VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

RADAROVÝ SIGNÁLOVÝ PROCESOR V FPGA

RADAR SIGNAL PROCESSOR IN FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JAN PŘÍVARA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. LUKÁŠ MARŠÍK**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Přívara Jan, Bc.**
Obor: Počítačové a vestavěné systémy
Téma: **Radarový signálový procesor v FPGA**
Radar Signal Processor in FPGA
Kategorie: Návrh číslicových systémů

Pokyny:

1. Prostudujte dostupnou literaturu týkající se Dopplerova radaru a použití FPGA pro zpracování signálu.
2. Seznamte se s principy funkce radarů a s principy programování FPGA včetně dostupných vývojových nástrojů.
3. Navrhněte detektor řešící předem zvolenou úlohu a vytvořte sadu vhodných testů za účelem ověření finální implementace.
4. Implementujte navrženou jednotku v FPGA.
5. Proveďte sérii testů na poskytnutých datech.
6. Diskutujte dosažené výsledky a navrhněte možné pokračování práce.

Literatura:

- M. Skolnik: Radar Handbook, 3rd edition, McGraw-Hill Professional, 2008
- M. Skolnik: Introduction to Radar Systems, McGraw-Hill Science, 3rd edition, 2002
- M. A. Richards: Fundamentals of Radar Signal Processing, 1st edition, McGraw-Hill, 2005
- B. R. Mahafza: Radar Signal Analysis and Processing Using MATLAB, Chapman and Hall, 2008

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Maršík Lukáš, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Práce se zabývá návrhem a realizací radarového procesoru v FPGA. Teoretická část se věnuje Dopplerově radaru, principům zpracování radarového signálu a cílové platformě Xilinx Zynq. Následně je popsán návrh radarového procesoru včetně jednotlivých komponent a řešení je implementováno. Komponenty pro FPGA jsou popsány v jazyce VHDL. V poslední části je provedeno vyhodnocení implementace, jsou shrnuty poznatky z práce a je navrženo možné pokračování.

Abstract

This work describes design and implementation of radar processor in FPGA. The theoretical part is focused on Doppler radar, principles of radar signal processing methods and target platform Xilinx Zynq. The next part describes design of radar processor including its individual components and the solution is implemented. FPGA components are written in VHDL language. In the end, the implementation is evaluated and possible continuation of this work is stated.

Klíčová slova

zpracování radarového signálu, radarový procesor, Dopplerův radar, Dopplerův efekt, diskrétní Fourierova transformace, rychlá Fourierova transformace, vestavěné systémy, hardwarová akcelerace, FPGA, Zynq

Keywords

radar signal processing, radar processor, Doppler radar, Doppler effect, discrete Fourier transform, fast Fourier transform, fft, embedded systems, hardware acceleration, FPGA, Zynq

Citace

PŘÍVARA, Jan. *Radarový signálový procesor v FPGA*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Maršík Lukáš.

Radarový signálový procesor v FPGA

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Lukáše Maršíka. Další informace mi poskytli Ing. Martin Musil a Ing. Petr Musil. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Přívara
15. května 2017

Poděkování

Děkuji vedoucímu práce, Ing. Lukáši Maršíkovi, za vedení mé práce, návrhy, rady a podněty, které mi pomohly k hlubšímu pochopení problematiky radarů, zpracování radarových signálů a ke zkvalitnění této práce. Dále děkuji Ing. Martinu Musilovi a Ing. Petru Musilovi za jejich ochotu a pomoc při vývoji na platformě Zynq.

Obsah

Kapitola 1

Úvod

Radary se s postupem času dostaly do spousty civilních aplikací. Jejich velkou výhodou je dobrá spolehlivost ve zhoršených světelných podmínkách (tma, déšť, mlha), schopnost přesného měření rychlosti a vzdálenosti, a v neposlední řadě také jejich nízká cena. Příkladem je využití této technologie pro meteorologii, meření rychlosti vozidel a pro klasifikaci objektů.

Zejména radary s kontinuální vlnou jsou velmi dostupné a stále nacházejí uplatnění v nových aplikacích. Přestože využití radaru může být různé, zpracování radarových dat je do určité fáze téměř vždy totožné, nebo se liší pouze nepatrně. Dalším společným rysem radarových systémů jsou často nároky na kompaktnost výsledného zařízení, vysoký výpočetní výkon a v poslední době také na spotřebu. Odpovědí na tyto požadavky jsou tzv. SoC (*System On Chip*) zařízení, které kombinují klasický univerzální procesor a programovatelnou logiku FPGA v rámci jednoho čipu. Díky tomuto úzkému propojení obou částí nabízejí dosud nevídané možnosti a jejich pontenciál již byl úspěšně využit v řadě aplikací pro zpracování videí, počítačové vidění, rádiové komunikace a další.

Cílem této práce je vytvořit radarový procesor v FPGA na platformě Zynq, který bude provádět základní zpracování radarových dat. Aplikace využívající radar tedy nebude muset implementovat nezbytné výpočty pro extrakci užitečných dat z radarového signálu, ale s výhodou bude moci využít již hotový a otestovaný modul. Výsledné řešení by mělo napomoci ke snížení nároků na dobu vývoje a výsledné ceny takové aplikace. Realizace procesoru v FPGA by měla umožnit dosažení vysokého výkonu a efektivity, přičemž výpočetní výkon univerzálního procesoru na Zynq bude aplikace moci využít libovolným způsobem. Jinou možností bude předřazení radarového procesoru jiné jednotce na FPGA, která může provádět další zpracování, které by na procesoru nebylo vhodné nebo dostatečně výkonné.

Motivací k této práci pro mne bylo především prohloubení znalostí a zkušeností s vývojem na FPGA a možnost přístupu k zajímavé platformě Xilinx Zynq.

Kapitola 2 shrnuje základní informace o radarech. Následující kapitola 3 se zabývá technikami zpracování radarového signálu a extrakcí klíčových informací. Kapitola 4 představuje platformu Zynq a popisuje způsob vývoje na této platformě. V kapitole 5 je navržen radarový procesor a jsou uvedeny testy pro ověření finální implementace. Kapitola 6 popisuje způsob implementace jednotlivých částí a kapitola 7 vyhodnocuje vlastnosti řešení z několika hledisek. V závěrečné kapitole 8 je provedeno shrnutí práce a je diskutováno její možné pokračování.

Kapitola 2

Radar

Radar (*Radio Detection And Ranging*) je zařízení, jehož primárním účelem je detekce objektů a měření vzdálenosti. Princip fungování spočívá ve vysílání elektromagnetického záření, které je objekty částečně pohlceno a částečně rozptýleno odrazem. Část rozptýleného záření je odražena zpět k radaru, kde je zachycena a zpracována. Právě díky analýze odraženého signálu je radar schopen poskytnout určité informace o daném objektu. Díky znalosti rychlosti šíření elektromagnetických vln v daném prostředí a znalosti doby mezi vysláním signálu a přijetím jeho odrazu je možné přesně stanovit vzdálenost objektu. Pro určení rychlosti objektu se využívá *Dopplerova jevu*, který popisuje závislost frekvence přijatého signálu na základě rychlosti objektu vzhledem k radaru a původní frekvence vyslaného signálu.

Díky technologickému rozvoji je dnes na trhu dostupné velké množství radarových modulů i kompletních radarových systémů určených pro nejrůznější aplikace. Lze vybírat na základě fyzických rozměrů, vyzařovaného výkonu, energetické spotřeby a dalších parametrů. Jedním z nejčastějších použití je monitorování provozu na silničních komunikacích a měření rychlosti vozidel. Z hlediska charakteristiky vysílaného signálu můžeme radary rozdělit do následujích kategorií:

- **Pulzní radary** - Signál má podobu periodicky vysílaných pulzů, mezi kterými jsou radarem zachytávány jejich odrazy. Dochází tak k neustálému přepínání radaru mezi režimem pro příjem a pro vysílání, díky čemuž lze použít pouze jednu sdílenou anténu. Je podporováno současné měření rychlosti i vzdálenosti, jelikož je možné jednoduše určit zpoždění mezi vysláním pulzu a přijetím jeho odrazu. Na druhou stranu je vyžadována složitější elektronika a díky pulznímu charakteru signálu také složitější metody zpracování. V případě leteckých radarů se používá parabolická anténa vyzařující signál ve tvaru velmi úzkého kužele. Díky směrování antény je tak možné kromě rychlosti a vzdálenosti určit také úhel, pod kterým se objekt nachází. Nejmodernější typy radarů využívají namísto pohyblivé směrové antény vychylování vyzařovaného paprsku pomocí interferencí dílčích signálů, které jsou vysílány vysokým počtem malých fixních modulů rozmístěných ve tvaru mříže.

- **Radary s kontinuální vlnou** - Jedná se o radary, které namísto přerušovaných pulzů vysílají kontinuální signál. Díky tomu je umožněna jednodušší fyzická realizace, menší rozměry a nižší cena takového radaru. Nevýhodou je to, že nelze jednoduchým způsobem měřit vzdálenost jako v případě pulzních radarů. Pro měření vzdálenosti je nutné využít některou z metod frekvenční modulace. Jelikož příjem signálu probíhá současně s vysíláním, tak je radar typicky vybaven dvěma anténami. Tento typ radarů

je obvykle používán na krátší vzdálenosti, pro které není nutné vysílat signál o vysokém výkonu. Jejich nejčastějším využitím je již zmíněné měření rychlosti vozidel a monitoring silničního provozu. V práci [14] byla zkoumána a prokázána schopnost klasifikace typů vozidel tímto typem radaru.

2.1 Dopplerův jev

Dopplerův jev popisuje relativní vnímání frekvence přijatého signálu ze zdroje, který se vůči pozorovateli pohybuje. Vnímaná frekvence pak závisí na rychlosti objektu vzhledem k pozorovateli a původní frekvenci vyslaného signálu. Pro přijatou frekvenci platí následující vztah:

$$f = (\frac{c + v_r}{c + v_s}) f_0 \tag{2.1}$$

kde f je pozorovaná frekvence, f_0 je frekvence generovaná vysílačem, c je rychlost šíření vln v daném médiu, v_r je rychlost přijímače a v_s je rychlost vysílače.

Je možné pozorovat, že pokud se vysílač a přijímač k sobě přibližují, bude vnímaná frekvence vyšší než frekvence generovaná a naopak. Předpokládá se, že vzájemná rychlost objektů je nižší než rychlost šíření vln v daném médiu. Vzájemná rychlost odpovídá součtu absolutní rychlosti přijímače a vysílače v daném prostředí pouze v případě, že se pohybují přímo proti sobě, resp. od sebe. V jiných případech se vzájemná rychlost snižuje podle velikosti úhlu, které mezi sebou vektory rychlostí zdroje a přijímače svírají.

V praxi radar funguje jako vysílač i jako přijímač. Vyslaný signál je tedy nejprve přijat objektem, přičemž už v této fázi dochází ke vzniku Dopplerova jevu. Část signálu je objektem odražena zpět do radaru, kde se při jeho zachycení opět projeví Dopplerův efekt. Vzorec 2.1 je možné zjednodušit za předpokladu, že rychlost šíření vln v daném médiu je mnohonásobně větší, než vzájemná rychlost zdroje a přijímače a vlnová délka signálu je mnohonásobně menší, než vzdálenost mezi zdrojem a přijímačem. To v případě použití radarů platí a je tak možné aplikovat vzorec, který popisuje změnu frekvence jako [15]:

$$f_d = \frac{2v_r}{\lambda} \tag{2.2}$$

kde f_d je rozdíl mezi generovanou a přijatou frekvencí, v_r je vzájemná rychlost (kladná, pokud se objekt přibližuje k radaru) a λ je vlnová délka signálu.

Pokud byl objekt měřen pod nenulovým úhlem, tj. nepohyboval se přímo proti radaru nebo od něj, nebude vypočtená rychlost objektu odpovídat skutečné hodnotě. V takovém případě je možné provést korekci pomocí *kosinového faktoru* [11]:

$$v_m = v_r \cdot \cos \alpha \tag{2.3}$$

kde v_m je změřená rychlost objektu, v_r je reálná rychlost objektu a α je úhel směru pohybu objektu vzhledem k radaru.

2.2 Šíření radarového signálu

Při šíření radarového signálu je třeba brát v úvahu pohlcování a odrazy signálu od objektů, použitou frekvenci, vysílací výkon, a další vlivy. Důležitou roli hraje i samotné prostředí,

kterým signál putuje. Bylo zjištěno, že vodní pára je schopna znatelně tlumit signály s frekvencí 22.24 GHz a kyslík signály s frekvencí 60 GHz [16]. Nelze opomenout, že v civilním sektoru radary podléhají legislativním omezením, které upravují povolené frekvence i maximální vyzařovaný výkon. Výkon signálu přijatého radarem je popsán následovně [8]:

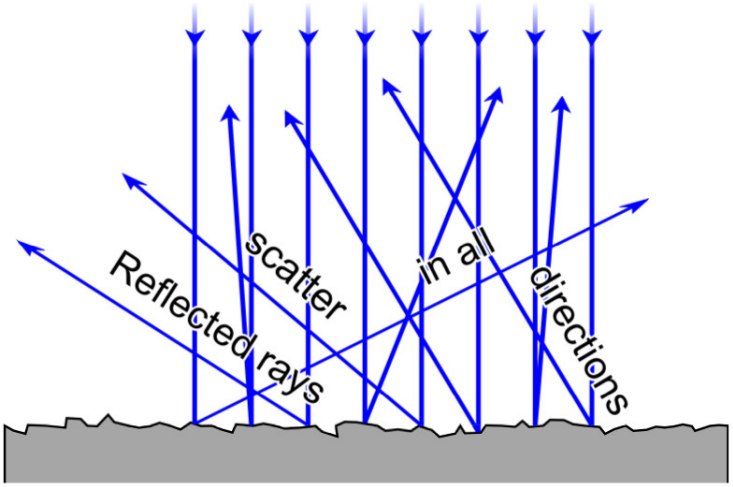
$$P_r = \frac{P_t A_{er} A_{et} \sigma}{4\pi R^4 \lambda^2 L_{sys}} \tag{2.4}$$

kde P_r je výkon přijatého signálu, P_t je výkon vyzařovaného signálu, A_{er} a A_{et} je efektivní plocha vstupní a výstupní antény, R je vzdálenost objektu, σ odpovídá efektivní odrazové ploše objektu (RCS – *Radar Cross Section*), λ je použitá vlnová délka a L_{sys} jsou ztráty systému, vzniklé na např. na anténách a v přenosovém médiu.

Odrazy od objektů

Jak již bylo řečeno, při dopadu radarového signálu na objekt je část pohlcena, další část je rozptýlena odrazem a zbytek může objektem prostoupit. To v jakém poměru bude signál rozložen do těchto částí je určeno frekvencí signálu, rozměry a tvarem objektu a materiálem, ze kterého je objekt tvořen. Kovy jsou materiály s největší reflexivitou elektromagnetického signálu.

Povrch objektů je tvořen počtem malých plošek s různou orientací, která určuje směr odrazu radarového signálu. I zdánlivě rovná plocha tak obsahuje nerovnosti, které odrážejí signál různými směry. Díky tomuto jevu, známému jako *difůzní odraz*, je možné radarem detekovat a měřit plochy, které nejsou kolmé vzhledem k radaru [12].



Obrázek 2.1: Odrazy signálu od nerovností povrchu. Převzato z [12].

Frekvence signálu

Frekvence signálu, kterou radar používá, je dalším důležitým aspektem ovlivňujícím způsob jeho šíření prostředím. Používají se frekvence řádově od stovek megahertz až po desítky gigahertz. Zjednodušeně by se dalo říct, že čím je frekvence použitého signálu vyšší, tím více

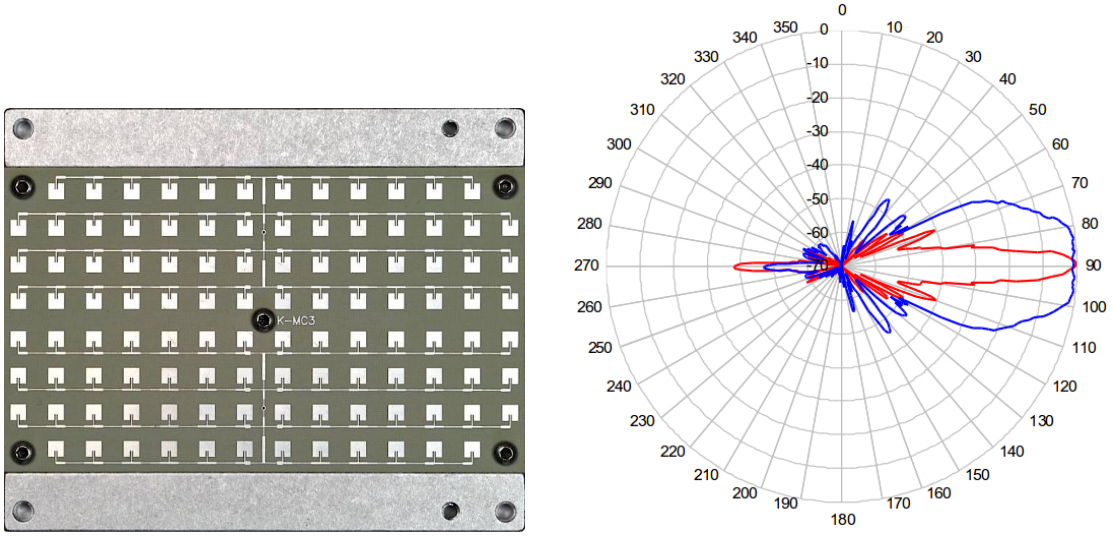
se bude signál chovat jako viditelné světlo. Tedy bude více pohlcován a odrážen objekty namísto toho, aby jimi procházel, bude více ovlivněn povětrnostními vlivy atd. Rozlišovací schopnost signálu s vyšší frekvencí je oproti nízkofrekvenčnímu signálu vyšší. Signály s nižší frekvencí naopak dosahují lepší propustnosti prostředím a jsou tedy vhodnější pro velké vzdálenosti. Současně platí, že nároky na fyzickou velikost antény jsou přímo úměrné použité vlnové délce. Tedy pro radary s vyšší frekvencí může být použita menší anténa, což je výhodné především pro radary ve vestavěných a přenosných systémech. Důvodem použití nízkých frekvencí v historii bylo částečně to, že technologie dané doby neumožnovala konstrukci vysokofrekvenčních radarů [17].

