5.1). Pracují s adresním prostorem jednotky SDM Client (resp. komponenty Mailbox Client IP) popsaným v tabulce 3.1, kdy k této jednotce je umožněn přístup pomocí ukazatele `((struct nfb_boot *)nor->priv)`, pro který je v kódu použito makro BOOT. Pro zápis do jednotky je volána knihovní funkce `nfb_comp_write32`, pro čtení z jednotky funkce `nfb_comp_read32`. První tři jmenované funkce dostávají jako parametr ukazatel na strukturu `nfb_boot`.

4.5.1 funkce `sdm_send_data`

Tato funkce se používá pro korektní odeslání dat do jednotky SDM Client. Data pak zapisuje buď na adresu, kde je namapována vstupní FIFO paměť nebo na adresu určenou pro zápis posledního slova příkazu (toto je určeno předaným parametrem `addr`). Tato data jsou funkci

předávána skrze vstupní buffer `data` (datový typ `u_char*`) o délce `len`, kde velikost jedné položky v bufferu je 1 byte. Funkce nevrací žádnou hodnotu.

Funkce nejprve čte informaci o volné kapacitě vstupní FIFO paměti komponenty, kterou si ukládá do proměnné `boot->sdm_empty_space` (viz úryvek kódu 4.1), která je na začátku funkce inicializována na hodnotu 0. Poté se její činnost řídí podle toho, kolik bytů dat je potřeba zapsat. Pokud je proměnná `len` >= WORD_WIDTH (velikost jednoho slova v bytech, tj. 4), zapíše se na příslušnou adresu celé slovo z bufferu, podle toho kde se nachází ukazatel `data`. Následně je ukazatel `data` posunut na další slovo (`data += 4`), délka bufferu `len` je zmenšena o 4 byty a pokud se jednalo o zápis do vstupní FIFO paměti, je dekrementována hodnota v `boot->sdm_empty_space`. Poté funkce opakuje svou činnost, dokud je ve vstupní FIFO paměti volné místo (boot->sdm_empty_space > 0), v případě že není místo, ale je stále co zapisovat, pokračuje funkce dalším čekáním na volnou kapacitu. Pokud počet bytů pro zápis v proměnné `len` klesne pod hodnotu WORD_WIDTH, ale je vetší než 0 (`len` může být menší než 4 hned při zápisu prvního slova), data jsou bitově doplněna nulami na šířku 4 bytů, odeslány na příslušnou adresu a funkce končí.

4.5.2 funkce `sdm_get_data`

Tato funkce se používá pro vyčítání odpovědí od SDM, které jsou k dispozici ve výstupní FIFO paměti komponenty Mailbox Client IP. K dispozici je jí přes parametr poskytnut buffer `data` (datový typ `u_char*`) o délce `len` bytů, do kterého jsou data funkcí zapisována (`data` může být NULL a `len` může být 0, je-li v odpovědi očekávána pouze hlavička). Funkce vrací různé hodnoty v závislosti na vyčtené odpovědi, a to:

- Počet přečtených bytů, pokud při čtení nenastala chyba.
- Hodnotu 0, pokud je odpověď tvořena pouze hlavičkou a vše proběhlo bez chyb.
- Zápornou hodnotu, pokud při čtení nastala chyba.

Funkce nejprve čeká, dokud není v registru ISR nastaven nejnižší bit do log. 1 (příznak, že jsou ve výstupní FIFO paměti validní data). Čekání je ošetřeno pomocí timeoutu, aby nemohlo dojít k zaseknutí stroje v případě nepřítomnosti dat pro vyčtení. Pokud je tento příznak nastaven, pokračuje funkce čekáním na zaplnění výstupní FIFO paměti a výskyt začátku paketu. V dalším kroku (je-li nastaven příznak začátku paketu a je-li výstupní FIFO alespoň z části zaplněno) je vyčtena hlavička odpovědi. Pokud neindikuje chybu, je z ní extrahována délka odpovědi (počet 4bytových slov odpovědi) a je dekrementováno zapamatované zaplnění výstupní FIFO paměti. Pokud odpověď nezahrnuje pouze hlavičku (zaplnění výstupní FIFO paměti je i po dekrementaci nenulové), pokračuje funkce čtením slov z výstupní FIFO paměti komponenty do bufferu `data` (až dokud nezbývá k vyčtení poslední slovo, případně čeká na znovunaplnění výstupní FIFO paměti). Souběžně s každým vyčteným slovem upravuje ukazatel `data` (`data += 4`), přičítá 4 byty k počtu vyčtených bytů a dekrementuje zapamatované zaplnění výstupní FIFO paměti a počet slov odpovědi získaný z hlavičky. Při vyčítání posledního slova čeká na nastavený příznak konce paketu. V případě nezarovnaného čtení je poslední slovo odpovídajícím způsobem zkráceno, aby nedošlo k přetečení bufferu `data`.