Zajímavým typem radarů, které potřebují současně dobrou prostupnost materiálem i odrazivost, jsou přenosné radary schopné detekce a lokace osob za zdí nebo jinou pevnou překážkou. Jde převážně o radary určené do policejního a vojenského prostředí, které využívají frekvence od 3 GHz do 10 GHz a jejich rozlišovací schopnost je do 15 cm [4].

2.3 Radar s kontinuální vlnou

Radar s kontinuální vlnou se vyznačuje vysíláním nepřerušovaného signálu se současným příjmem radarového odrazu. Měření rychlosti objektu probíhá stejným způsobem jako v případě pulzních radarů – tj. na základě analýzy frekvence přijatého signálu a dosazení do vzorce Dopplerova jevu (2.2). Rozdíl je však ve způsobu měření vzdálenosti. Pulzní radary jsou na měření vzdálenosti uzpůsobeny přímo pulzní charakteristikou vysílaného signálu, což umožňuje jednoduše měřit dobu mezi vysláním pulzu a přijetím jeho odrazu. U radarů s kontinuální vlnou nelze sledovat tuto dobu jinak než za pomocí některé z metod frekvenční modulace. Frekvenční modulace umožňuje do vysílaného signálu vložit časové značky, díky kterým může být určen posun (a tedy i zpoždení) oproti přijatému odrazu.

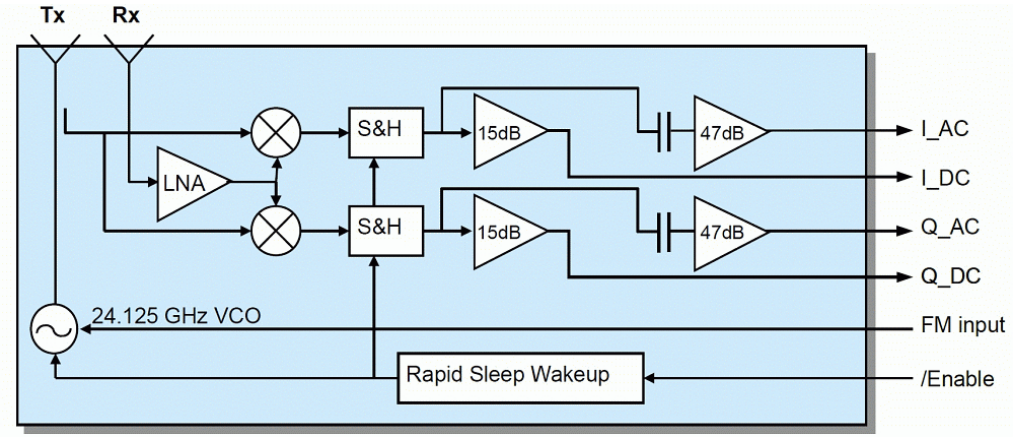
Na trhu je několik radarových modulů spadajících do této kategorie. Mezi výrobce patří firmy jako RFbeam Microwave GmbH, ViaSat, Sivers IMA, InnoSenT a další. Je možné vybírat od nejjednodušších modelů určených např. pro automatické otevíraní dveří až po takové, které umožňují provádět přesná měření na několik desítek až stovek metrů, včetně možnosti použití frekvenční modulace. Primárním určením pokročilejších modelů je často použití v automobilové dopravě. Frekvence je díky legislativním omezením pro civilní sektor nejčastěji 24 GHz. Tato frekvence dovoluje použít antény o velmi malých rozměrech. Toho se s výhodou využívá a antény tak mají nečastěji podobu tištěného pole malých plošek na přední straně modulu (obrázek 2.2 vlevo), což výrazně snižuje výrobní náklady i celkové fyzické rozměry.



Obrázek 2.2: Čelní pohled na radarový modul K-MC3 a jeho anténu (vlevo). Na pravé straně je zobrazen sférický diagram popisující směrovou charakteristiku této antény. Převzato z [2]

Návrh antény ovlivňuje mimo jiné směrovou charakteristiku vysílaného signálu. Příklad takové charakteristiky je možné vidět na obrázku 2.2 vpravo. V případě nasměrování většiny vyzařované energie do prostoru formou úzkého kuželu se zvýší efektivní dosah radaru, ale vysílaný signál pak pokryje menší oblast.

Schéma radarového modulu



Obrázek 2.3: Vnitřní schéma radarového modulu K-MC3. Převzato z [2]

Požadovaným výstupem modulu je signál, jehož frekvence odpovídá rozdílu frekvence výstupního a přijímaného signálu (ovlivněného Dopplerovým jevem). Vstupní signál je ihned po jeho zachycení anténou zesílen, protože energie tohoto signálu je velice malá a bez zesílení by nebylo možné signál použít (2.4). Pro určení Dopplerovy frekvence je prováděno

směšování výstupního a zesíleného vstupního signálu [17]:

$$\begin{aligned} &\cos(2\pi f_{rx}t) \cdot \cos(2\pi f_{tx}t) = \\ &= \tfrac{1}{2}[\cos(2\pi(f_{rx} + f_{tx})t) + \cos(2\pi(f_{rx} - f_{tx})t)] \end{aligned} \tag{2.5}$$

kde f_{rx} je přijímaná frekvence a f_{tx} je vysílaná frekvence.

Výsledkem násobení jsou dva periodické signály, jejichž frekvence odpovídají součtu a rozdílu vstupní a výstupní frekvence. Signál $\frac{1}{2}\cos(2\pi(f_{rx} + f_{tx})t)$ bývá rovný přibližně dvojnásobku výstupní frekvence a je odfiltrován přímo elektronikou radaru [15]. Výstupem je tedy periodický signál s frekvencí danou rozdílem $f_{rx} - f_{tx}$. Objekty s nulovou rychlostí vůči radaru nebudou proto na výstupu generovat žádné periodické signály. Funkce cosinus je sudá a není tedy možné z jejího výstupu rozeznat, zda má kladnou či zápornou frekvenci (zda se objekt pohyboval směrem k radaru nebo od radaru). Radar je proto vybaven komplexním výstupem s kanály I (*Inphase*) a Q (*Quadrature*), mezi kterými je fázový posun $\frac{\pi}{2}$ v případě pohybu k radaru a $-\frac{\pi}{2}$ v případě pohybu od radaru. Tento odstavec byl převzat z [14].

Po tomto kroku následuje ještě další zesílení, přičemž na výstupech modulu může být podle modelu zesílená i nezesílená varianta signálu. Vzhledem k tomu, že spousta vestavěných systémů je napájena z baterie, bývají některé radarové moduly vybaveny také jednotkou umožňující vypínat klíčové prvky radaru a šetřit tak enerii ve chvílích, kdy radaru není zapotřebí.

Kapitola 3

Zpracování radarového signálu

Analogový výstup Dopplerova radaru v sobě nese mnoho informací, které ale nejsou v časové doméně na první pohled patrné. Při digitálním zpracování je nevyhnutelným prvním krokem A/D převod signálu. Rychlost objektů je reprezentována frekvencí signálu. Situaci komplikuje fakt, že směr pohybu objektu (od radaru nebo k radaru) je určen fázovým posunem mezi kanály I a Q. Pokud se v zorném poli radaru nachází více objektů, analýza signálu v časové oblasti se stává ještě o poznání složitější.

Díky tomu je daleko vhodnější signál zpracovat ve frekvenční oblasti. Pro výpočet spektra signálu existuje více metod, nejznámější a nejpoužívanější je však diskrétní Fourierova transformace (DFT), resp. rychlá Fourierova transformace (FFT), která je výpočetně efektivnější. Jednou z výhod DFT je to, že díky její komplexní povaze je možné na její vstup přivést současně kanály I a Q, čímž odpadne nutnost zpracovávat tyto kanály samostatně a porovnávat jejich fáze za účelem zjištění směru pohybu objektu.

Ze spektrogramu je následně možné extrahovat požadované informace. Je nutné počítat s tím, že signál bude z různých důvodů obsahovat kromě užitečné informace také šum. Z hlediska kvality signálu je proto důležitý poměr intenzity signálu a šumu (SNR – *Signal To Noise Ratio*). V některých aplikacích radarová data obsahují kromě užitečných segmentů signálu, kdy se před radarem nacházel sledovaný objekt, také velké množství úseků, kdy se před radarem nic nenacházelo. Jednou z technik pro identifikaci užitečných segmentů v signálu je prahování, které sleduje překročení zadané hodnoty energie na jednotlivých frekvenčních složkách signálu. Potom je možné spočíst rychlost, vzdálenost nebo např. efektivní odrazovou plochu detekovaného objektu. Interpolace a vyhlazování měřené veličiny se také mohou ukázat jako vhodné techniky, protože některé části radarového signálu nemusí být použitelné.

3.1 A/D převod signálu

S procesem převodu analogového signálu na digitální souvisí *vzorkování* a *kvantizace*. Vzorkování spočívá v periodickém měření spojitého analogového signálu. Vzorkovací perioda udává dobu mezi jednotlivými měřeními. Platí, že vzorkovací perioda, resp. vzorkovací frekvence, musí splňovat Nyquistův teorém [10]:

$$F_s \geq 2F_{max} \tag{3.1}$$

kde F_s je vzorkovací frekvence a F_{max} je nejvyšší frekvence vyskytovaná ve vzorkovaném signálu. Pokud je vzorkovací perioda nižší, bude docházet k *aliasingu*, což je nežádoucí jev

znemožňující bezchybnou rekonstrukci navzorkovaného signálu zpět na signál analogový. V případech, kdy není možné zajistit dostatečnou vzorkovací frekvenci, je možné vstupní signál alespoň upravit pomocí filtru typu *dolní propusť*. Díky tomu sice ze signálu budou odstraněny vyšší frekvence, ale bude již možné splnit vzorkovací teorém a takový signál rekonstruovat.

Kvantizace představuje způsob reprezentace navzorkované hodnoty v digitální podobě. Větší počet kvantizačních hladin zvyšuje kvalitu záznamu. V praxi je vzorkování i kvantizace prováděna současně pomocí A/D převodníků, které bývají součástí mnoha obvodů včetně FPGA čipů. Typicky takové převodníky umožňují vzorkovat až několik miliónů vzorků za sekundu s přesností cca 8 až 20 bitů na vzorek.

3.2 Frekvenční analýza

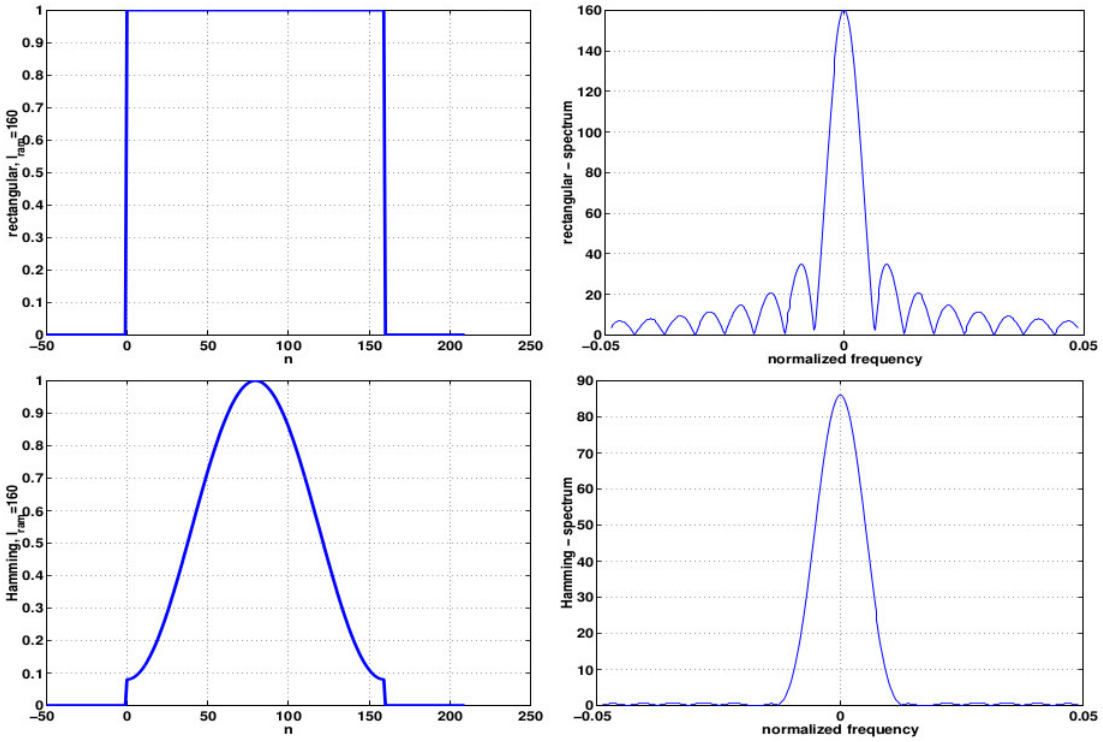
Frekvenční analýzu pomocí DFT je možné rozdělit do několika kroků. DFT typicky nepracuje nad celým vstupním signálem současně, ale je postupně užita pro výpočet krátších segmentů spektra. Výstupem je tzv. *spektrogram*, který zachycuje zastoupení jednotlivých frekvencí v signálu a jejich změny v čase.

Segmentace signálu

V oblasti zpracování radarového signálu se obvykle používají délky segmentu 256 až 1024 vzorků. Čím je délka segmentu kratší, tím většího rozlišení v čase u spektrogramu dosáhneme. Naopak pro dlouhé segmenty roste rozlišení ve frekvenci. Je důležité si uvědomit, že to, jak velký časový úsek segment v signálu reprezentuje, závisí kromě počtu vzorků segmentu také na použité vzorkovací frekvenci. Pro plynulejší přechod mezi jednotlivými výstupy DFT se mohou segmenty částečně překrývat.

Okenní funkce

Pokud bychom vzorky signálu pro daný segment použili bez jakékoliv úpravy, výsledné spektrum by nebylo příliš kvalitní. Důvodem je to, že prosté vyjmutí posloupnosti vzorků ze signálu lze chápat jako použití obdélníkové okenní funkce, která úplně utlumuje všechny vzorky signálu kromě dané posloupnosti, kterou zachová v původní podobě. Charakteristika takové funkce nabízí vysokou frekvenční selektivitu, vhodnou pro přesné určování pozic lokálních extrémů ve spektru. Její nevýhodou je však to, že jednotlivé frekvence budou značně ovlivněny hodnotami jejich sousedních frekvencí. Pro radarové zpracování se jako vhodnější jeví Hammingova a Hannova okenní funkce [13]. Jejich společným rysem je částečné utlumení vzorků na začátku a konci segmentu, čímž je dosaženo určitého potlačení vlivu sousedních frekvencí. Oproti obdélníkové okenní funkci ale nabízí menší frekvenční selektivitu.



Obrázek 3.1: Srovnání Hammingovy a obdélníkové okenní funkce. Na levé straně jsou zobrazeny okenní funkce v časové oblasti a na pravé straně ve frekvenční oblasti. Převzato z [25].

Diskrétní Fourierova transformace

Výstupem DFT je posloupnost hodnot určující zastoupení jednotlivých frekvencí ve vstupním signálu. Následující rovnice popisuje výpočet jednotlivých prvků [10]:

$$X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-j \frac{2\pi}{N} k \ n}, \qquad k \ = 0, 1, ..., N-1 \tag{3.2}$$

kde n je index prvku vstupního segmentu, k je index počítaného frekvenčního prvku a N je délka segmentu. Počet výstupních hodnot odpovídá počtu prvků vstupního segmentu, krok mezi jednotlivými frekvencemi tedy je:

$$\Delta f = \frac{F_s}{N} \tag{3.3}$$

kde F_s je vzorkovací frekvence a N délka vstupního segmentu. Z toho vyplývá, že při vyšším počtu vzorků také roste rozlišení ve frekvenci. Na druhou stranu pak vstupní segment obsahuje signál o delším časovém úseku, což zvyšuje riziko nestacionarity. Tento problém ale lze kompenzovat zvýšením vzorkovací frekvence.

Díky komplexnímu výstupu radaru a vhodně řešenému problému signalizace směru pohybu (kanály I a Q) se po zpracování signálu diskrétní Fourierovou transformací objekt objeví na příslušné straně spektra, tj. na kladné nebo záporné frekvenci podle směru jeho pohybu.

V praxi je ještě nutné řešit problém ovlivnění nízkých frekvencí nenulovou stejnosměrnou složkou radarového signálu. Možností řešení je několik, jednou z nich je odstranění stejnosměrné složky z jednotlivých vstupních rámců vypočtením střední hodnoty, která bude odečtena od každého vzorku. Další možností je použít IIR (*Infinite Impulse Response*) filtr, který bude predikovat střední hodnotu vstupního signálu z dlouhodobého hlediska [13]. Je také možnost vypořádat se s tímto problémem až po výpočtu transformace ve frekvenční oblasti např. pomocí prahování, které nastaví vyšší detekční práh pro ovlivněné nízké frekvence a nižší práh pro vysoké frekvence.

Rychlá Fourierova transformace

Výpočetní složitost DFT je N^2. Pro výpočet DFT nad signály o délce 2^N lze použít FFT, která produkuje stejné výsledky, ale její výpočetní složitost je $N\texttt{log}_2 N$.

Vzorec 3.2 pro výpočet koeficientů DFT je možné při zanedbání finální normalizace hodnotou $\frac{1}{N}$ upravit následujícím způsobem:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad k = 0, 1, ..., N-1 \tag{3.4}$$

$$W_N = e^{-j\frac{2\pi}{N}} \tag{3.5}$$

Pro *fázový faktor* W_N platí:

$$W_N^{k+(N/2)} = -W_N^k \tag{3.6}$$

$$W_N^{k+N} = W_N^k \tag{3.7}$$

Tyto vlastnosti, známé jako *symetrie* (3.6) a *periodicita* (3.7) fázového faktoru, jsou FFT využity pro zefektivnění výpočtu oproti klasické DFT [5].

Radix-2

V této podsekci bude vysvětlena tzv. *radix-2* verze FFT využívající decimaci v čase. Existuje také verze založená na decimaci ve frekvenci [10].

Jelikož je počet vstupních hodnot mocninou dvou, můžeme je rozdělit na dva stejně dlouhé úseky, kde první obsahuje hodnoty na sudé pozici a druhý hodnoty na liché pozici:

$$X(k) = \sum_{n=0}^{(N/2)-1} x(2n) W_N^{2nk} + \sum_{n=0}^{(N/2)-1} x(2n+1) W_N^{(2n+1)k} \tag{3.8}$$

$$= \sum_{n=0}^{(N/2)-1} x(2n) W_N^{2nk} + W_N^k \sum_{n=0}^{(N/2)-1} x(2n+1) W_N^{2nk} \tag{3.9}$$

$$= \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nk}, \quad k = 0, 1, ..., N-1 \tag{3.10}$$

Tímto způsobem je výpočet rozdělen na dvě samostatné DFT délky $N/2$:

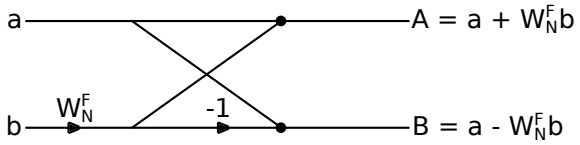
$$X(k) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nk} \tag{3.11}$$

$$= F_1(k) + W_N^k F_2(k), \qquad k = 0, 1, ..., N-1 \tag{3.12}$$

$F_1(k)$ a $F_2(k)$ jsou DFT, které jsou periodické s periodou $\frac{N}{2}$. Tedy $F_1(k+\frac{N}{2}) = F_1(k)$ a $F_2(k+\frac{N}{2}) = F_2(k)$. Fázový faktor $W_N^{k+\frac{N}{2}} = -W_N^k$. Díky tomu je možné výpočet zefektivnit [5]:

$$X(k) = F_1(k) + W_N^k F_2(k), \quad k = 0, 1, ..., \frac{N}{2} - 1 \tag{3.13}$$

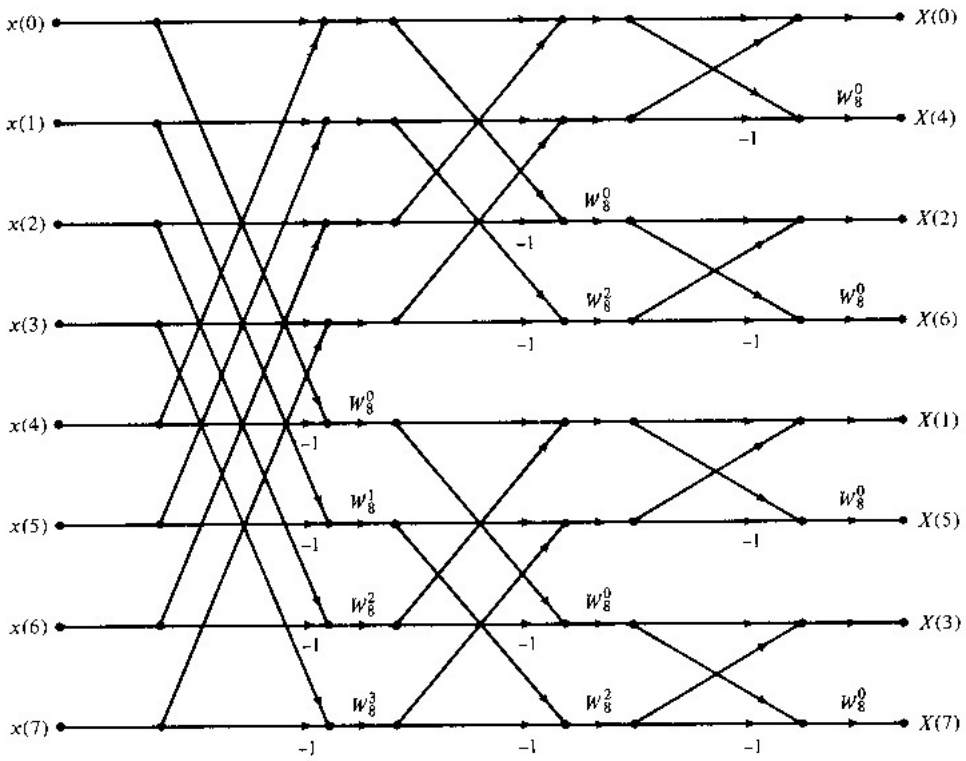
$$X(k + \frac{N}{2}) = F_1(k) - W_N^k F_2(k), \quad k = 0, 1, ..., \frac{N}{2} - 1 \tag{3.14}$$



Obrázek 3.2: Základní struktura používaná v FFT *radix-2* verze s decimací v čase.

Princip rozdělování na DFT poloviční délek je možné opakovat, dokud nedojdeme na DFT délky 1. Výsledek DFT pro vstup o délce 1 je samotná vstupní hodnota:

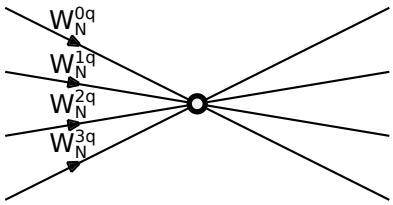
$$X(0) = \sum_{n=0}^{1-1} x(n) W_N^{0n} = n(0) \tag{3.15}$$



Obrázek 3.3: Struktura FFT ve verzi *radix-2* pro N = 8. Převzato z [5].

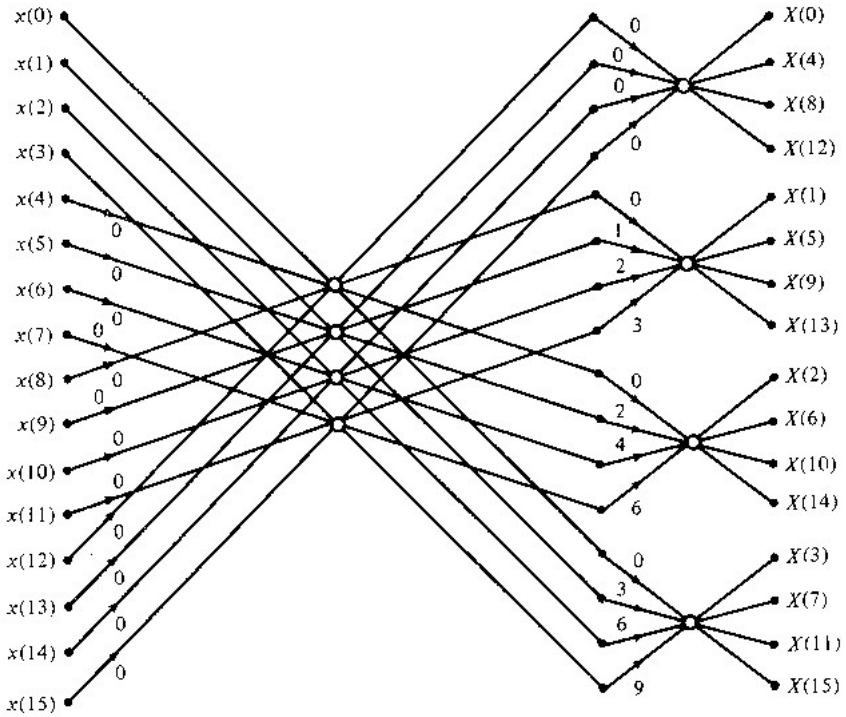
Radix-4

Pro DFT o délce 4^N je možné aplikovat *radix-4* verzi, která využívá složitější základní strukturu se čtyřmi vstupy a čtyřmi výstupy, viz. obrázek 3.4.



Obrázek 3.4: Základní struktura používaná v FFT *radix-4* verze.

Oproti *radix-2* verzi je dosaženo určitého zlepšení výpočetní složitosti, především je ale zde zapotřebí polovina výpočetních fází. Tedy při paralelním zpracování na FPGA může být dosaženo zhruba poloviční latence [23]. Obrázek 3.5 ukazuje výslednou strukturu pro FFT délky 16.



Obrázek 3.5: Struktura FFT ve verzi *radix-4* pro N = 16. Převzato z [5].

3.3 Reprezentace desetinných čísel

Samotnému zpracování radarového signálu předchází rozhodnutí, v jakém formátu budou reprezentovány mezivýsledky a výsledky s desetinnými číselnými hodnotami. Je možné se rozhodnout pro formát s pevnou desetinnou čárkou nebo pro formát s pohyblivou desetinnou čárkou. Oba formáty mají své výhody a nevýhody.

Čísla s plovoucí desetinnou čárkou

Pro klasické univerzální procesory, používané např. v osobních počítačích, jednoznačně dominuje reprezentace pomocí pohyblivé desetinné čárky. Součástí těchto procesorů je standardně jednotka FPU (*Floating Point Unit*) pro práci s čísly s pohyblivou desetinnou čárkou, která výpočty v tomto formátu akceleruje. Principem je vyhrazení určitých bitů z celkového počtu pro tzv. *mantisu* a *exponent*. Další bit pak určuje, zda bude číslo kladné nebo záporné.

$$X = (-1)^S * M * 2^E \tag{3.16}$$

V praxi se používají především dvě verze definované standardem *IEEE 754* [1] – verze s tzv. základní přesností, která z celkového počtu 32 bitů vyhrazuje 23 bitů pro mantisu a 8 pro exponent a verze s tzv. dvojnásobnou přesností, používající 52 bitů pro mantisu a 11 bitů pro exponent. Vzhledem k tomu, že exponent může vyjadřovat kladnou i zápornou hodnotu, je možné takto reprezentovat extrémně velké i malé hodnoty. Tato vlastnost, známá jako

vysoký dynamický rozsah, je současně největší výhodou čísel s pohyblivou desetinnou čárkou. Je tak možné reprezentovat s relativně vysokou přesností hodnoty, o kterých nemáme apriorní znalost o tom, v jakém rozsahu se budou nacházet.

Čísla s pevnou desetinnou čárkou

Jinou možností je reprezentace pomocí pevné desetinné čárky. V tomto případě je pevně určený počet bitů z celkového počtu vyhrazen pro celočíselnou část, příp. i znaménko, a zbytek bitů reprezentuje desetinnou část. Stejně jako pro celočíselné typy, je pro nezáporná čísla možné použít přímý binární kód, zatímco pro záporná se používá dvojkový doplněk. Existuje několik notací, kterými se vyjadřuje počet bitů pro celočíselnou a desetinnou část. Pravděpodobně nejpoužívanější je ve tvaru typu Q3.2, kde Q indikuje typ s pevnou desetinnou čárkou, následované počtem bitů pro celočíselnou část, tečkou a počtem bitů pro desetinnou část. Interpretace hodnoty čísla ve formátu např. Q3.2 je následující:

$$X = b_4 2^2 + b_3 2^1 + b_2 2^0 + b_1 2^{-1} + b_0 2^{-2} \tag{3.17}$$

kde X je reprezentovaná hodnota a b_x představuje hodnotu bitu na indexu x.

Pro jednotlivé základní aritmetické operace (sčítání, odečítání, násobení a dělení) platí odlišná pravidla z pohledu potřebného formátu operandů a formátu výsledku [9]:

- **Sčítání a oděčítání** - oba operandy musí být ve stejném formátu, tj. musí mít shodný počet bitů pro celočíselnou i desetinnou část. Výsledek je ve stejném formátu jako operandy. Může nicméně dojít k přetečení, které způsobí rozšíření o jeden nejvýznamnější bit.

- **Násobení** - operandy mohou být v odlišném formátu. Počet bitů jednotlivých částí výsledku je sumou počtu bitů částí operandů. Tedy při násobení čísla a ve formátu $Qi_a.f_a$ a čísla b ve formátu $Qi_b.f_b$ bude výsledek ve formátu $Qi_a + i_b.f_a + f_b$.

- **Dělení** - operandy opět mohou být v odlišném formátu. Počet bitů výsledku je určen rozdílem počtu bitů operandů. Při dělení čísla a ve formátu $Qi_a.f_a$ číslem b ve formátu $Qi_b.f_b$ bude výsledek ve formátu $Qi_a - i_b.f_a - f_b$.

Pro operace v tomto formátu je možné využít obvody pro práci s celočíselnými typy. Mezi výhody patří také efektivní využití všech bitů – např. při uložení čísel v rozsahu $< 0; 1)$ ve fortmátu *IEEE 754* se základní přesností a ve formátu Q0.32 bude větší přesnosti dosaženo v druhém případě, protože není potřeba ukládat informaci o exponentu.

Reprezentace v FPGA

Čísla s pohyblivou řádovou čárkou nabízejí pro většinu aplikací dostatečnou přesnost a díky vysokému dynamickému rozsahu je jejich použití velice univerzální. Na druhou stranu obvody pro práci s nimi jsou poměrně složité a na FPGA zabírají velké množství zdrojů [24].

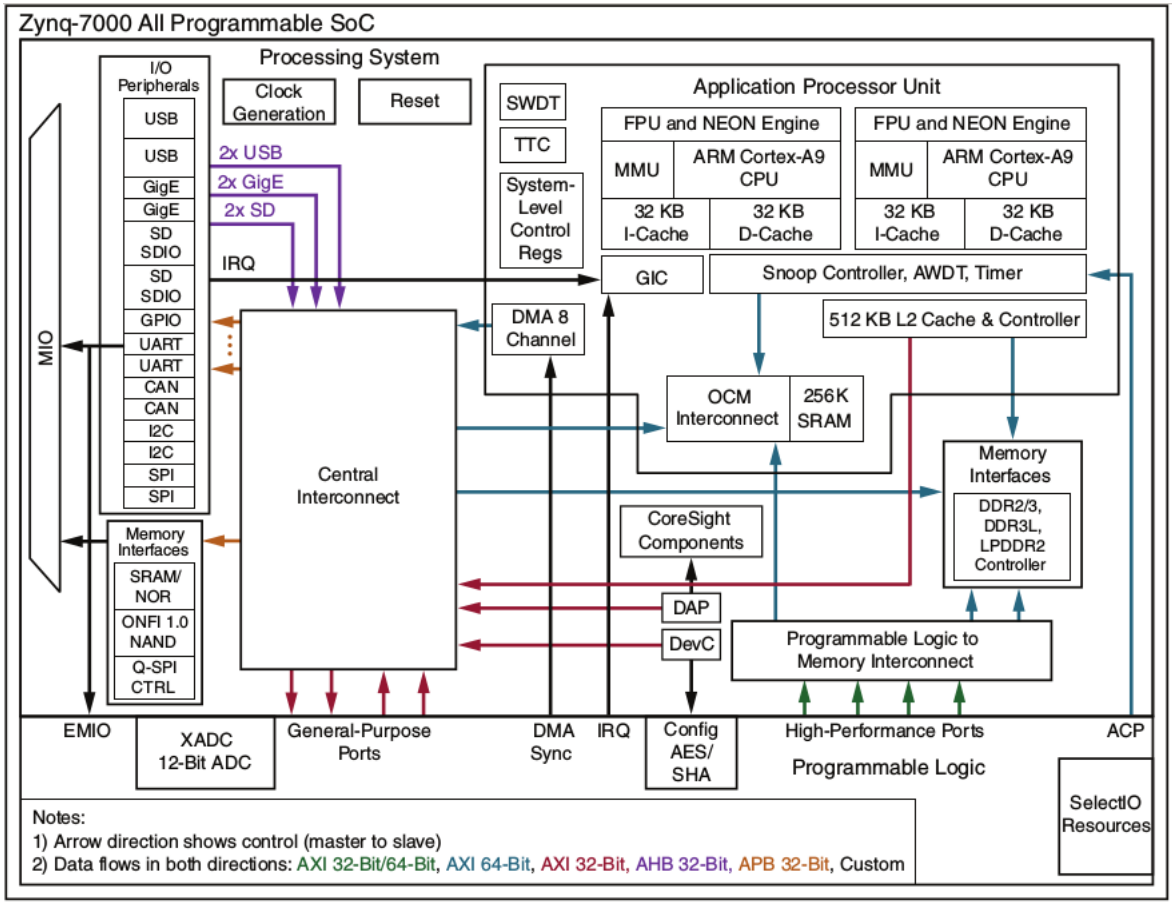
Pro použití formátu s pevnou řádovou čárkou v FPGA nahrává především jednoduchost realizace výpočetních obvodů a fakt, že FPGA je z principu dobře uzpůsobeno pro použití různého počtu bitů pro celočíselnou a desetinnou část podle potřeb vyvíjené aplikace.

Lze říci, že použití čísel s pevnou desetinnou čárkou přináší úsporu v počtu použitých zdrojů a nabízí vyšší výkon za cenu analýzy povahy vstupních dat (jejich rozsahu) a složitějšího návrhu. Pro čísla malého rozsahu také nabízejí vyšší přesnost.

Kapitola 4

Platforma Zynq

Platforma Zynq představuje jeden z nejnovějších produktů firmy Xilinx. Jedná se o architekturu propojující procesorový systém, založený na dvoujádrovém procesoru *ARM Cortex-A9* a programovatelnou logiku v rámci jednoho čipu. Na rozdíl od předchocích programovatelných obvodů není srdcem zařízení programovatelná logika, ale výkonný procesorový systém, který je při startu zařízení inicializován jako první a který poté provádí konfiguraci částí programovatelné logiky. Díky úzkému propojení obou subsystémů na jednom čipu platforma nabízí flexibilitu, škálovatelnost a vysoký výkon. Produktová řada *Zynq-7000* sestává z několika variant modelů. Uživatel si tak může vybrat čip, který pokryje jeho nároky a současně nebude zbytečně drahý. Důležité je, že všechny čipy Zynq obsahují stejný procesorový systém a liší se ve velikosti a výbavě programovatelné logiky. Tím je umožněno, aby návrh pro jednotlivé varianty probíhal shodným způsobem a aby případná migrace aplikace na jiný model vyžadovala pouze minimum změn. Čipy nižší modelové řady využívají programovatelnou logiku shodnou s tou, která je použita v čipech *Artix-7*, zatímco ty z vyšší řady jsou vybaveny logikou čipů *Kintex-7*. Architektura platformy Zynq je zobrazena na obrázku 4.1.