4.5.3 funkce `sdm_send_header`

Tato funkce se stará o sestavení hlavičky příkazu a o její odeslání na správnou adresu. Funkce dostává přes parametr operační kód `cmd_code` příkazu a jeho délku `cmd_len` (počet

4bytových slov). Funkce vrací 0, pokud nenastala v průběhu vykonávání žádná chyba, jinak zápornou hodnotu.

Funkce nejprve zkontroluje platný rozsah předaných parametrů. Následně pomocí bitových posunů sestaví z ID (`boot->sdm_cmd_id`), operačního kódu a délky příkazu hlavičku. Před samotným odesláním hlavičky je do kódu přidáno čekání (důvod je vysvětlen v sekci 5.3). Díky tomuto čekání postrádá význam použití ID pro rozlišení jednotlivých příkazů, protože před posláním dalšího příkazu jednotce se vždy počká na dokončení předchozího. Proto je zatím logika číslování jednotlivých příkazů v kódu zakomentována. Po uplynutí doby čekání následuje samotné zapsání hlavičky na adresu, která je určena podle toho, zda je příkaz tvořen pouze hlavičkou nebo zda za hlavičkou následují argumenty.

4.5.4 funkce `sdm_qspi_open_set_cs`

Tato funkce slouží pro získání exkluzivního přístupu k flash paměti a k vybrání výchozího QSPI zařízení. Jako parametr dostává ukazatel na strukturu `spi_nor`. Vrací 0, pokud nenastala v průběhu vykonávání žádná chyba, jinak zápornou hodnotu.

Funkce pro svou činnost používá výše jmenované pomocné funkce. Nejdříve je poslána hlavička pro příkaz QSPI_OPEN (operační kód = 0x32, délka = 0). Po vyčtení odpovědi je zaslána druhá hlavička pro příkaz QSPI_SET_CS (operační kód = 0x34, délka = 1). Jediným a tedy i posledním argumentem tohoto druhého příkazu je výchozí hodnota 0, která značí selektor pro připojené QSPI zařízení (viz sekce 3.3.5).

4.5.5 funkce `sdm_qspi_close`

Tato funkce slouží pro ukončení exkluzivního přístupu k flash paměti. Jako parametr dostává ukazatel na strukturu `spi_nor`. Vrací 0, pokud nenastala v průběhu vykonávání chyba, jinak zápornou hodnotu.

Funkce stejně jako předchozí jmenovaná využívá výše zmiňované pomocné funkce. De facto jedinou úlohou, kterou vykonává, je odeslání hlavičky pro příkaz QSPI_CLOSE (operační kód = 0x33, délka = 0).

4.6 Funkce rozhraní ovladače

Tyto funkce implementují rozhraní ovladače, které se mapuje na obslužné funkce pro práci s flash pamětí, a proto mají pevně daný prototyp. Pro svoji činnost využívají pomocné funkce popsané v podkapitole 4.5. Všem je pomocí parametru předáván ukazatel na strukturu `spi_nor`. Všechny také na začátku volají přípravnou funkci `sdm_qspi_open_set_cs` a na konci funkci `sdm_qspi_close`. Tato fakta nejsou u funkcí znovu explicitně uváděna.

4.6.1 funkce `sdm_qspi_read`

Tato funkce slouží pro čtení dat z QSPI flash paměti. Maximální počet slov pro vyčtení je 1024 (4kB). Jako parametry jsou jí předávány adresa (`from`), ze které se má ve flash paměti číst (tato adresa musí být zarovnaná na 4 byty), počet bytů k přečtení (`len`) a buffer, do kterého se mají vyčtená data zapsat (`buf`). Funkce vrací počet vyčtených bytů nebo zápornou hodnotu při chybě.

Funkce nejprve kontroluje zarovnání adresy na 4 byty. Pokud adresa není zarovnaná, je vrácena chyba. Po kontrole je poslána hlavička pro příkaz QSPI_READ (operační kód = 0x3A, délka = 2). Nyní je provedeno případné zarovnání na maximální počet čtených

slov (1024). V dalším kroku jsou zapsány argumenty. Prvním je adresa pro čtení ve flash paměti, druhým pak počet vyčítaných slov. Nakonec je volána funkce `sdm_get_data` pro přečtení odpovědi a zapsání vyčtených slov do bufferu.

4.6.2 funkce `sdm_qspi_read_reg`

Tato funkce slouží pro čtení registrů QSPI flash paměti. Maximální velikost čtených dat je 8 bytů. Jako parametry jsou jí předávány operační kód pro čtecí příkaz (`opcode`), buffer, do kterého mají být zapsána vyčtená data (`buf`) a počet bytů k přečtení (`len`). Funkce vrací 0, pokud nenastala v průběhu vykonávání chyba, jinak zápornou hodnotu.

Funkce nejprve zasílá hlavičku pro příkaz QSPI_READ_DEVICE_REG (operační kód = 0x35, délka = 2), poté jsou zapsány argumenty. Prvním argumentem je operační kód pro čtecí příkaz, druhým počet čtených bytů. Na závěr je vyčtena odpověď do bufferu. Pro práci na DevKitu bylo potřeba čtení registrů obejít jiným způsobem (viz sekci 5.3).

4.6.3 funkce `sdm_qspi_write`

Tato funkce slouží pro zápis dat do QSPI flash paměti. Maximální počet slov pro zapsání je 1024 (4kB). Jako parametry jsou jí předávány adresa (`to`), na kterou se mají data ve flash paměti zapsat (tato adresa musí být zarovnaná na 4 byty), počet bytů k zapsání (`len`) a buffer se zapisovanými daty (`buf`). Funkce vrací počet bytů, které se podařilo zapsat nebo zápornou hodnotu při chybě.

Funkce nejprve provede případné zarovnání na maximální počet zapisovaných slov (1024) a kontroluje zarovnání adresy na 4 byty. Pokud adresa není zarovnaná, je vrácena chyba. Po kontrole je poslána hlavička pro příkaz QSPI_WRITE (operační kód = 0x39, délka = 2+počet zapisovaných slov) následovaná argumenty. Prvním je adresa ve flash paměti, na kterou mají být data zapsána, druhým je počet zapisovaných slov a třetím jsou samotná zapisovaná data. Nakonec je zkontrolováno, zda se data povedlo zapsat.

4.6.4 funkce `sdm_qspi_write_reg`

Tato funkce slouží pro zápis do registrů QSPI flash paměti. Maximální velikost zapisovaných dat je 8 bytů. Jako parametry jsou jí předávány operační kód pro zapisovací příkaz (`opcode`), buffer s daty pro zapsání (`buf`) a počet bytů k zapsání (`len`). Funkce vrací 0, pokud nenastala v průběhu vykonávání chyba, jinak zápornou hodnotu.

Funkce nejprve zasílá hlavičku pro příkaz QSPI_WRITE_DEVICE_REG (operační kód = 0x36, délka = 2+počet zapisovaných slov), poté jsou zapsány argumenty. Prvním argumentem je operační kód pro zapisovací příkaz, druhým počet zapisovaných bytů a případným třetím samotná zapisovaná data. Nakonec je zkontrolováno, zda se data povedlo zapsat.

4.6.5 funkce `sdm_qspi_erase`

Tato funkce slouží pro mazání sektoru QSPI flash paměti. Příkaz QSPI_ERASE sice podporuje několik velikostí mazaných sektorů, funkce ale kvůli pěvně danému prototypu podporuje pouze maximální velikost 64kB. Přes parametr je jí předávána adresa (`off`), kde se má začít mazat (tato adresa musí být zarovnaná na 64 kB). Funkce vrací 0, pokud nenastala v průběhu vykonávání chyba, jinak zápornou hodnotu.