Obrázek 4.1: Architektura platformy Zynq-7000. Převzato z [22]

Zynq nabízí několik výhod oproti řešením, kde je FPGA i procesor na samostatných čipech propojených pomocí sběrnice nebo linek. Mezi ně patří menší velikost, jednodušší návrh systému, nižší spotřeba, vyšší propustnost, flexibilita sběrnic mezi procesorem a FPGA a v neposlední řadě také zjednodušení celkového vývoje díky sofistikovaným nástrojům. Další alternativou k platformě Zynq může být FPGA o dostatečné velikosti, ve kterém bude realizován procesor pomocí prostředků programovatelné logiky. Výhodou tohoto řešení je flexibilita, protože tyto tzv. *soft–procesory* mají obvykle širokou nabídku volitelných možností, kterými lze ovlivňovat jejich výsledký výkon a množství zabraných zdrojů v FPGA. Současně je možné v FPGA vytvořit několik takových procesorů - limitujícím faktorem jsou pouze dostupné zdroje FPGA. Příkladem *soft–procesorů* jsou procesory *MicroBlaze*, které Xilinx podporuje ve svých vývojových nástrojích. Nevýhodou oproti platformě Zynq je mnohem nižší výpočetní výkon. Výkon procesoru *MicroBlaze* s optimalizacemi na výkon v programovatelné logice Artix-7 je 300 DMIPs (Dhrystone Millions of Instructions Per second) [20], zatímco procesor *ARM Cortex-A9* v Zynq má výpočetní výkon až 5000 DMIPs [22].

4.1 Propojovací systém

Propojení procesorového systému a programovatelné logiky je dosaženo pomocí několika AXI kanálů, které vycházejí z rodiny sběrnic ARM AMBA. Je podporováno několik současných *master–slave* přenosů. Propojovací soustava je navržená takovým způsobem, aby zařízení požadující nízkou latenci (ARM procesor) byly připojeny k paměti pomocí co nejkratší cesty. Obdobně je sběrnice přizpůsobena pro zařízení, která požadují vysokou propustnost. Dále je podporována regulace a zajištění provozu pomocí QoS (*Quality Of Service*) bloku. Jsou k dispozici čtyři univerzální porty a čtyři *high performance* porty, které umožňují přenosy s vysokou propustnostností mezi programovatelnou logikou a DDR pamětmi, resp. vestavěnou pamětí na čipu. Poslední možností propojení procesorového systému a programovatelné logiky je ACP (*Accelerator Coherency Port*) port, který dovoluje zařízením v programovatelné logice koherentně přistupovat přímo do cache pamětí procesoru.

Mimo datových kanálů je k dispozici také několik signálů pro přerušení v obou směrech. Na straně procesorového systému je jednotka GIC (*Generic Interrupt Controller*), která je schopna povolovat, maskovat a nastavovat prioritu pro jednotlivé zdroje přerušení.

4.2 Procesorový systém

Jak je možné vidět na obrázku 4.1, procesorový systém je tvořen především blokem APU (*Application Processor Unit*), který obsahuje dvoujádrový procesor *ARM Cortex-A9*. Procesor ARM je schopen bežet na maximální frekvenci 667 MHz až 1 GHz v závislosti na konkrétní variantě modelu. Každé jádro je mimo jiné vybaveno jednotkou pro zpracování čísel s plovoucí desetinnou čárkou FPU, akcelerační jednotkou NEON umožňující výpočty pomocí SIMD instrukcí a 32 kB L1 cache pro instrukce a data. 512 kB L2 cache je stejně jako malá 256 kB *on-chip* SRAM sdílená. Pro přístup k ostatním periferiím se využívá paměťového mapování.

Periferie

Možnost připojení externích periferií je další důležitou vlastností procesorového systému. K tomuto účelu existuje rozhraní MIO (*Multiplexed Input/Output*), které umožnuje přiřazovat piny čipu ke konkrétním periferiím a paměťovému rozhraní. Je k dispozici 54 univerzálních pinů, které mohou být pomocí MIO přímo připojeny k procesoru, resp. paměťovému rozhraní. V případě potřeby většího počtu pinů je nutné využít bloku EMIO (*Extendable Multiplexed I/O*), který je schopný piny mapovat do programovatelné logiky. Mezi periferie procesorového systému patří:

- **Ethernet** - Celkem 2 rozhraní podporující rychlosti 10 / 100 / 1000 Mbps.
- **USB** - K dispozici jsou 2 rozhraní verze USB 2.0 OTG (*On The Go*), které podporuje dynamickou změnu módu USB Host a USB zařízení.
- **SD/SDIO** - Paměťové karty jsou často využívány pro uložení operačního systému a konfigurace programovatelné logiky, které se načtou při startu systému. Jde o jeden z možných způsobů pro nahrání konfigurace, který eliminuje nutnost použití speciálních programovacích rozhraní na zařízení.
- **SPI a I2C** - Standardní sériové synchronní sběrnice.

- **UART** - Rozhraní umožňující komunikaci po asynchronní sériové sběrnici. Při vývoji je často využíváno pro interaktivní terminál zařízení, ke keterému se z počítače lze připojit pomocí sběrnice RS232.
- **GPIO** - Univerzální vstupně / výstupní piny, které se obvykle používají spíše pro nízkoúrovňové řízení jiných zařízení - rozsvěcování LED diod, aktivace / deaktivace specifických funkcí, snímání tlačítek, přepínačů atd.

4.3 Programovatelná logika

Jak již bylo zmíněno, programovatelná logika na platformě Zynq byla převzata z produktové řady *Artix-7* a *Kintex-7*. Je tvořena CLB (*Configurable Logic Blocks*) bloky pro realizaci kombinační a sekvenční logiky, IOB (*Input/Output Blocks*) bloky pro připojení fyzických pinů a dvěma typy speciálních zdrojů, kterými jsou blokové RAM paměti a DSP (*Digital Signal Processing*) bloky. Kromě užitečných zdrojů musí být realizována také přenosová a propojovací logika.

CLB bloky

Elementárním prvkem CLB bloku je tzv. LUT (*Look-Up Table*) tabulka, která je schopna realizovat logickou funkci až se šesti vstupy, nebo může být použita jako malá ROM/RAM paměť. LUT mohou být konfigurovány jako jedna 6-ti vstupá LUT s jedním výstupem, nebo jako dvě 5-ti vstupé LUT se sdílenými vstupy. Výstup každé LUT může být volitelně uložen ve *flip-flop* registru (bez hodinové synchronizace). Čtyři LUT, osm s nimi spojených *flip-flop* registrů, přenosová logika (*carry logic*) a přidružené multiplexory tvoří jeden *slice*. CLB blok je tvořen dvěma *slice* bloky. Část ze všech *slice* bloků může využít jejich LUT jako 64-bitovou distribuovanou paměť nebo jako 32-bitové shift registry [19].

IOB bloky

IOB Bloky poskytují rozhraní mezi zdroji programovatelné logiky a fyzickými piny a jsou obvykle rozmístěny po okrajích čipu. Každý IOB blok je schopný obsloužit jeden vstupní nebo výstupní bit.

BRAM paměti

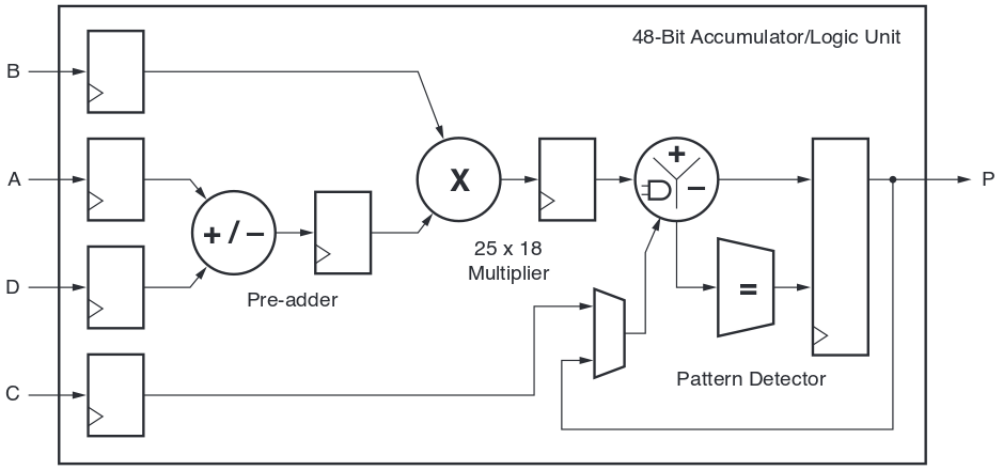
Každá bloková RAM paměť má kapacitu 36 Kbit a je vybavena dvěma nezávislými porty. Každý přístup do paměti (čtení i zápis) je řízen hodinovým signálem. Veškeré vstupy včetně dat, adresy a povolovacích signálů jsou ukládány do registrů. Volitelně je možné použít registr pro uložení výstupu paměti a umožnit tak zvýšení propustnosti paměti pomocí řetězení. Cenou je pak navýšení latence paměti o jeden hodinový takt. Šířka paměti, resp. šířka slova je konfigurovatelná. Port paměti může být nastaven v rozmezí 32K x 1 až 512 x 72. Každou paměť lze navíc rozdělit na dvě nezávislé 18 Kbit paměti.

Paměti jsou navíc vybaveny logikou pro detekci a opravu chyb pomocí Hammingova kódu. Této funkce je možné využít při délce slova 64 bitů. K datům je přidáno 8 zabezpečujících bitů, které umožňují opravu jedno-bitové chyby a detekci dvou-bitové chyby.

DSP bloky

Přestože je možné implementovat aritmetické operace libovolné délky pomocí LUT tabulek, pro dlouhé délky slov není tento přístup efektivní, jelikož výsledný obvod pak většinou zabírá velké množství zdrojů a není schopen provozu na vysokých frekvencích. Pro tyto účely existují DSP bloky, které nabízí vysokou efektivitu a rychlost zpracování.

Každý DSP blok, označovaný jako *DSP48E1*, sestává z dedikované 25x18 bitové násobičky a 48-bitového akumulátoru, který může být použit jako čítač, a je schopný bežet na frekvenci až 741 MHz. Dále je přítomna škálovatelná SIMD jednotka, která je schopna zpracovávat dva operandy o velikosti 24 bitů nebo čtyři operandy o velikosti 12 bitů. Zjednodušené schéma DSP bloku je znázorněno na obrázku 4.2. Mezi další funkce DSP slice patří logická jednotka schopná generovat jednu z deseti logických funkcí dvou operandů, 48-bitový detektor vzorů (komparátor), před-sčítačka a další. Podobně jako u BRAM pamětí je možné využít řetězení, které vede k vyšší rychlosti zpracování a lepší efektivitě jednotky [18].



Obrázek 4.2: Schéma bloku DSP48E1. Převzato z [18])

A/D převodníky

Další komponentou programovatelné logiky je *XADC* blok, který obsahuje dva A/D převodníky schopné vzorkovat milión hodnot za sekundu s rozlišením 12 bitů. Kromě toho je tento blok vybaven senzory pro měření teploty pouzdra čipu a napětí integrovaného napájecího zdroje. Převodníky jsou schopny provozu v unipolárním i bipolárním režimu. Práce a konfigurace s převodníky je možná skrze DRP (*Dynamic Reconfiguration Port*). Limitací *XADC* může být fakt, že převodníky jsou schopné zpracovat pouze signály s napěťovými úrovněmi 0V až 1V, případně -0,5V až 0,5V v bipolárním režimu [21].

4.4 Vývoj na Zynq

Na procesorovém systému lze provozovat několik typů operačních systémů. Jsou dostupné *real-time* operační systémy (RTOS), které se hodí pro případy, kdy potřebujeme mít garantováno zpracování úlohy ve stanoveném čase. Pro maximální využití zdrojů a výkonu

čipu jsou naopak vhodné jiné operační systému, jako je Linux nebo Android. Vzhledem k tomu, že je Zynq víceprocesorový systém, lze provozovat současně více operačních systémů. Varianta, označovaná jako *asymmetric multi-processing* označuje vykonávání různých typů operačních systémů. Je tedy možné, aby se na jednom jádře vykonával např. Linux a na druhém nějaký RTOS. Komunikace mezi operačními systémy pak probíhá pomocí sdílené paměti. Stejně tak je možné, aby na všech jádrech běžely oddělené instance stejného operačního systému (*symmetric multi-processing*) [6].

Bootovací proces je rozdělen do několika fází. Po startu zařízení se na základě konfiguračních pinů vybere zdroj, ze kterého se bude nahrávat FSBL (*First Stage Boot Loader*). Mezi tyto zdroje patří NOR, NAND, Quad-SPI, paměťová karta (SD) a odlaďovací rozhraní JTAG. Procesor vykoná program uložený v paměti ROM, který překopíruje FSBL do paměti na čipu a začne jej vykonávat. FSBL pak pokračuje v bootovacím procesu a může inicializovat programovatelnou logiku. Typicky FSBL nahraje přímo uživatelskou aplikaci (v případě, kdy není použit operační systém), nebo SSBL (*Second Stage Boot Loader*) jako např. *U-Boot*, který nahraje cílový operační systém [22].

Vývojové prostředí Vivado

Vývojové prostředí *Vivado* je následovníkem *ISE Design Suite* a integruje všechny důležité nástroje do jednoho celku s jednotným uživatelským rozhraním. Pokrývá tak všechny fáze vývoje, mezi které patří:

- **Návrh** - Počáteční fáze, ve které dochází k návrhu obvodu. Jako přídavek ke klasickému popisu pomocí HDL jazyků lze využít blokový editor, který umožňuje přidávat a napojovat jednotlivé komponenty pomocí grafického rozhraní.
- **Syntéza** - V tomto kroku je návrh obvodu překládán z RTL reprezentace do tzv. *netlistu*, který využívá komponent cílové architektury (FPGA). Je k dispozici několik parametrů, pomocí kterých lze ovlivňovat průběh syntézy a např. tak optimalizovat obvod s cílem na vysoký výkon nebo nízkou spotřebu.
- **Implementace** - Implementace provádí mapování jednotlivých částí obvodu na konkrétní prvky cílového zařízení a vytváří spojení mezi nimi. Mimo jiné dochází k napojování výstupů obvodu, které jsou na to určeny, k fyzickým pinům čipu.
- **Programování a odlaďování** - Je generován *Bitstream*, což je binární reprezentace obvodu, kterou je možné přímo nahrát do FPGA.

Dále jsou k dispozici nástroje pro simulaci, práci s IP cores, odhad spotřeby a další.

Vivado podporuje vývoj pro FPGA řady 7 a pro čipy Zynq-7000, starší modely jako *Spartan* již nejsou podporovány. Při jednotlivých fázích vývoje jsou k dispozici detailní přehledy o vlastnostech vyvíjeného projektu. Jednou ze zajímavých vlastností je rozhraní pro jazyk TCL (*Tool Command Language*), díky kterému je možné plnohodnotně nahradit grafické uživatelské rozhraní pomocí skriptů. Lze pozorovat zřetelný důraz na vysokoúrovňovou syntézu HLS (*High Level Synthesis*), která umožňuje pro popis obvodů použít namísto tradičních HDL jazyků vysokoúrovňové jazyky C/C++.

IP cores

Při vývoji rozsáhlých vestavěných systémů hraje důležitou roli znovupoužitelnost částí, které již byly implementovány a otestovány. Tyto části, nazývané jako IP (*Intellectual Property*) cores, řeší specifické úlohy a ve výsledku přispívají ke zkrácení doby vývoje systému a ke zvýšení jeho spolehlivosti. IP cores jsou do zbytku systému připojeny pomocí rozhraní realizující vstup a výstup, přičemž jejich vnitřní implementace nemusí být uživateli známa. Dalším efektem používání těchto komponent je zvyšování úrovně abstrakce návrhu, což také přispívá ke kvalitnějšímu procesu vývoje. IP cores je možné získat několika způsoby. Samozřejmým zdrojem jsou starší projekty, jejichž návrh byl vhodně dekomponován na více na sobě nezávislých a obecných komponent. Dále jsou to samotní výrobci FPGA a třetí strany, kteří poskytují velké množství volně dostupných i placených IP cores. Ve vývojovém prostředí Vivado jsou IP cores velmi úzce integrované do blokového editoru, kde je možné vkládat a napojovat IP cores jako bloky pomocí grafického rozhraní. Tvorba IP cores je možná pomocí průvodce, který po zadání potřebných údajů vytvoří archiv, který je možné ve Vivadu naimportovat a IP core použít.

Rozhraní AXI

Standardem pro přenos dat v FPGA se stalo rozhraní AXI. To je podmnožinou rodiny sběrnic ARM AMBA, přičemž používaná verze je AXI4. Jedná se o otevřenou specifikaci propojovacího protokolu, určeného pro přenos dat a řízení mezi jednotkami na čipu. Rozhraní AXI je navrženo pro vysoký výkon a vysoké frekvence. Na jeho návrhu se podílela přímo firma Xilinx, díky čemuž je možná efektivní implementace v FPGA. V rámci standardizace se toto rozhraní běžně používá pro komunikaci s IP cores. Jsou k dispozici tři verze protokolu [6]:

- **AXI4** - Základní plnohodnotná varianta, která umožňuje paměťově mapovaný přenos dat. Pro zvýšení propustnosti jsou umožněny dávkové přenosy, kdy je po fázi vystavení adresy možné provést až 256 cyklů datových přenosů.
- **AXI4-Lite** - Odlehčená verze, která se od té předchozí liší tím, že nepodporuje dávkové přenosy. Díky tomu má zjednodušené rozhraní a zabírá méně zdrojů programovatelné logiky. Je vhodná pro řízení a přenosy dat, pro které není kritická vysoká propustnost.
- **AXI4-Stream** - Tato varianta zcela odstraňuje adresování, takže jsou přenášena pouze data a řídící signály. Odstranění adresy umožňuje provádět dávkové přenosy neomezené délky. To je vhodné v případě proudového zpracování, kdy data postupně prochází jednotlivými bloky. Je podporován pouze jednosměrný *master–slave* přenos. V případech, kdy požadujeme komunikaci v obou směrech, musí být použito dvou těchto rozhraní.