Funkce nejprve kontroluje zarovnání adresy na 64kB. Po kontrole zasílá hlavičku pro příkaz QSPI_ERASE (operační kód = 0x38, délka = 2) následovaný argumenty. Prvním je adresa ve flash paměti, odkud se má začít mazat, druhým je velikost sektoru (v tomto případě pevně stanovená na 64kB). Nakonec je kontrolován úspěch vykonávání příkazu.

Kapitola 5

Ověření funkčnosti

5.1 Omezení

Kvůli globálnímu nedostatku čipů, který v poslední době pociťuje většina technologických odvětví, se datum dodání nové 400GE karty (viz podkapitolu 3.1) několikrát posunovalo, takže otestování této práce na cílovém hardware v podobě této karty nebylo ve výsledku možné.

Pro vytvoření této práce byl aktivně používán prostředek Intel Stratix 10 GX Development Kit (viz podkapitolu 3.2), který ale vzhledem k tomu, že se jednalo o tzv. Engineering Sample, nepodporoval některé důležité funkce (včetně RSU ze sekce 3.4). V tomto případě software Intel Quartus Prime nepodporoval programování tohoto zařízení za použití více bitstreamů najednou, což bylo nutným předpokladem pro vytvoření počátečního RSU konfiguračního obrazu, resp. vytvoření layoutu v konfigurační flash paměti, který by RSU podporoval. S využitím pouze jednoho bitstreamu totiž neexistoval způsob, jak v konfigurační paměti vytvořit blok PCB tak, aby na něho bylo možné odkazovat z bloků Decision Firmware (viz podkapitolu 3.4.1), které je možné upravovat pouze skrze komerční software Intel Quartus Prime.

5.2 Přepsání továrního obrazu

Výše uvedená omezení vyústila v to, že byl do konfigurační paměti skrze software Intel Quartus Prime nahrán pouze tovární obraz (společně s bloky Decision Firmware, které na něho odkazují) ve formátu `.jic`. Funkčnost implementovaného software pak bylo možné ověřit pouze tím stylem, že skrze vlastní softwarovou implementaci byl tento tovární obraz (včetně Decision Firmware) přepsán vygenerovaným bitstreamem ve formátu `.rpd`. Pro zápis konfiguračního bitstreamu do slotu pro tovární obraz byl použit příkaz `"nfb-boot -w 0 archive.tar"`, kde `archive.tar` obsahuje soubor s bitstreamem ve formátu `.rpd` a soubor s Device Tree ve formátu `.dtb`. Po provedení zápisu byl stroj s kartou manuálně rebootován a pomocí nástroje `nfb-info`[1] pak bylo podle data vytvoření bitstreamu, kterým se zařízení při bootování konfigurovalo, zjišťováno, zda přepsání obrazu proběhlo úspěšně.

[1] `nfb-info` je další z uživatelských aplikací implementovaných v rámci software sdružení CESNET. Tato aplikace zprostředkovává základní informace o kartě.

5.3 Práce s registry QSPI paměti

Tato sekce rozebírá problém při čtení a zápisu některých používaných registrů QSPI flash konfigurační paměti na prostředku DevKit. Jednalo se o problém, kdy se nedařilo vyčítat smysluplné hodnoty z registrů a zároveň při zápisu do některých registrů zařízení odpovídalo s chybou. Mezi konkrétní příklady tohoto chování patří následující:

- Při čtení identifikačního Device ID flash paměti (operační kód 0x9E nebo 0x9F) při zavádění modulu pro ovladač odpovídalo SDM sice bez chyby, ale jako Device ID byly vyčteny samé nuly. Pro tuto situaci byl přidán do funkce `sdm_qspi_read_reg` kód zobrazený v úryvku 5.1, který nastavuje provizorní Device ID, pro která je následně přidána provizorní podpora do souboru `spi-nor.c`, aby bylo možné práci na prostředku DevKit otestovat.
- Při čtení stavového registru (operační kód 0x05) nebylo možné vyčíst v průběhu nahrávání konfigurace do flash paměti příznak WIP (Write in Progress Bit), který určuje, zda je ve flash paměti právě zpracováván nějaký příkaz. To vedlo k nestálému chování, kdy byl občas příkaz přepsán ještě před svým dokončením dalším příkazem a komunikace se zasekla. Jako řešení problému bylo do kódu ovladače (konkrétně do funkce `sdm_send_header`) přidáno čekání po určitou dobu, aby se zpracovávaný příkaz stihl provést celý a následující příkaz se začal provádět až po dokončení tohoto předchozího.
- Při zápisu příkazu pro povolení/zakázání zápisu do flash (operační kódy 0x06/0x04) odpovídal SDM s chybovým kódem. Toto chování ale pro samotný zápis konfigurace neznamenalo žádné komplikace.

```
sdm.c:
    ...
    #define DEVKIT_ES 1

    // Intel EPCQ-L Silicon IDs
    // JEDEC ID is replaced by Silicon ID from EPCQ-L device specification
    // (to differentiate between the flashes)
    #define SID_EPCQL256 0x19
    #define SID_EPCQL512 0x20
    #define SID_EPCQL1024 0x21
    ...

    int sdm_qspi_read_reg(..., u8 opcode, u8 *buf, ...) {
        ...
    #if DEVKIT_ES
        // DEVKIT in use does not respond with the device ID (returns zeros)
        if (opcode == 0x9e || opcode == 0x9f) {
            if (len < 3) {
                return -EINVAL;
            }
            buf[0] = 0;
            buf[1] = 0;
            buf[2] = SID_EPCQL1024;
```

```
            return 0;
        }
    #endif
        ...
    }

spi-nor.c:
    static const struct flash_info spi_nor_ids[] = {
        ...
        /* Intel EPCQ-L */
        // JEDEC ID is replaced by Silicon ID from EPCQ-L device
        // specification (to differentiate between the flashes)
            { "epcql256" , INFO(0x19, 0, 0x10000, 512 , 0) },
            { "epcql512" , INFO(0x20, 0, 0x10000, 1024, 0) },
            { "epcql1024", INFO(0x21, 0, 0x10000, 2048, 0) },
            ...
        }
```

Výpis 5.1: Řešení pro nefunkční čtení registrů QSPI flash paměti na přípravku DevKit

Kapitola 6

Závěr

Cílem této bakalářské práce bylo nastudování možností konfigurace FPGA čipů Intel Stratix 10 a Intel Agilex a následné navržení a implementace hardwarové jednotky pro komunikaci s konfigurační QSPI flash pamětí těchto čipů. Dalším cílem bylo navržení a implementování softwarového nástroje, který by umožnil skrze tuto jednotku zvolená FPGA příslušným způsobem konfigurovat. Na závěr bylo potřeba ověřit funkčnost tohoto nově implementovaného systému nahráním bitstreamu do konfigurační paměti vybraného čipu a provedením jeho rekonfigurace s využitím tohoto nově nahraného bitstreamu.

Po nastudování potřebných materiálů byla navržena a implementována nová hardwarová jednotka pro obsluhu konfigurování FPGA čipů Intel Stratix 10 a Intel Agilex podle schématu Active Serial. Tato jednotka nese název SDM Client a používá se pro přístup k externí QSPI flash konfigurační paměti prostřednictvím SDM. Implementovaná jednotka je napojená na MI sběrnici a skrze tuto sběrnici s ní komunikuje připravený softwarový nástroj. Software zasílá příkazy do komponenty SDM, které umožňují nahrání nového konfiguračního obrazu do zmiňované QSPI flash paměti. Dále byla v rámci bakalářské práce připravena softwarová podpora pro jednotku SDM Client rozšířením uživatelské aplikace `nfb_boot` a knihovny `libnfb`. Uvedený software byl rozšířen o možnost zpracování bitstreamu ve formátu `.rpd`. Dále byly implementovány nové nízkoúrovňové funkce pro ovladač, které se řídí definovaným protokolem pro práci s flash pamětí. Pro prototypování byla zvolena FPGA karta Intel Stratix 10 GX Development Kit ES. Jednotka SDM Client byla následně zapojena do připraveného designu pro FPGA a v rámci ověření její funkčnosti byla použita pro přepsání továrního obrazu v konfigurační QSPI flash paměti u zvolené karty. Zápis nového designu a rekonfigurace čipu Stratix 10 GX proběhla úspěšně.