Rozhraní AXI je obecně dobře škálovatelné, je možné zvolit datovou šířku a spousta signálů je volitelných. Kromě samotných rozhraní jsou k dispozici také nejrůznější přepínače, fronty, DMA a další podpůrné jednotky, které umožňují plnohodnotné využití rozhraní.

Kapitola 5

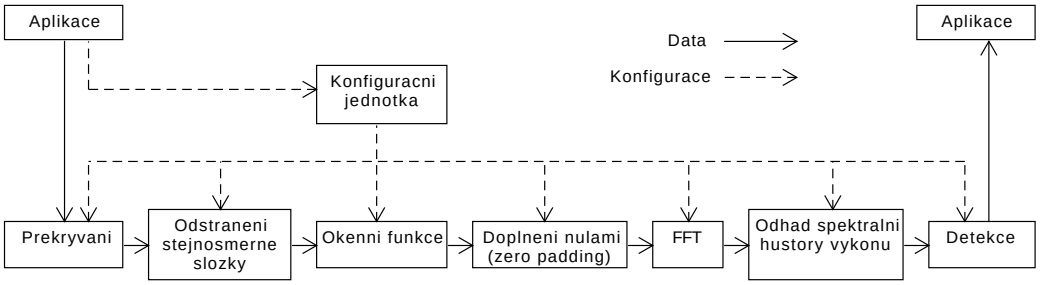
Návrh

Tato kapitola popisuje návrh radarového procesoru pro cílovou platformu Xilinx Zynq. Je specifikováno zadání, které popisuje především vstupní a výstupní data a konfigurovatelné parametry zpracování. Následně je představen návrh celkové architektury a jsou popsány jednotlivé bloky, ze kterých se skládá řetězec zpracování dat. Současně je proveden návrh programového řešení v C++, které bude sloužit především k vyhodnocení výkonu finální implentace. Poslední část se zabývá návrhem testů pro ověření správnosti, přesnosti, rychlosti a plochy na čipu.

5.1 Specifikace zadání

Radarový procesor v FPGA bude přijímat navzorkované hodnoty jednoho radarového kanálu. Výstupem bude výkonové spektrum signálu spolu s informací o tom, zda došlo k detekci objektu. Pro reprezentaci hodnot budou pužita čísla s pevnou řádovou čárkou. Mezi parametry konfigurovatelné za běhu bude patřit délka rámce, velikost překrytí, hodnoty okenní funkce, délka FFT a detekční prahy pro jednotlivé frekvenční složky. Parametry ovlivňující přesnost zpracování a limity jednotky (přesnost výstupu udaná v počtech bitů, maximální dovolená velikost FFT atd.) budou konfigurovatelné v době syntézy. Radarová data budou uložena v souborech na procesorovém systému. Pro předání dat do FPGA a následné vyčtení a uložení výsledků vznikne aplikace, která současně provede nastavení procesoru požadovaným způsobem. Cílovou platformou bude Xilinx Zynq, na kterém bude nainstalovaný operační systém Linux.

5.2 Blokové schéma



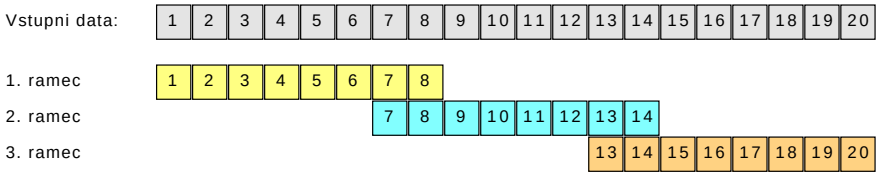
Obrázek 5.1: Blokové schéma radarového procesoru.

Konfigurační jednotka

Vzhledem k tomu, že některé parametry zpracování jsou konfigurovatelné z aplikace za běhu, je nutné odpovídající bloky správně nakonfigurovat ještě před zahájením odesílání dat radarové nahrávky do procesoru.

Překrytí rámců

Tento blok bude přijímat proud navzorkovaných dat a bude je rozdělovat do segmentů požadované délky. Segmenty se mohou překrývat o zadaný počet vzorků, přičemž maximální hodnota překrytí bude omezena v době syntézy. Pro překrytí bude využita interní paměť, do které se budou ukládat poslední vzorky aktuálního rámce. Následující rámec pak budou tvořit vzorky z této paměti následované několika vstupními vzorky tak, aby délka rámce odpovídala nakonfigurované délce.



Obrázek 5.2: Rozdělení vstupních dat do rámců pro délku rámce 8 a překrytí 2.

Odstranění stejnosměrné složky

Úkolem této jednotky bude odstranění neužitečné stejnosměrné složky signálu rámce. Z hlediska efektivnosti implementace na FPGA by bylo vhodné použít filtr, který by predikoval střední hodnotu na základě nekolika málo předchozích vzorků a tu odečítal od vzorku aktuálního. V případě programového řešení je však častější varianta, ve které se střední hodnota

spočte ze všech vzorků rámce. Tento způsob vede k lepším výsledků a bude proto v této práci použit.

Střední hodnota tak bude známa až po přijetí posledního vzorku rámce. To znamená, že celý vstupní rámec bude nutné uložit do paměti. Následně bude od každého vzorku odečtena střední hodnota a rámec bude předán k dalšímu zpracování. Latence tohoto bloku tak bude odpovídat délce rámce.

Okenní funkce

Blok okenní funkce provede násobení vzorků rámce s nakonfigurovanými koeficienty. Koeficienty okenní funkce budou uloženy v paměti jednotky a budou modifikovatelné aplikací. V době syntézy bude nutné určit přesnost koeficientů v počtu bitů. Vzhledem k použití čísel s pevnou desetinnou čárkou bude při násobení docházet k navýšení počtu bitů na vzorek rámce.

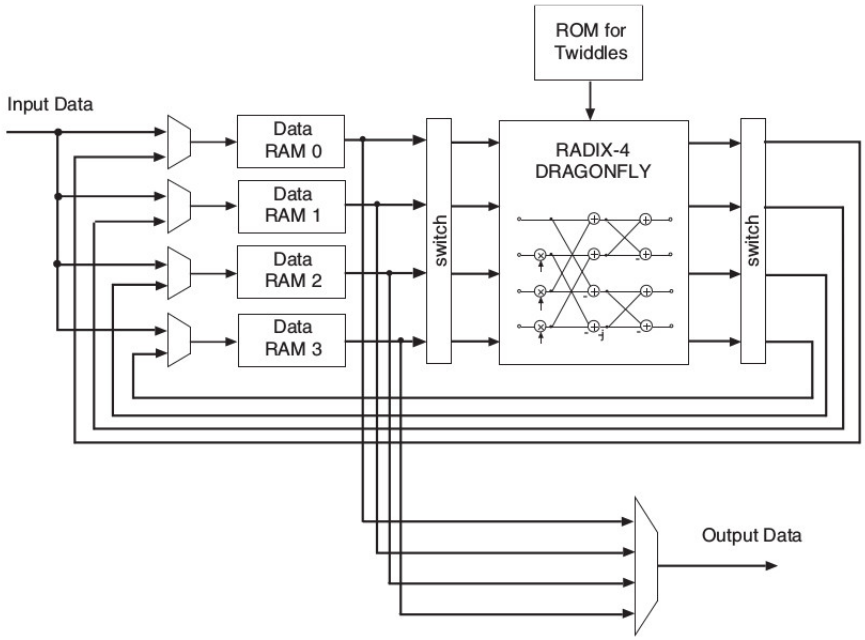
Doplnění rámce nulami

Délka rámce vstupujícího do FFT musí být 2^N (v případě *radix-4* verze FFT dokonce 4^N). Pokud délka rámce po předchozím zpracování tuto podmínku nesplňuje, je možné použít doplnění rámce nulovými hodnotami tak, aby byl rámec prodloužen na nejbližší vyhovující délku.

Rychlá Fourierova transformace

Blok provádějící frekvenční analýzu pomocí FFT. Za tímto účelem bude použit Xilinx XFFT IP core [23], který je schopen výpočtu komplexní FFT nad hodnotami uloženými ve formátu s pevnou i pohyblivou desetinnou čárkou. Bude využita vlastnost jednotky konfigurace délky FFT za běhu, takže odpadne nutnost použít několik verzí FFT pro jednotlivé podporované délky rámce. V době syntézy je nutné zadat mimo jiné maximální dovolenou délku rámce a počet bitů na vstupní vzorek. IP core podporuje FFT ve verzi *radix-2* i *radix-4*. Procesor bude navržen tak, aby bylo možné použít obě verze a provést srovnání vlivu na výsledný výkon.

Z hlediska výkonu je vhodné poznamenat, že Xilinx XFFT IP core při dávkovém zpracování ve verzi *radix-2* i *radix-4* nevytváří kompletní strukturu FFT (jako např. na obrázku 3.5), ale v průběhu výpočtu využívá opakovaně jeden základní blok – viz. obrázek 5.3.



Obrázek 5.3: Architektura Xilinx XFFT IP core *radix-4* verze pro dávkové zpracování. Převzato z [23].

Tímto způsobem jsou šetřeny zdroje FPGA. Dále je podporována zřetězená architektura, která vytváří kompletní *radix-2* strukturu FFT (obrázek 3.3)). Tato verze je však velmi náročná na zdroje a klade další požadavky na vstupní data.

Spektrální hustota výkonu

Z výstupu FFT bude následně spočtena hustota spektrálního výkonu podle vztahu [25]:

$$P(k) = \frac{|X(k)|^2}{N} = \frac{X(k)\overline{X(k)}}{N} \tag{5.1}$$

kde X jsou výstupní koeficienty FFT, $\overline{X}$ jsou komplexně sdružené verze a N je délka rámce. Vzorec je možné dále upravit:

$$P(k) = \frac{X(k)\overline{X(k)}}{N} = \frac{Re(X(k))^2 + Im(X(k))^2}{N} \tag{5.2}$$

Takto je komplexní výstup FFT transformován do reálného signálu, který je pro detekci objektů vhodnější.

Detekce

Tento blok porovnává hodnotu každého koeficientu rámce s jemu odpovídající prahovou hodnotou. V případě, že je koeficient roven prahové hodnotě nebo je větší, je k rámci přidána informace o tom, že došlo k detekci. Prahové hodnoty jsou, podobně jako v případě okenní funkce, dostupné aplikaci a jsou konfigurovatelné za běhu procesoru.

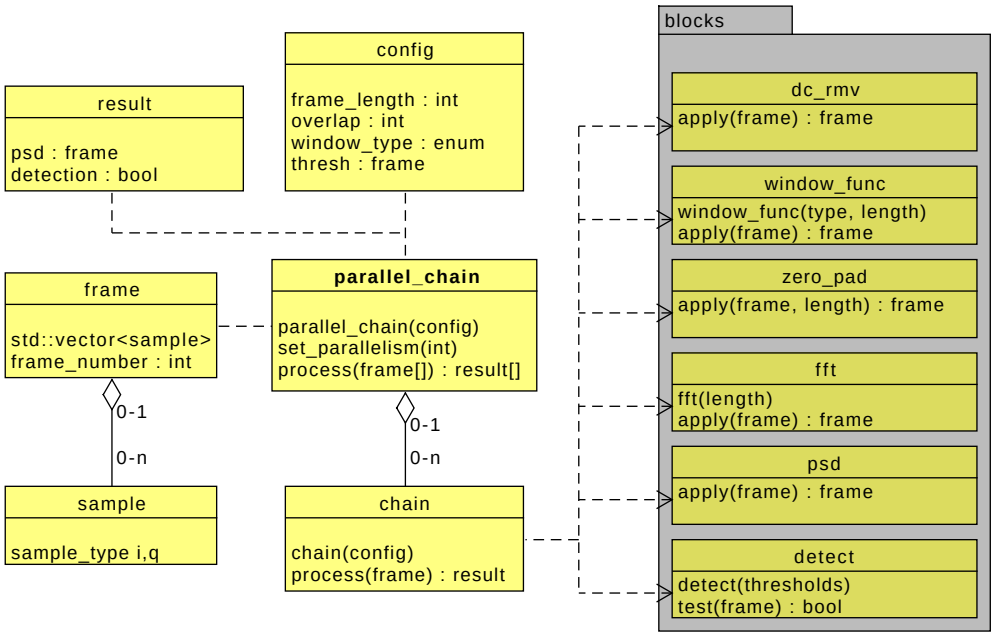
Díky možnosti nastavení různých detekčních hodnot pro jednotlivé frekvence lze např. detekovat pouze objekty pohybující se směrem k radaru (kladná část výkonového spektra) a objekty v opačném směru ignorovat.

Ve výsledku je tak do rámce se spektrální hustotou výkonu přidána informace, zda došlo k detekci. Výstup je pak předán aplikaci.

5.3 Programová část

Radarový procesor v FPGA bude obsluhovat aplikace napsaná v jazyce C++, která poběží na operačním systému Linux přímo na platformě Zynq. Pro přenos radarových dat a výsledků z procesoru bude použit DMA přenos. Za tímto účelem mi byl poskytnut DMA ovladač, který řeší kernelové záležitosti a čtení i zápis přes DMA zjednodušuje na klasické souborové operace. Pro přenos konfigurace se použije paměťově mapovaný přístup na fyzickou adresu, na kterou bude připojena konfigurační jednotka v FPGA pomocí rozhraní AXI. V tomto případě půjde pouze o jednosměrnou komunikaci. Dále bude nutné řešit generování okenních funkcí a načítání radarových dat z poskytnutých nahrávek.

Kromě obslužné aplikace vznikne také čistě programová implementace výše popsaného řetězce zpracování. Díky přenositelnosti kódu C++ tak bude možné porovnat dosažený výkon FPGA řešení nejen oproti implementaci na integrovaném procesoru ARM platformy Zynq, ale i na výkonnějších procesorech. Důležitou vlastností bude schopnost využít všechna procesorová jádra, aby byl co nejvíce využit potenciál daného procesoru. Obrázek 5.4 zachycuje diagram tříd a způsob paralelizace úlohy.



Obrázek 5.4: Zjednodušený diagram tříd. Paralelizace na vícejádrovém procesoru bude dosaženo vytvořením několika řetězců zpracování, přičemž každý poběží v samostatném vlákně.

Rychlá Fourierova transformace

Pro výpočet FFT je k dispozici mnoho C++ knihoven. Mezi ty nejvýkonnější patří knihovna *FFTW* [7], která obsahuje obrovské množství optimalizací. *FFTW* disponuje nástrojem *Wisdom*, který na cílové platformě spouští mnoho testů za účelem nalezení co nejefektivnějšího plánu provedení transformace. Tímto způsobem mohou být využity např. SIMD instrukce, pokud je daný procesor má v instrukční sadě. Výsledný plán může být uložen do souboru a být znovu použit později, takže není nutné pokaždé spouštět časově náročné testy. Jinou možností je např. knihovna *KissFFT* [3], která je oproti *FFTW* velmi jednoduchá. Výhodou *KissFFT* je, že podporuje výpočty v číslech s pevnou řádovou čárkou, takže její použití pro jednoduché architektury může být vhodnější. V této práci budou použity obě knihovny a bude srovnán jejich vliv na celkový výkon aplikace.

5.4 Testy pro ověření a vyhodnocení implementace

Použitá data

Nejpravděpodobnější využití radarového procesoru je analýza silničního provozu. Z tohoto důvodu jsem se rozhodl použít radarové nahrávky pořízené v práci [14]. K pořízení těchto nahrávek byl použit radarový modul *K-MC1* od výrobce RFbeam Microwave GmbH. Radar byl umístěn na přechodu pro chodce nad rychlostní silnicí a projíždějící vozidla snímal shora z výšky cca 8 metrů pod úhlem okolo 35 stupňů.

Správnost a přesnost

Správnost a přesnost výpočtů bude kontrolována srovnáním výsledků z referenčních dat procesoru a vzorové implementace v prostředí *Matlab*, kde budou použita čísla s pohyblivou řádovou čárkou s dvojnásobnou přesností (datový typ *double*). Bude sledován rozdíl v přesnosti v závislosti na délce rámce a velikosti FFT v rozsahu od 8 do 4096.

Rychlost

Rychlost zpracování bude měřena aplikací z operačního systému. Jako počátek měření bude zahájení odesílání dat do FPGA a konec bude přijetí posledního výsledku. Obdobně ohraničené bude měření v případě čistě programového řešení. V rámci implementace na FPGA bude porovnán vliv *radix-2* a *radix-4* verze FFT. Pro lepší orientaci bude rychlost měřena v počtu zpracovaných vzorků za sekundu.