Během řešení práce se vyskytlo několik problémů, které se podařilo obejít, a tak nezabránily ověření základní implementované funkcionality. Prvním problémem byla nefunkční podpora pro RSU na zvolené kartě. Tento problém se podařilo obejít tak, že namísto přidání nebo přepsání zvoleného aplikačního obrazu byl přepsán celý tovární obraz. Tento problém dále znemožnil softwarové vynucení rekonfigurace zařízení, tudíž pro nakonfigurování FPGA z nově zapsaného továrního obrazu bylo nutné manuálně restartovat stroj. Dalším problémem bylo nefunkční čtení a zápis registrů QSPI flash paměti. Tento problém byl obejit vložením čekání na dokončení předchozího příkazu do kódu ovladačů (aby nedocházelo k přepisování nezpracovaných příkazů a následné chybě při zápisu nové konfigurace do této flash paměti). Toto nefunkční čtení registrů znemožnilo vyčítání Device ID pro identifikaci flash paměti při zavádění modulu pro ovladač. Proto bylo nutné nastavit hodnotu Device ID ručně a přidat podporu pro tuto flash paměť do softwarového nástroje.

Na tuto bakalářskou práci je možné navázat např. zprovozněním a následným otestováním funkcionality na 400GE FPGA kartě, která bude dostupná v následujících měsících. Nová 400GE karta umožní dále ověřit možnost softwarového rebootování zařízení s využitím RSU. Dalším možným rozšířením této bakalářské práce by bylo přidání podpory pro dynamickou práci s aplikačními obrazy v konfigurační QSPI flash paměti. Takto rozšířený softwarový nástroj by umožňoval dynamické přidávání a odebírání aplikačních obrazů a odpovídajícím způsobem by upravoval ukazatele na jednotlivé obrazy v konfigurační paměti.

Literatura

[1] Blanchard, J. *QSPI NOR Flash Part 3 — The Quad SPI Protocol* [online]. Duben 2021 [cit. 2022-05-08]. Dostupné z: https://www.jblopen.com/qspi-nor-flash-part-3-the-quad-spi-protocol/.

[2] CESNET. *MI bus specification* [online]. 2021 [cit. 2022-05-06]. Dostupné z: https://cesnet.github.io/ofm/comp/mi_tools/readme.html.

[3] CESNET. *CESNET vyvinul ve spolupráci s francouzskou společností Reflex CES akcelerační kartu pro 400Gb sítě* [online]. Únor 2022 [cit. 2022-05-05]. Dostupné z: https://www.cesnet.cz/sdruzeni/zpravy/tiskove-zpravy/cesnet-vyvinul-ve-spolupraci-s-francouzskou%20-spolecnosti-reflex-ces-akceleracni-kartu-pro-400gb-site/.

[4] Chan, K. *What are the Differences of Single vs Dual vs Quad SPI?* [online]. Květen 2020 [cit. 2022-05-05]. Dostupné z: https://www.totalphase.com/blog/2020/05/what-are-the-differences-of-single-vs-dual-vs-quad-spi/.

[5] Corbet, J., Rubini, A. a Kroah Hartman, G. *Linux Device Drivers*. 3. O'Reilly, 2005. ISBN 0-596-00590-3.

[6] Intel. *EPCQ-L Serial Configuration Devices Datasheet* [online]. Květen 2018 [cit. 2022-05-05]. Dostupné z: https://www.intel.com/content/www/us/en/docs/programmable/683710/current/epcq-l-serial-configuration-devices.html.

[7] Intel. *Intel Stratix 10 GX FPGA Development Kit User Guide* [online]. Duben 2020 [cit. 2022-05-05]. Dostupné z: https://www.intel.com/content/www/us/en/docs/programmable/683674/current/overview.html.

[8] Intel. *Avalon Interface Specifications* [online]. Leden 2022 [cit. 2022-05-05]. Dostupné z: https://www.intel.com/content/www/us/en/docs/programmable/683091/20-1/introduction-to-the-interface-specifications.html.

[9] Intel. *Intel Agilex Configuration User Guide* [online]. Duben 2022 [cit. 2022-05-05]. Dostupné z: https://www.intel.com/content/www/us/en/docs/programmable/683673/22-1/configuration-user-guide.html.

[10] Intel. *Intel Stratix 10 Configuration User Guide* [online]. Duben 2022 [cit. 2022-05-05]. Dostupné z: https://www.intel.com/content/www/us/en/docs/programmable/683762/22-1/configuration-user-guide.html.

[11] Intel. *Mailbox Client Intel FPGA IP User Guide* [online]. Duben 2022 [cit. 2022-05-05]. Dostupné z: https://www.intel.com/content/www/us/en/docs/programmable/683290/22-1-20-1-2/mailbox-client-fpga-ip-user-guide.html.

[12] Silberschatz, A., Galvin, P. B. a Gagne, G. *Operating System Concepts*. 10. Wiley, 2018. ISBN 978-1-119-32091-3.

Příloha A

Device Tree - implementace

V této příloze jsou uvedeny zdrojové kódy pro zapojení jednotky SDM Client do Device Tree designu pro FPGA a soubor reprezentující finální zapojení vybraných komponent, který je jedním z výstupů sestavování designu. Sestavování Device Tree je realizováno pomocí tcl skriptů, kdy k proměnné `ret` je postupně připojován další obsah, dokud nejsou analyzovány všechny relevantní komponenty zapojené v designu. To, které komponenty jsou v designu použité, je dohledáváno při překladu podle hierarchie souborů `Modules.tcl`[1], který u sebe ve složce má každá hardwarová komponenta a který podává informaci o podkomponentách zapojených v dané komponentě.

```
# 1. base - base address on MI bus
proc dts_sdm_controller {base} {
        set ret ""
        append ret "intel_sdm_controller {"
        append ret "compatible = \"netcope,intel_sdm_controller\";"
        append ret "reg = <$base 44>;"
        append ret "type = <0>;"
        append ret "version = <0x00000001>;"
        append ret "};"
        return $ret
}
```

Výpis A.1: Soubor DevTree.tcl pro jednotku SDM Client

```
...
# Populate main components
    append ret "boot:" [dts_sdm_controller $BOOT_ADDR]
...
```

Výpis A.2: Zapojení této jednotky do DevTree.tcl pro celé FPGA

[1]Soubor `Modules.tcl` pro komponentu SDM Client je k nahlédnutí na adrese `https://github.com/CESNET/ofm/blob/main/comp/ctrls/sdm_client/Modules.tcl`

```
/dts-v1/;

/ {

        firmware {
                ...

                mi0: mi_bus {
                        #address-cells = <0x01>;
                        #size-cells = <0x01>;
                        compatible = "netcope,bus,mi";
                        resource = "PCI0,BAR0";
                        width = <0x20>;

                        boot: intel_sdm_controller {
                                compatible = "netcope,intel_sdm_controller";
                                reg = <0x2000 0x2c>;
                                type = <0x00>;
                                version = <0x01>;
                        };

                        ...
                };
        };
};
```

Výpis A.3: Textová forma výsledného souboru obsahujícího Device Tree po překladu designu ve formátu `.dts` (programu `nfb-boot` je předávána jeho binární podoba ve formátu `.dtb`)

Příloha B

Přidání podpory pro formát bitstreamu `.rpd` do `nfb-boot`

Jedná se o přidání opakovaného vyhledání a přečtení souboru obsahujícího konfigurační data v případě, že není nalezen soubor s daty v jiném formátu. Přiložený kus kódu se vztahuje k funkci `do_write_with_dev`[1].

```
...
size = archive_read_first_file_with_extension(filename, ".bit", &fdata);
if (size < 0) {
        size = archive_read_first_file_with_extension(filename, ".rpd", &fdata);
}

if (fdata != NULL) {
        fd = fmemopen(fdata, size, "rb");
} else {
        fd = fopen(filename, "r");
}

if (fd == NULL) {
        warnx("failed open firmware file");
        ret = ENOENT;
        goto err_fopen;
}

size = nfb_fw_read_for_dev(dev, fd, &data);
...
```

Výpis B.1: Rozhodování o typu bitstreamu ve funkci `do_write_with_dev`

[1]Zdrojový kód stávající aplikace se nachází na adrese `https://github.com/CESNET/ndk-sw/blob/e035cb19f892b7b346bb523322628489e3fa7430/tools/boot/boot.c#L223`

Příloha C

Přidání podpory pro formát bitstreamu `.rpd` do `libnfb`

Pro potřeby této práce jsou důležité 3 soubory – `boot.c`, `boot.h` a `filetype_bit.c` – všechny ve složce `libnfb/src/boot`.

```
enum bitstream_format {
    ...
    BITSTREAM_FORMAT_INTEL_AS,
};
```