Další sledované parametry

Další sledované parametry se budou týkat především vlastností FPGA implementace:

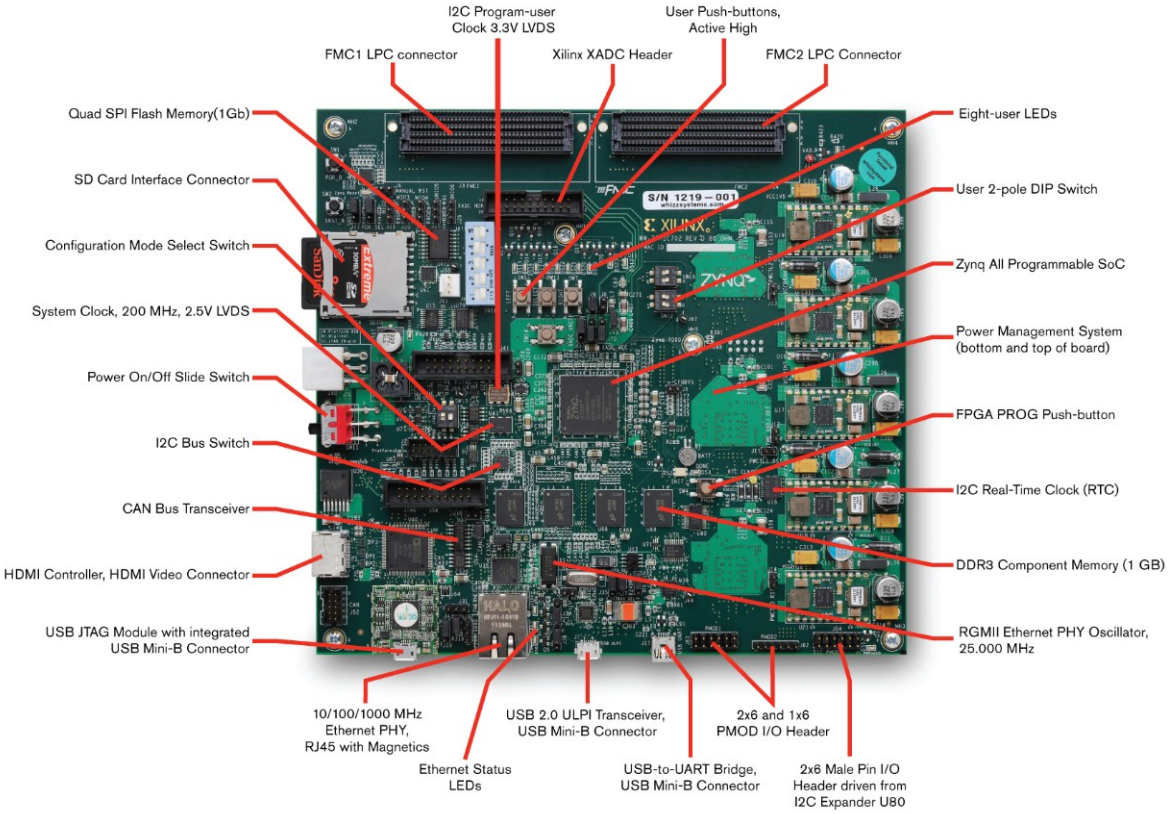
- Množství využitých zdrojů FPGA (počet LUT tabulek, DSP a BRAM bloků) pro různé konfigurace procesoru.
- Maximální dosažitelná frekvence obvodu pro různé konfigurace.
- Latence a propustnost procesoru - společně s předchozí metrikou bude určovat maximální teoretické limity procesoru. Pro tento účel bude použita behaviorální simulace.

Kapitola 6

Implementace

6.1 Použitý vývojový kit

V této práci byl použit vývojový kit Xilinx ZC702. Kromě čipu Zynq obsahuje 1 GB DDR3 RAM paměti, 1000 Mbps Ethernet, USB UART, USB OTG a další, díky čemuž je možné na tomto kitu vyvíjet bez nutnosti připojování dalších součástek nebo zařízení. Programování FPGA je umožněno pomocí rozhraní JTAG, případně může být konfigurace FPGA obvodu uložena na paměťové kartě nebo v integrované paměti typu Flash. Součástí je i USB-JTAG modul, který umožňuje ve vývojovém prostředí Vivado přímo sledovat průběh signálů v FPGA, což výrazně usnadňuje hledaní chyb v navrženém obvodu a ladění.



Obrázek 6.1: Vývojový kit Xilinx ZC702.

6.2 FPGA komponenty

FPGA komponenty byly implementovány v jazyce VHDL. Pro komunikaci mezi jednotlivými bloky byla použita sběrnice AXI Stream, přičemž výstup každého bloku je zpravidla nejprve zapsán do registru pro zlepšení časování. Výjimku z tohoto pohledu představují bloky *spektrální hustota výkonu* a *detekce*, které provádějí proudové zpracování nad celým rámcem z výstupu FFT.

Konfigurační jednotka

Jednotka je připojena k procesoru ARM pomocí sběrnice AXI Lite. Aplikace může zapisovat do konfiguračních registrů, jejichž obsah je pak na vyžádání přenesen do samotných bloků radarového procesoru. Tento způsob, kdy jsou data nejprve přenesena do konfigurační jednotky a až poté jsou šířena do jednotek procesoru, vede k lepšímu časování obvodu. Následující tabulka uvádí kompletní seznam konfiguračních registrů a jejich adresy.

Parametr	Adresa (offset)
iniciace konfigurace	0x0000
délka rámce	0x0004
délka nepřekrývající se části	0x0008
délka překrytí	0x000C
1/délka rámce	0x0010
počet nul k doplnění rámce	0x0014
délka FFT	0x0018
okenní funkce	0x1000 - 0x1fff
detekční prahy	0x2000 - 0x2fff

Tabulka 6.1: Adresy konfiguračních registrů.

Po nastavení všech parametrů je zápisem na příslušnou adresu aktivován samotný proces konfigurace. Ten lze rozdělit do dvou fází – v té první se konfigurují skalární parametry (délka rámce, překrytí, velikost FFT atd.) a ve druhé koeficienty pro okenní funkci a prahovací hodnoty pro blok detekce. Poté, co jsou do jednotek přeneseny všechny parametry, je aktivován resetovací signál, který uvede bloky do výchozího stavu a je vystaven signál `configuration_done`, který povoluje vstup radarových dat. Je tak zaručeno, že data budou zpracována korektně nastavenými bloky.

Překrytí rámců

Tato funkcionalita byla implementována pomocí dvou bloků. První rozděluje proud vstupních vzorků do segmentů o délce nepřekrývající se části rámce a následující blok před tyto segmenty vloží několik posledních vzorků předchozího segmentu – připojí k nim překrývající se část.

Rozdělování proudu vstupních vzorků do segmentů spočívá v periodickém přidávání signálu *TLAST*, který ohraničuje konec rámce. Druhý blok tyto ohraničené rámce přijímá a odesílá je na výstup. Současně se vzorky rámce ukládájí do kruhového bufferu, jehož velikost odpovídá maximální hodnotě překrytí (nastavuje se parametrem v době syntézy). Po přijetí a odeslání posledního vzorku rámce (aktivní signál *TLAST*) je několik posledních vzorků z tohoto bufferu odesláno na výstup, čímž je utvořen začátek následujícího rámce. Délka překrytí rámců v počtu vzorků je konfigurovatelná za běhu.

Bylo nutné zavést speciální stav pro zpracování prvního rámce, protože pro tento rámec nejsou k dispozici poslední vzorky jemu předcházejícího rámce.

Odstranění stejnosměrné složky

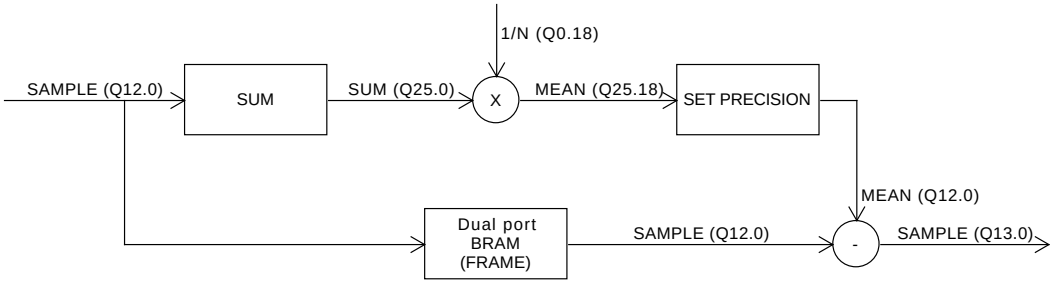
Pro odstranění stejnosměrné složky bylo nutné nejprve spočíst sumu všech vzorků rámce a tu následně vydělit počtem vzorků rámce. To znamená, že celý rámec bylo nutné uložit do paměti. Následně mohly být vzorky z rámce postupně vyčítány a od každého se střední hodnota odečetla.

Velikost registru, do kterého se ukládá suma všech vzorků rámce, je dána vztahem:

$$X = I \ + \log_2 N \tag{6.1}$$

kde X je potřebný počet bitů registru, I je počet bitů na vzorek a N je délka rámce (počet sčítaných hodnot). Tato suma se vynuluje před uložením prvního vzorku rámce a každý vzorek vstupující do paměti je k této sumě přičten.

Po přičtení posledního vzorku je suma kompletní a je potřeba ji vydělit délkou rámce. Situaci však komplikuje fakt, že délka rámce vstupujícího do tohoto bloku nemusí být mocninou čísla dvou. Proto není možné sumu nad všemi vzorky rámce vydělit pomocí bitového posunu vpravo o $\log_2 N$ bitů, kde N je počet vzorků rámce. Protože dělení není v FPGA příliš efektivní, rozhodl jsem se sumu vynásobit hodnotou $\frac{1}{N}$. Tato obrácená hodnota délky rámce je reprezentována ve formátu s pevnou desetinnou čárkou ve tvaru typu $Q0.F$ (všechny bity jsou použity pro desetinnou část), přičemž přesnost v počtu bitů F je stanovena v době syntézy. V dalším kroku je takto spočtená střední hodnota ve tvaru $QX.F$ upravena do tvaru $QX.0$ – bity reprezentující desetinnou část jsou zahozeny a střední hodnota je tak ořezána na celočíselnou hodnotu, která je následně odečítána od každého vzorku rámce.



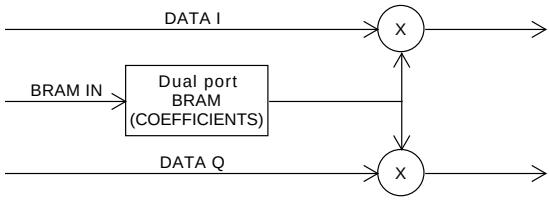
Obrázek 6.2: Blokové schéma odstranění stejnosměrné složky. Pro názornost jsou v závorkách uvedeny ukázkové formáty dat. Schéma je zobrazeno pro jednu složku vstupních dat (I nebo Q).

Jednotka byla implementována tak, že v průběhu vyprázdňování uloženého rámce z paměti je možné nahrávat následující rámec a počítat jeho sumu. Využívá se dvouportové blokové RAM paměti, kde jeden port slouží pouze jako čtecí a druhý pouze jako zápisový.

Je vhodné zmínit, že vzorky rámce mohou být po zpracování tímto blokem kladné i záporné (používá se dvojkový doplněk), zatímco vstupní vzorky jsou vždy kladné.

Okenní funkce

Tento blok je implementován pouze pomocí násobičky a paměti, ve které jsou uloženy koeficienty okenní funkce. Pro indexaci paměti se používá čítač, jehož hodnota se inkrementuje s každým vzorkem rámce a při vynásobení posledního vzorku se vynuluje.



Obrázek 6.3: Blokové schéma okenní funkce. Koeficienty uložené v blokové RAM paměti se používají pro násobení obou komponent (I a Q) vstupních dat.

Pro uložení koeficintů okenní funkce je použita dvouportová bloková RAM paměť, kde jeden port je využit samotnou jednotkou pro čtení koeficientů a druhý port je napojen na konfigurační jednotku, která tímto způsobem může nastavovat koeficinty okenní funkce na požadované hodnoty. Velikost paměti v počtu koeficinetů a jejich bitové šířce (přesnosti) je zvolena v době syntézy a představuje maximální možnou délku okna.

Samotným vynásobením vzorku s okenním koeficientem dochází k navýšení jeho počtu bitů.

Doplnění rámce nulami

Ke každému vstupmínu rámci je přidán nakonfigurovaný počet nulových vzorků. Tento počet může být i nulový a nastavuje se skrze konfigurační jednotku za běhu.

Z hlediska implemetace se jedná o jednoduchý stavový automat s čítačem.

Rychlá Fourierova transformace

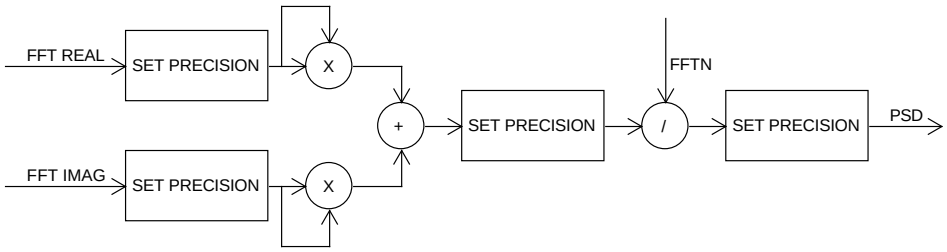
Pro výpočet FFT byl použit Xilinx XFFT IP core. Jednotka obsahuje obsahuje dva vstupní AXI Stream kanály – jeden pro konfiguraci a druhý pro data. Při instanciaci komponenty se nastavují parametry jako maximální možná délka FFT, bitová šířka vstupních dat, architektura (*radix-2*, *radix-4*), zda má být délka FFT konfigurovatelná za běhu a další. Nevýhodou je, že při změně velikosti okenních koeficientů je nutné ručně upravit sířku vstupních dat.

V případě použití čísel s pevnou řádovou čárkou je možné zvolit, zda mají být mezivýsledky škálovány podle předloženého schématu. To zabraňuje tomu, aby měly výsledky větší bitovou šířku než vstupní data, ale hrozí riziko přetečení [23]. V této práci tato funkcionalita nebyla využita, protože se redukce počtu bitů výsledku řeší v bloku *spektrální hustota výkonu*. Výsledky se tak oproti vstupním datům rozšíří o $\log_2(\text{MAX\_FFT}) + 1$ bitů, kde MAX_FFT je maximální podporovaná délka FFT.

Před tento Xilinx XFFT IP core je předřazen blok, který provádí konfiguraci a zarovnání vstupních dat do požadovaného formátu (vstupní data musí být zarovnána na násobek 8 bitů).

Spektrální hustota výkonu

Ze vzorce 5.2 je vidět, že výpočet spektrální hustoty výkonu spočívá v umocnění reálné a imaginární složky FFT koeficientu, jejich sečtení a vydělení délkou rámce.



Obrázek 6.4: Blokové schéma výpočtu spektrální hustoty výkonu.

V průběhu tohoto zpracování se ukázalo jako vhodné redukovat počet bitů komponent. Toho je možné dosáhnout dvěma způsoby – zahozením několika nejméně významových bitů anebo zahozením několika nejvíce významových bitů. První způsob vede ke zmenšení přesnosti výsledku a je možné jej použít vždy. Naopak druhý způsob neovlivňuje přesnost výsledku, ale omezuje jeho možný rozsah. V případě, kdy víme (nebo předpokládáme), že povaha vstupních dat a způsob zpracování omezuje rozsah výsledku na určitý počet bitů, můžeme zbývající horní bity (které budou mít všechny hodnotu 0 nebo 1 podle znaménka výsledku) zahodit, aniž by byla reprezentovaná hodnota jakkoliv ovlivněna. Pokud však náhodou předpokládaný rozsah výsledku nebude dodržen a horní bity se zahodí, hodnota výsledku se tímto zneplatní. Proto se při zahazování horních bitů vždy kontroluje, zda nebyly tyto bity využity a v případě pozitivního zjištění se nastaví chybový bit indikující přetečení. Tento bit je následně šířen s daty a aplikace zpracovávající výsledky je tak informována, že došlo k chybě a může např. radarová data zpracovat jiným způsobem.

Oproti bloku *odstranění stejnosměrné složky* je zde zaručené, že délka vstupního rámce bude vždy mocninou čísla dvou (vlastnost FFT). Tedy dělení je možné realizovat bitovým posunem vpravo o $\log_2 N$ bitů, kde N je délka rámce. Počet bitů, o kolik se výsledek bitově posune, je konfigurovatelný za běhu.

Nastavení rozsahu a přesnosti výsledků a mezivýsledků tohoto bloku se provádí v době syntézy.

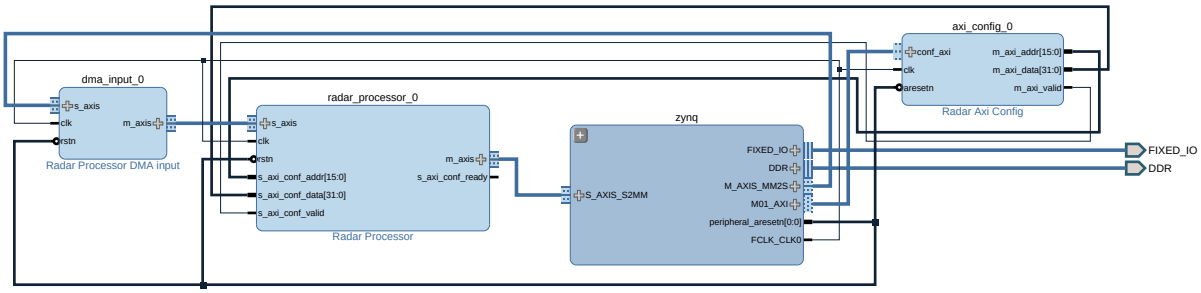
Detekce

Detekce je implementována velmi podobně jako blok okenní funkce. Na rozdíl od násobení vstuních dat s koeficienty je zde prováděno porovnání, zda je vzorek větší nebo roven příslušné prahové hodnotě. Pokud ano, je do dat vzorku přidán indikátor, že došlo k detekci. Tímto způsobem může aplikace zpracovávající výsledky rozpoznat, že došlo k detekci a může i zjistit na kterých frekvenčních složkách byla překročena prahová hodnota.

6.3 Integrace na Zynq

Jednotlivé komponenty byly propojeny do jednoho většího celku, ze kterého byl vytvořen uživatelský IP Core. Ten je v prostředí Vivado možné naimportovat do jiných projektů a blok radarového procesoru připojit do schématu pomocí grafického blokového editoru.