Výpis C.1: `boot.h` – přidání nového formátu bitstreamu do stávajícího výčtu podporovaných formátů

```
...
fdt_for_each_compatible_node(fdt, node, "netcope,boot_controller") {
    ...
}

fdt_for_each_compatible_node(fdt, node, "netcope,intel_sdm_controller") {
        format = BITSTREAM_FORMAT_INTEL_AS;
}
...
```

Výpis C.2: `boot.c` – nastavení formátu při nalezení jednotky SDM Client v designu (funkce `nfb_fw_read_for_dev`)

```
ssize_t nfb_fw_open_bit(FILE *fd, void **pdata, enum bitstream_format f)
{
	ssize_t ret;
	ssize_t i;
	uint8_t *data8;

	if (f == BITSTREAM_FORMAT_INTEL_AS) {
		ret = nfb_fw_open_rpd_raw(fd, pdata);
		if (ret > 0) {
			data8 = (uint8_t*)(*pdata);
			for (i = 0; i < ret; i++) {
				*data8 = bitReverseTable256[*data8];
				data8++;
			}
		}
	} else {
		ret = nfb_fw_open_bit_raw(fd, pdata);
		if (ret > 0 && f != BITSTREAM_FORMAT_SPI4)
			nfb_fw_bitstream_reverse_bits_16(*pdata, ret);
	}
	return ret;
}
```

Výpis C.3: `filetype_bit.c` – funkce `nfb_fw_open_rpd_raw` alokuje paměť a nahrává do ní bitstream ze souboru. Nad touto pamětí (dostupnou přes ukazatel `pdata`) je poté provedena bitová reverzace zmiňovaná v sekci 4.3

Příloha D

Příklad implementace nízkoúrovňové funkce pro ovladač

```
ssize_t sdm_qspi_write(struct spi_nor *nor, loff_t to, size_t len, \
        const u_char *buf) {
    uint32_t words_len;
    uint32_t uto = (uint32_t)to;
    u_char *last_word;
    uint32_t last_word_bytes;
    ssize_t ret;

    if (sdm_qspi_open_set_cs(nor) == RET_ERR) { return RET_ERR; }

    // align data length to words
    words_len = (len & (WORD_WIDTH-1))? (len>>2)+1 : len>>2;
    // max number of words
    if (words_len > MAX_WORDS) {
        words_len = MAX_WORDS;
    }

    last_word_bytes = (len & (WORD_WIDTH-1))? \
        (len & (WORD_WIDTH-1)) : WORD_WIDTH;

    // error for not correctly aligned address
    if (to & (WORD_WIDTH-1)) { return RET_ERR; }
    if (sdm_send_header(BOOT, SDM_QSPI_WRITE_OP, 2+words_len) \
        == RET_ERR) { return RET_ERR; }

    // extracting last word of data
    last_word = (u_char *)buf + (words_len-1)*WORD_WIDTH;

    // 1. argument = flash address offset to start writing (word aligned)
    sdm_send_data(BOOT, SDM_MC_CMD_FIFO, (u_char *)&uto, WORD_WIDTH);
    // 2. argument = number of words to write
    sdm_send_data(BOOT, SDM_MC_CMD_FIFO, (u_char *)&words_len, WORD_WIDTH);
```

```
    // 3. argument = data to be written (except last word)
    sdm_send_data(BOOT, SDM_MC_CMD_FIFO, (u_char *)buf, \
        len - last_word_bytes);
    // last word of data
    sdm_send_data(BOOT, SDM_MC_CMD_LAST_WORD, last_word , last_word_bytes);

    ret = (sdm_get_data(BOOT, NULL, 0) == RET_ERR)? RET_ERR : len;

    if (sdm_qspi_close(nor) == RET_ERR) { return RET_ERR; }

    return ret;
}
```

Výpis D.1: Implementace funkce sdm_qspi_write. `last_word` a `last_word_bytes` slouží v kódu pro nalezení posledního slova zapisovaných dat (i nezarovnaného), aby bylo možné jej zapsat odděleně na adresu pro poslední slovo příkazu, jak to vyžaduje protokol pro práci s Mailbox Client IP.