Pro integraci procesoru na platformu Zynq byl vytvořen nový projekt, který radarový procesor zpřístupňuje procesoru ARM přes DMA. Konfigurační rozhraní je připojeno pomocí sběrnice AXI-Lite. Celkové schéma je zobrazeno na obrázku 6.5.



Obrázek 6.5: Schéma zapojení radarového procesoru na čipu Zynq ve vývojovém prostředí Vivado. Blok "zynq"v sobě zahrnuje především jednotku DMA, její napojení na procesorový systém a obvody pro obsluhu periferií.

6.4 Obslužná aplikace

Úkolem obslužné aplikace je konfigurace radarového procesoru, přenos dat do FPGA a čtení výsledků. Tyto funkcionality jsou popsány včetně ukázkových zdrojových kódů v následujících podsekcích. Pro implementaci obslužné aplikace a programové verze řešení byl použit jazyk C++.

Vstup a výstup

Pro čtení radarových nahrávek mi byla poskytnuta třída v prostředí Matlab. Byl implementován skript, který ze souboru s radarovou nahrávkou vyextrahoval jeden kanál a uložil ho v binární podobě do dalšího souboru. Tento binární soubor s jedním kanálem pak slouží jako zdroj dat pro FPGA. Z hlediska výstupu je situace obdobná – výstupní data jsou reprezentována ve struktuře, která kromě samotných výsledků obsahuje informaci o případném přetečení a detekci. Tato struktura může být uložena do binárního souboru. Z hlediska prezentace a vyhodnocení výsledků bylo vhodnější výstupní data zpracovat v prostředí Matlab. Pro tento účel byl implementován Matlabovský načítací skript.

Zápis konfigurace

Provádí se zápisem na fyzickou adresu, na které je umístěn příslušný registr konfigurační jednotky. V operačním systému Linux slouží k tomuto účelu funkce `nmap`, která je schopna namapovat segment fyzického adresového prostoru do uživatelského prostoru aplikace.

```
1  const uint32_t conf_mem_addr = 0x43C00000;
2  const uint32_t conf_mem_size = 1 << 16; // 64 kB
3
4  int conf_mem_fd = open( "/dev/mem", O_RDWR | O_SYNC );
5
6  if (conf_mem_fd < 0)
7  {
8    LOG_error("opening conf mem device failed!");
9  }
10
11 uint32_t * conf_mem_map = (uint32_t *)(mmap(0, conf_mem_size,
12     PROT_READ | PROT_WRITE, MAP_SHARED, conf_mem_fd, conf_mem_addr));
13
14 conf_mem_map[1] = 8;
15
16 close(conf_mem_fd);
```

Zdrojový kód 6.1: Způsob implementace přístupu na fyzickou adresu a zápis do konfiguračního registru. Paměťový segment je přístupný skrze ukazatel na 32 bitová čísla, index 1 (řádek č. 14) tak odpovídá offsetu 4 bajty – podle tabulky 6.2 by v tomto případě bylo zapsáno do registru obsahujícího délku rámce.

DMA přenos dat

Přenos radarových dat pomocí DMA byl realizován poskytnutým ovladačem, který již byl nainstalovaný na vývojovém kitu.

```
1  int dma_fd = open("/dev/axi_dma0", O_RDWR);
2
3  if (dma_fd == -1)
4  {
5    LOG_error("opening dma device failed!");
6  }
7
8  ioctl(dma_fd, START_AXI2MM_DMA, 0);
9  ioctl(dma_fd, START_MM2AXI_DMA, 0);
10
11 // data transfer
12
13 ioctl(dma_fd, STOP_AXI2MM_DMA, 0);
14 ioctl(dma_fd, STOP_MM2AXI_DMA, 0);
15
16 close(dma_fd);
```

Zdrojový kód 6.2: Přístup k DMA zařízení, zahájení a ukončení přenosu.

Pro přenos samotných dat DMA byly použity funkce `read` a `write`

```
1  int bytes_written = write(dma_fd, write_buf, write_size);
2  int bytes_read    = read( dma_fd, read_buf,  max_read_size);
```

Použité nástroje a překlad

Projekt obsahuje skript pro nástroj *CMake*, který jednoduchým způsobem vygeneruje standardní Makefile. Je nutné mít nainstalovanou knihovnu *Boost*. Pro překlad na procesor ARM je potřeba v prostředí *CMake* nastavit odpovídající nástroje (např. z balíku `g++-arm-linux-gnueabihf`). Takto přeložené programy je poté možné spustit na operačním systému Linux přímo na čipu Zynq.

6.5 Programové řešení

V případě programového řešení bylo potřeba vhodně rozdělit výpočet do více vláken, aby bylo možné co nejvíce využít potenciál vícejádrových procesorů. Způsob rozdělení výpočtu do vláken je zachycen na diagramu tříd na obrázku 5.4. Pro implementaci jsem použil třídu `std::thread`, která je součástí standardní knihovny C++ od standardu verze *C++11*. Pro synchronizaci vláken byly použity třídy `std::mutex` a `std::condition_variable`. Vzhledem k tomu není možné použít starší verze kompilátorů, které standard *C++11* nepodporují.

Výběr použité knihovny pro FFT se nastavuje přes *CMake* před kompilací (na výběr je mezi *FFTW* a *KissFFT*). Způsob překladu je stejný, jako pro obslužnou aplikaci.

Kapitola 7

Vyhodnocení

7.1 Přesnost

Implementovaný radarový procesor má konfigurovatelnou výstupní přesnost v počtu bitů. Rozhodl jsem se pro srovnání FPGA implementace s programovou implementací použít 32 bitů na výstupní vzorek (z toho 1 bit byl vyhrazen pro indikaci přetečení a 1 bit pro indikaci detekce), aby velikost výstupních dat byla v obou případech stejná a bylo tak možné objektivněji posoudit dopad použití výpočtu nad čísly s pevnou řádovou čárkou.

Následovalo rozhodnutí, kolik bitů vyhradit pro celočíselnou a desetinnou část. Byly vygenerovány verze s výsledky ve formátu Q30.0, Q28.2, Q26.4 a Q22.8. Na radarových nahrávkách však kromě verze Q30.0 vždy došlo k přetékání. Při testování implementace s uměle vygenerovanými signály se ukázalo, že k přetečení by mohlo dojít při větších délkách rámce i u verze Q30.0. Rozhodl jsem se proto vygenerovat další verze, ve kterých se zahazuje i několik nejméně významových bitů z celočíselné části výsledku. Je tak opět použito všech 30 bitů pro celočíselnou část, ale lze reprezentovat větší hodnoty za cenu další ztráty přesnosti. Výsledky je pak potřeba pro správnou interpretaci vynásobit odpovídajícím faktorem. Např. při zahození 2 nejméně významových bitů celočíselné části je faktor $2^2 = 4$. Vznikly verze, které zahazují 2, 4 a 8 nejméně významových bitů (faktor 4, 16 a 256).

Situaci by šlo řešit i jednodušeji tak, že by se navýšil počet bitů na celočíselnou část. Tímto by ale došlo k nárůstu objemu výsledných dat a srovnání přesnosti oproti datovému typu `float`, použitým v programové implementaci, by nebylo tak objektivní.

Přesnost programového řešení, které počítá nad čísly s posuvnou řádovou čárkou se základní přesností (datový typ `float`), byla srovnatelná s referenčním řešením.

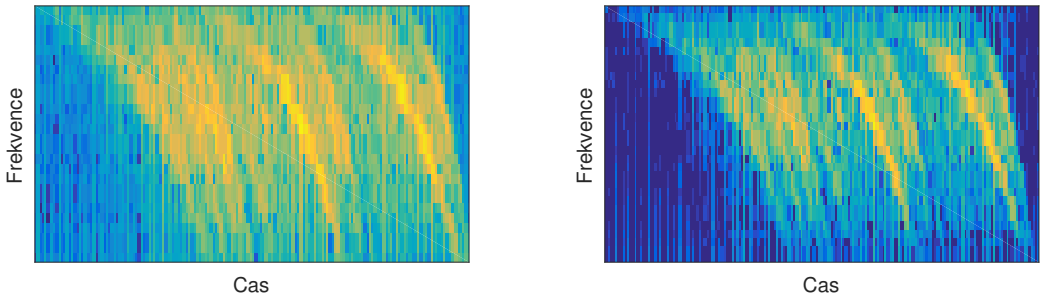
Při vyhodnocování přesnosti jednotlivých implementací jsem se zaměřil na poměr chybového signálu (rozdílu výsledků od referenčního řešení) vůči správným hodnotám. Byl sledován průměrný a maximální dosažený poměr chybového signálu vůči referenčnímu řešení na jeden rámec. Následující tabulka zobrazuje tyto poměry po zpracování signálu bez použití okenní funkce.

Délka rámce	Faktor 1		Faktor 4		Faktor 16		Faktor 256	
	Prům.	**Max**	**Prům.**	**Max**	**Prům.**	**Max**	**Prům.**	**Max**
128	1.65	11.26	3.15	16.89	6.84	37.79	33.16	100.0
256	1.16	8.50	2.25	12.39	5.00	22.38	25.09	96.45
512	0.56	5.95	1.12	9.52	2.56	18.13	13.36	76.87
1024	0.16	0.52	0.32	0.79	0.76	1.50	4.27	8.46
2048	0.13	0.41	0.27	0.63	0.64	1.25	3.66	6.60
4096	0.12	0.35	0.25	0.57	0.60	1.17	3.46	6.26

Tabulka 7.1: Průměrná a maximální energie chybového signálu rámce vůči referenčnímu rešení bez použití okenní funkce. Všechny hodnoty jsou uvedeny v procentech.

Nejlepších výsledků dosahuje verze s faktorem 1 (Q30.0), protože je zde efektivně využito nejvíce bitů. Na druhou stranu je zde největší riziko potenciálního přetečení.

Je možné vysledovat, že se zvyšující se délkou rámce se průměrný i maximální poměr chybového signálu snižuje. Ukázalo se, že toho je zapříčiněno nepřesným výpočtem stejnosměrné složky. V bloku *odstranění stejnosměrné složky* je totiž střední hodnota rámce počítána jako celočíselná, čímž dochází k určité chybě výpočtu. V bloku *spektrální hustota výkonu* je tato chyba ještě navýšena díky umocňování reálné i imaginární složky výstupu FFT. Se zvyšující se délkou rámce se význam této frekvenční složky v celkovém poměru snižuje.



Obrázek 7.1: Srovnání spektrogramů vozidla pro verzi s faktorem 1 (vlevo) a 256 (vpravo) s délkou rámce 512. U verze s faktorem 256 došlo k utlumení či úplněnému potlačení některých nízkoenergetických struktur. Hlavní rysy jsou však stále patrné a pro některé aplikace může být tato přesnost dostačující.

Stejný test byl proveden za použití Hammingovy okenní funkce, přičemž koeficienty byly uloženy na 16 bitech.

Délka rámce	Faktor 1		Faktor 4		Faktor 16		Faktor 256	
	Prům.	**Max**	**Prům.**	**Max**	**Prům.**	**Max**	**Prům.**	**Max**
128	4.34	34.07	7.27	47.34	14.35	73.02	54.85	100.0
256	3.55	23.54	5.79	31.44	11.32	50.87	43.76	105.85
512	2.98	15.56	4.34	19.47	7.71	34.95	28.49	103.44
1024	1.63	7.61	2.11	9.11	3.30	13.04	13.34	44.15
2048	1.20	3.45	1.44	3.77	2.06	4.55	6.93	11.57
4096	0.34	0.92	0.59	1.26	1.22	2.08	5.91	9.07

Tabulka 7.2: Průměrná a maximální energie chybového signálu rámce vůči referenčnímu rešení za použití Hammingovy okenní funkce. Všechny hodnoty jsou uvedeny v procentech.

Aplikace okenní funkce tak vedla k dalšímu snížené přesnosti. Pro reprezentaci spektra se tedy jako efektivnější ukázaly čísla s posuvnou řádovou čárkou, protože výsledné hodnoty mají relativně vysoký dynamický rozsah. U čísel s pevnou řádovou čárkou se musí formát hodnot dimenzovat tak, aby nemohlo dojít k přetečení. Nejlepších výsledků je možné dosáhnout tak, že se na reálných datech nejprve zjistí, při jakém formátu dat ještě nebude docházet k přetékání a následně se zvyšuje počet bitů pro desetinnou část, resp. se snižuje použitý faktor.

7.2 Rychlost

Behaviorální simulace

Nejprve byla vyhodnoce rychlost procesoru v behaviorální simulaci. V tomto případě se nebral v potaz vstup a výstup dat přes DMA. Následující tabulka zobrazuje naměřené výsledky pro implementaci s FFT ve verzi *radix-2*.

Délka rámce	Latence FFT (cyklů)	Latence 1. rámce (cyklů)	Latence dalších rámců (cyklů)	Propustnost @ 100MHz (Msps)
64	477	503	472	13.6
128	885	911	880	14.5
256	1741	1767	1736	14.7
512	3557	3583	3552	14.4
1024	7421	7447	7416	13.8
2048	15637	15663	15632	13.1
4096	33069	33095	33064	12.4

Tabulka 7.3: Změřená latence a propustnost obvodu pro různé délky rámce při použití FFT ve verzi *radix-2*.

Z tabulky je patrné, že celková rychlost zpracování prakticky odpovídá rychlosti FFT jednotky. Latence zpracování následujících rámců je dokonce o několik taktů menší, než udávané zpoždění FFT, protože všechny ostatní bloky jsou zřetězené a jednotka FFT je schopna si v průběhu zpracování rámce dopředu načíst několik dalších vzorků (které ale v této fázi ještě nezpracovává).

Dále je možné si povšimnout, že latence výsledku prvního rámce je vždy o 31 cyklů větší, než latence následujících rámců bez ohledu na délku rámce. Očekával jsem, že tento rozdíl se bude s narůstající délkou rámce zvětšovat, protože blok *odstranění stejnosměrné složky* je implementován tak, že fáze výstupu uloženého rámce a vstup rámce následujícího se mohou překrývat. Tato optimalizace se však bohužel plně nevyužije, protože samotná jednotka FFT není zřetězená a další rámec je začat zpracováván až poté, co je vypočten výsledek nad aktuálním rámcem.

Další tabulka zobrazuje naměřené hodnoty pro FFT ve verzi *radix-4*, která oproti předchozí verzi dosahuje zhruba dvojnásobného zrychlení.

Délka rámce	**Latence FFT (cyklů)**	**Latence 1. rámce (cyklů)**	**Latence dalších rámců (cyklů)**	**Propustnost @ 100MHz (Msps)**
64	272	298	267	24.0
128	507	533	502	25.5
256	891	917	886	28.9
512	1814	1840	1809	28.3
1024	3478	3504	3473	29.5
2048	7345	7371	7340	27.9
4096	14513	14539	14508	28.2

Tabulka 7.4: Změřená latence a propustnost obvodu pro různé délky rámce při použití FFT ve verzi *radix-4*.

Radarové nahrávky

Další test byl zaměřen na vyhodnocení rychlosti zpracování radarových nahrávek. Byla porovnána rychlost programové implementace v jazyce C++ za použití knihovny *FFTW* na více procesorech a rychlost implementace v FPGA.

V případě FPGA implementace samotné měření prováděla aplikace na operačním systému, která data předávala přes DMA do FPGA a stejným způsobem z něj vyčítala výsledky. Radarová data byla nejprve načtena do operační paměti, aby nebyl výsledek zkreslen zpožděním souborových operací.

Při testování nebylo použito překrytí rámců, které výslednou propustnost v odpovídajícím poměru zpomaluje jak v případě programové implementace, tak i v případě implementace na FPGA. Naopak okenní funkce, která byla aplikována, mírně zpomaluje pouze programovou implementaci – na FPGA je okenní funkce aplikována vždy (okenní koeficienty mohou mít hodnotu 1).

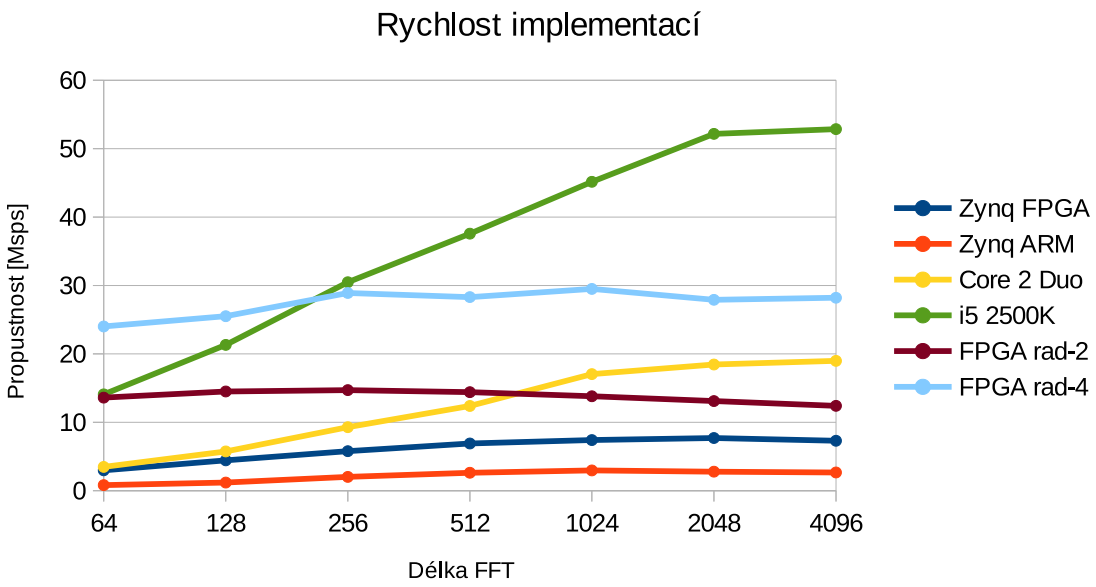
Délka rámce	Intel Core2 Duo @ 2100 MHz	Intel i5-2500K @ 4300 MHz	Zynq ARM @ 666 MHz	Zynq FPGA @ 100 MHz
64	3.5	14.1	0.8	3.0
128	5.7	21.3	1.2	4.4
256	9.3	30.5	2.0	5.8
512	12.4	38.8	2.6	6.9
1024	17.0	45.2	3.0	7.4
2048	18.4	52.2	2.8	7.7
4096	19.0	52.9	2.7	7.3

Tabulka 7.5: Naměřená rychlost zpracování radarových nahrávek různými implementacemi. Všechny hodnoty jsou v jednotkách Msps. Uvedené hodnoty představují průměrnou hodnotu z 5 měření. Pro hodnoty na Zynq (FPGA i ARM) byl rozptyl hodnot do 0.0023 a směrodatná odchylka do 0.0474. Pro zbylé implementace pak byl nejvyšší rozptyl 2.2877 a směrodatná odchylka 1.5125.

V tabulce je uveden pouze jeden sloupec pro FPGA, protože verze *radix-2* i *radix-4* dosahovaly stejných výsledků, které navíc neodpovídají hodnotám zjištěným v behaviorální simulaci. Toto lze vysvětlit pouze omezením propustnosti režií způsobenou přenosem dat přes DMA. Oproti implementaci na procesoru ARM platformy Zynq bylo i tak dosaženo zhruba trojnásobného zrychlení.

Je vidět, že propustnost se s rostoucí délkou rámce zvyšuje jak v případě programové implementace, tak i v případě implementace na FPGA. Na FPGA je to způsobeno tím, že přenos přes DMA je efektivnější pro větší bloky dat (blok odpovídal jednomu rámci). Pro délku rámce 4096 je však vidět malý pokles propustnosti oproti délce rámce rovné 2048 – nejvyšší délka bloku podporovaná DMA byla 2048 a větší bloky musely být rozděleny na více částí. V případě programové implementace zase pro malé délky rámce roste režie spojená se synchronizací vícevláknového výpočtu, kopírováním dat atd.

Následující graf srovnává propustnost jednotlivých implementací včetně hodnot změřených v behaviorální simulaci.



Obrázek 7.2: Změřená rychlost jednotlivých implementací. *FPGA rad-2* a *FPGA rad-4* představují hodnoty změřené v behaviorální simulaci.

Z hlediska schopnosti zpracování radarových dat v reálném čase považuji za vyhovující všechny verze implementace. Pro představu, v práci [14] byla použita radarová data se vzorkovací frekvencí 50 kHz, což odpovídá hodnotě 0.05 Msps.

V případě použití radarového procesoru jako akcelerační jednotky pro procesor ARM je ale žádoucí co nejvyšší propustnost, aby radarové nahrávky mohly být zpracovány v co nejkratším čase. Zde je celková propustnost omezena s nejvyšší pravděpodobností jednotkou DMA, resp. způsobem předávání dat do FPGA a vyčítání výsledků. Pokud by se tento problém podařilo odstranit, bylo by možné dále navýšit propustnost jednotky zvýšením taktovací frekvence pro blok FFT. Tu je možné dle dokumentace zvýšit ze současných 100 MHz až na 250 MHz. Propustnost u verze *radix-2* by se tak zvýšila na hodnotu přes 30 Msps a pro verzi *radix-4* dokonce přes 60 Msps. Radarový procesor by tak využíval dva hodinové signály. Zvyšování taktovací frekvence pro ostatní bloky zpracování by bylo zbytečné.

Srovnání FFT knihoven

V této části je porovnán vliv FFT knihoven na rychlost aplikace při celkovém zpracování.

Délka rámce	KissFFT		FFTW	
	Intel Core2 Duo @ 2100 MHz	Intel i5-2500K @ 4300 MHz	Intel Core2 Duo @ 2100 MHz	Intel i5-2500K @ 4300 MHz
64	3.2	13.6	3.5	14.1
128	5.0	18.9	5.7	21.3
256	8.9	27.6	9.3	30.5
512	10.9	31.4	12.4	38.8
1024	13.9	39.2	17.0	45.2
2048	14.4	39.2	18.4	52.2
4096	14.8	41.2	19.0	52.9

Tabulka 7.6: Srovnání vlivu použitých FFT knihoven na rychlost zpracování.

7.3 Plocha na čipu

Jako první byl vyhodnocen počet alokovaných zdrojů pro samostatný radarový procesor, tedy bez napojení přes DMA na procesorový systém Zynq. Následně bylo množství zdrojů změřeno i po integraci na vývojový kit ZC702.

Samotný radarový procesor

Vzhledem k tomu, že radarový procesor je do velké části konfigurovatelný v době syntézy z hlediska maximálních parametrů zpracování, bylo vyhodnocení provedeno pro více konfigurací. Jednotlivé konfigurace zobrazuje následující tabulka:

Název	Max. délka rámce	Max. překrytí	Verze FFT
1024_rad2	1024	512	radix-2
1024_rad4	1024	512	radix-4
4096_rad2	4096	2048	radix-2
4096_rad4	4096	2048	radix-4

Tabulka 7.7: Vybrané konfigurace pro vyhodnocení množství alokovaných zdrojů.

Dále bylo nastaveno 16 bitů pro šířku okenních koeficientů i pro reciproční hodnotu délky rámce. Výstup byl ve tvaru Q30.0. Následující tabulka ukazuje využití FPGA zdrojů pro tyto konfigurace:

Konfigurace	LUT	LUTRAM	FF	BRAM	DSP
1024_rad2	2248	402	3821	20	26
1024_rad4	4449	810	7152	22	54
4096_rad2	2287	402	3863	31	26
4096_rad4	4483	811	7194	33	54

Tabulka 7.8: Množství alokovaných zdrojů pro jednotlivé konfigurace.

Je možné pozorovat, že verze FFT zásadně ovlivňuje alokované množství všech zdrojů kromě BRAM pamětí. Počet alokovaných BRAM pamětí je naopak velmi závislý na maxi-

mální podporované délce rámce a překrytí, což souvisí s ukládáním zpracovávaného rámce do paměti v některých jednotkách.

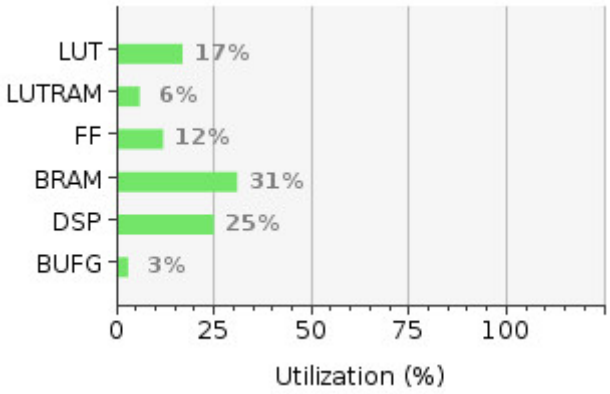
V případě potřeby by bylo možné zredukovat počet využitých DSP bloků, jelikož násobičky i jednotka FFT mohou být implementovány pomocí LUT. Dále je možné dosáhnout úspory zdrojů, pokud by přesnost a rozsah výsledku byly omezeny ještě před jednotkou FFT. V aktuálním řešení se toto řeší až v bloku *spektrální hustota výkonu* a FFT tak vždy počítá s plnou přesností. Toto řešení bylo zvoleno zejména proto, že takto lze snadněji měnit konfiguraci procesoru (v době syntézy pomocí generického parametru bez nutnosti překonfigurovávat jednotku FFT) a dosažení co nejmenší plochy na čipu nebylo v této práci prioritou.

Po integraci na Zynq

Následující tabulka ukazuje využití FPGA zdrojů po integraci radarového procesoru do čipu Zynq (popsáno v 6.3). Pro tento účel byla zvolena největší z testovaných konfigurací.

Konfigurace	LUT	LUTRAM	FF	BRAM	DSP
4096_rad4	8815	1104	13057	43	54

Tabulka 7.9: Množství alokovaných zdrojů po integraci do čipu Zynq.



Obrázek 7.3: Procentuální využití FPGA zdrojů na vývojovém kitu Xilinx ZC702.

Z tabulky je patrné, že přidáním DMA a dalších podpůrných obvodů se plocha celkového obvodu zhruba zdvojnásobila. Přesto však na čipu zbývá spousta volných zdrojů pro potencionální další využití (obrázek 7.3).

Kapitola 8

Závěr

Cílem této práce bylo navrhnout a implementovat radarový procesor v FPGA. V kapitolách 2 a 3 byly shrnuty základní informace o Dopplerově radaru a metodách zpracování radarového signálu. Kapitola 4 popisuje cílovou architekturu Xilinx Zynq a způsob vývoje pro tuto platformu. Dále byl proveden návrh radarového procesoru (kapitola 5) a navržené řešení bylo úspěšně implementováno (kapitola 6). Kapitola 7 pak vyhodnocuje implementaci z hlediska rychlosti, přesnosti a množství zabraných zdrojů na čipu.

Pro výpočet na FPGA byly použity čísla s pevnou řádovou čárkou. To na jednu stranu vede k efektivnímu zpracování na FPGA, ale současně přináší problémy s omezenou přesností a rozsahem výsledků. Tato problematika je popsána v sekci 7.1, kde je přesnost implementace srovnána oproti referenčnímu řešení, které používá čísla s posuvnou řádovou čárkou. Domnívám se, že vhodnost tohoto řešení je závislá především na způsobu zpracování výsledků radarového procesoru – pokud by měl radarový procesor sloužit pouze jako akcelerační jednotka pro procesor ARM čipu Zynq, bylo by pravděpodobně lepší výpočet provádět nad čísly s posuvnou řádovou čárkou. Naopak v případě zpracování výsledků v FPGA považuji současné řešení za vhodné.

Z hlediska rychlosti zpracování dat se implementace ukázala jako naprosto dostačující. Limitujícím prvkem byl přenos radarových dat a výsledků mezi procesorem ARM a FPGA přes jednotku DMA. Přesto je propustnost finální implementace více než 60krát vyšší, než jaká je potřebná pro zpracování radarových dat v reálném čase. Toho by bylo možné využít např. při zpracování rádiového signálu, jelikož použití radarového procesoru nemusí být nutně omezeno jen na radarová data.

Radarový procesor zabírá pouze malou část zdrojů programovatelné logiky. Bylo by tedy možné použít více těchto jednotek na jednom čipu, případně výsledky dále zpracovávat na FPGA nebo zde implementovat jinou funkcionalitu.

Jako možné pokračování práce navrhuji především napojení radarového procesoru přímo na A/D převodník radaru. Výstup z radarového procesoru pak může být dále zpracován v FPGA, např. v případě klasifikace objektů pomocí radaru. Dále by bylo možné vylepšit jednotku *detekce* tak, aby umožňovala režim adaptivního prahování – prahovací hladiny pro jednotlivé frekvenční složky by se tak nastavovaly automaticky podle energie šumu v signálu.

Literatura

[1] *754-1985 - IEEE Standard for Binary Floating-Point Arithmetic*. Institute of Electrical and Electronics Engineers, New York, 1985, iSBN 0-7381-1165-1.

[2] RFbeam Microwave GmbH: K-MC3 RADAR TRANSCEIVER [online]. 2011 [cit. 2016-12-28].
Dostupné z:
http://www.rfbeam.ch/fileadmin/downloads/datasheets/Datasheet_K-MC3.pdf

[3] Borgerding, M.: Kiss FFT [online]. 2013 [cit. 2017-04-17].
Dostupné z: https://sourceforge.net/projects/kissfft/

[4] Camero-Tech: Xaver 100 Handheld, Through-Wall Life Detector [online]. 2015 [cit. 2016-12-25].
Dostupné z: http://www.camero-tech.com/files/files/Xaver100_web.pdf

[5] CMLaboratory: Fast Fourier Transform (FFT) [online]. 2006 [cit. 2017-04-17].
Dostupné z: http://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/transform/fft.html

[6] Crockett, L. H.; Elliot, R. A.; Enderwitz, M. A.; aj.: *The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. Strathclyde Academic Media, 2014, iSBN 978-0-99-297870-9.

[7] Frigo, M.; Johnson, S. G.: *The Design and Implementation of FFTW3*, ročník 93. 2005, 216–231 s., special issue on "Program Generation, Optimization, and Platform Adaptation".

[8] Issakov, V.: *Microwave Circuits for 24 GHz Automotive Radar in Silicon-based Technologies*. Springer, 2010, iSBN 978-3-642-13597-2.

[9] Ivancescu, G.: Fixed Point Arithmetic and Tricks [online]. 2007 [cit. 2017-04-17].
Dostupné z: http://x86asm.net/articles/fixed-point-arithmetic-and-tricks/

[10] Kuo, S. M.; Lee, B. H.: *Real-Time Digital Signal Processing*. John Wiley & Sons, England, 2001 [cit. 2016-29-12], iSBN 0-470-84534-1.

[11] Martinů, L.: *Meření rychlosti s použitím radaru*. Bakalářská práce. Brno, FIT VUT v Brně, 2013.

[12] Maršík, L.: *Algoritmy zpracování signálu v FPGA*. Diplomová práce. Brno, FIT VUT v Brně, 2010.

[13] Maršík, L.: *Accelerated and Embedded Radar Signal Processing Algorithms*. Thesis statement. Faculty of Information Technology, Brno University of Technology, 2012.

[14] Přívara, J.: *Klasifikace objektů s použitím radaru*. Bakalářská práce. Brno, FIT VUT v Brně, 2014.

[15] Skolnik, M.: *Radar Handbook*. McGraw-Hill, New York, třetí vydání, 2008, iSBN 978-0-07-148547-0.

[16] Varshney, L.: Radar Principles [online]. 2002 [cit. 2016-12-24]. Dostupné z: http://www.ifp.illinois.edu/~varshney/cornell/publications/radar%20principles.pdf

[17] Wolff, C.: Radar Tutorial [online]. 2011 [cit. 2016-12-28]. Dostupné z: http://www.radartutorial.eu

[18] Xilinx: 7 Series DSP48E1 Slice, UG479 [online]. 2014 [cit. 2016-12-28]. Dostupné z: http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

[19] Xilinx: 7 Series FPGAs Overview, DS180 [online]. 2015 [cit. 2016-12-26]. Dostupné z: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[20] Xilinx: MicroBlaze Soft Processor Core [online]. 2015 [cit. 2016-12-26]. Dostupné z: http://www.xilinx.com/products/design-tools/microblaze.html

[21] Xilinx: 7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter, UG480 [online]. 2015 [cit. 2016-12-28]. Dostupné z: http://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf

[22] Xilinx: Zynq-7000 All Programmable SoC Overview, DS190 [online]. 2015 [cit. 2016-12-28]. Dostupné z: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[23] Xilinx: Fast Fourier Transform v9.0, LogiCORE IP Product Guide, PG109 [online]. 2015 [cit. 2017-04-17]. Dostupné z: https://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_0/pg109-xfft.pdf

[24] Xilinx: Floating-Point Operator v7.1, LogiCORE IP Product Guide, PG060 [online]. 2015 [cit. 2017-04-18]. Dostupné z: https://www.xilinx.com/support/documentation/ip_documentation/floating_point/v7_1/pg060-floating-point.pdf

[25] Černocký, J.: Zpracování řečových signálů, studijní opora. FIT VUT v Brně. 2006.

Přílohy

Příloha A

Seznam zkratek

Zkratka	Význam	Zkratka	Význam
A/D	Analog to Digital	LUT	Look-Up Table
AMBA	Advanced Microcontroller Bus Architecture	LUTRAM	LUT RAM
APU	Application Processing Unit	MIO	Multiplexed I/O
ARM	Advanced RISC Machine	Msps	Mega Sample Per Second
AXI	Advanced eXtensible Interface	NAND	Not AND
BRAM	Block RAM	NEON	instrukce SIMD pro ARM
CLB	Configurable Logic Block	NOR	Not OR
DDR	Double Data Rate	Q	Quadrature
DFT	Discrete Fourier Transform	QoS	Quality Of Service
DMA	Direct Memory Access	Quad-SPI	Quad-Serial Peripheral Interface
DMIPS	Dhrystone Millions of Instructions Per second	RADAR	Radio Detection And Ranging
DRP	Dynamic Reconfiguration Port	RAM	Random Access Memory
DSP	Digital Signal Processor	RCS	Radar Cross Section
EMIO	Extendable Multiplexed I/O	ROM	Read Only Memory
FF	Flip-Flop	RS232	sériová linka
FFT	Fast Fourier Transform	RTL	Register-Transfer Level
FPGA	Field Programmable Gate Array	RTOS	Real-Time Operating System
FPU	Floating Point Unit	SD	Secure Digital
FSBL	First Stage Boot Loader	SDIO	Secure Digital Input Output
GIC	Generic Interrupt Controller	SIMD	Single Instruction Multiple Data
GPIO	General Purpose I/O	SNR	Signal to Noise Ratio
HDL	Hardware Description Language	Soc	System on Chip
HLS	High Level Synthesis	SPI	Serial Peripheral Interface
I	Inphase	SRAM	Static RAM
I2C	Inter-Integrated Circuit	SSBL	Second Stage Boot Loader
IEEE	Institute of Electrical and Electronics Engineers	TCL	Tool Command Language
IIR	Infinite Impulse Response	UART	Universal Asynchronous Receiver/Transmitter
I/O	Input / Output	U-Boot	Universal Boot Loader
IOB	I/O Block	USB	Universal Serial Bus
IP core	Intellectual Property core	USB OTG	USB On-The-Go
ISE	Integrated Synthesis Environment	VHDL	Very High Speed Integrated Circuit HDL
JTAG	Joint Test Action Group	XADC	Xilinx A/D Converter
LED	Light-Emitting Diode		

Tabulka A.1: Seznam zkratek

Příloha B

Obsah CD

Adresář	**Popis**
`text`	Technická zpráva včetně zdrojových textů
`c++`	Zdrojové kódy pro programovou část v jazyce C++
`fpga`	Zdrojové kódy pro FPGA část v jazyce VHDL
`matlab`	Pomocné skripty pro prostředí Matlab
`data`	Ukázková radarová data

Tabulka B.1: Obsah CD