VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

IMPLEMENTACE HDL MODULU PRO PŘEDZPRACOVÁNÍ DAT Z VÍCEKANÁLOVÉHO ADC

IMPLEMENTATION OF HDL MODULE FOR DATA PREPROCESSING FROM MULTICHANNEL ADC

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Petr Matoušek

AUTHOR

VEDOUCÍ PRÁCE — Ing. Petr Petyovský, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Petr Matoušek ***ID:*** 217431

Ročník: 2 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Implementace HDL modulu pro předzpracování dat z vícekanálového ADC

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout, realizovat a ověřit funkčnost univerzálního HDL modulu pro předzpracování dat z vícekanálového analogově-digitálního převodníku.

1. Seznamte se s vlastnostmi A/D převodníků a metodami pro předzpracování získaných dat.
2. Definujte požadované parametry převodu a zvolte vhodné metody pro předzpracování získaných dat. Navrhněte blokové schéma výsledného řešení, včetně komponenty pro komunikaci s nadřazeným systémem pomocí sběrnice SPI.
3. Prezentujte vlastnosti a parametry jednotlivých komponent navrženého blokového schématu. Definujte parametry protokolu pro komunikaci s nadřazeným systémem.
4. Vyberte vhodný obvod FPGA a vícekanálový A/D převodník.
5. Implementujte zvolené metody předzpracování dat v HDL jazyce.
6. Implementujte komponentu zajišťující komunikaci s nadřazeným systémem.
7. Proveďte ověření funkčnosti implementace jednotlivých komponent pomocí vhodných testů.
8. Demonstrujte na vhodné úloze funkčnost a vlastnosti výsledného řešení.
9. Zhodnoťte dosažené výsledky, uveďte výhody a nevýhody řešení a navrhněte další možná rozšíření.

DOPORUČENÁ LITERATURA:

[1] PINKER, J. POUPA, M: Číslicové systémy a jazyk VHDL. BEN, 2006, ISBN 80-7300-198-5.

[2] ŠŤASTNÝ, J.: FPGA prakticky. BEN, 2011, ISBN 978-70-7300-2.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 15.5.2024

Vedoucí práce: Ing. Petr Petyovský, Ph.D.

Konzultant: Ing. Jaroslav Dohnal, TESCAN ORSAY HOLDING, a.s.

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

ABSTRAKT

Diplomová práce se zabývá návrhem a implementací digitálních filtrů v hradlovém poli typu FPGA. Výsledkem práce je univerzální komponenta v jazyce VHDL, která je znovuvyužitelná v projektech, kde je nutné předzpracovávat data. Navržené zařízení komunikuje s A/D převodníkem, ze kterého filtruje vstupní data pomocí FIR a CIC filtrů v FPGA. Pro okolní svět se chová jako zařízení typu Slave a s nadřazeným zařízením komunikuje pomocí sběrnice SPI. V práci je uveden teoretický popis A/D převodníků, FPGA, digitálních filtrů a zvoleného hardware pro testování. Z praktické části je popsána implementace řešení ve VHDL a testování navrženého řešení na reálné aplikaci. Výstupem práce je VHDL komponenta, která je použitelná v projektech, kde se předzpracovávají data.

KLÍČOVÁ SLOVA

FPGA, VHDL, ADC, SPI, FIR, CIC, předzpracování dat, digitální filtr, TLA2518, MachXO2

ABSTRACT

Master's thesis focuses on designing and implementing digital filters inside FPGA to create versatile VHDL components for data pre-processing. The goal was to develop a reusable solution that efficiently filters input data from an ADC using FIR and CIC filters implemented inside FPGA. Externally, the device operates as a slave component, communicating via the SPI bus for integration into complex data processing systems. Theoretical discussions covers ADC converter fundamentals, FPGA architectures, digital filter theory, and hardware selection. Practical implementation describes VHDL design, optimization for performance, and rigorous real-world testing, including simulation, synthesis, and evaluation with real data inputs. This work produces a VHDL component for data pre-processing, suitable for projects that requires efficient data filtering.

KEYWORDS

FPGA, VHDL, ADC, SPI, FIR, CIC, data preprocessing, digital filter, TLA2518, MachXO2

MATOUŠEK, Petr. *Implementace HDL modulu pro předzpracování dat z vícekanálového ADC*. Diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2024. Vedoucí práce: Ing. Petr Petyovský, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Petr Matoušek
VUT ID autora:	217431
Typ práce:	Diplomová práce
Akademický rok:	2023/24
Téma závěrečné práce:	Implementace HDL modulu pro předzpracování dat z vícekanálového ADC

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Děkuji vedoucímu práce panu Ing. Petru Petyovskému, Ph.D. za konzultace a podnětné návrhy k práci. Dále děkuji panu Ing. Jaroslavu Dohnalovi za odborné vedení práce a společnosti TESCAN GROUP a. s., za poskytnutí prostředků pro realizaci práce.

V Brně dne 15. května 2024

. . . . . .

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Cílem této práce je navrhnout rozšíření v jazyce `VHDL` pro předzpracování dat pomocí digitálních filtrů v hradlovém poli typu FPGA. Předzpracování dat je důležitý proces při kterém je snaha odstranit nebo opravit chyby, nepřesnosti v datech, což je klíčové pro další bezproblémové zpracování dat. Vstupní data pro tuto práci jsou typicky elektrické veličiny získávané prostřednictvím analogově-digitálního převodníku. Práce je koncipována tak, aby navržený HDL modul byl univerzální a znovupoužitelný. Implementace modulu pro předzpracování dat v HDL jazyce, resp. v FPGA je narozdíl od mikrokontroléru výhodná např. v tom, že lze využít paralelního přístupu pro řešení a dosáhnout tak velké propustnosti. Detailněji jsou cíle práce popsané v následující kapitole věnující se podrobnějšímu rozboru zadání.

V práci je popsán teoretický základ o analogově-digitálních převodnících a o metodách pro předzpracování dat, čímž jsou v této práci číslicové filtry. Je uveden teoretický základ o funkci programovatelných hradlových polí a vysvětlení používaného protokolu pro komunikaci zařízení. V práci je teoreticky i prakticky popsána implementovaná komponenta včetně rozboru její funkce a registrů pro její konfigurování. V závěru jsou uvedeny reálné testy komponenty v různých konfiguracích na reálně používané kartě firmy TESCAN GROUP a. s.

Cíle práce

Jak bylo nastíněno v úvodu, tak práce se zabývá předzpracováním dat pomocí digitálních filtrů v programovatelném hradlovém poli typu FPGA. Zadání práce je vytvořeno ve spolupráci se společností TESCAN GROUP a. s, která je jedním z předních světových výrobců elektronových mikroskopů. Z elektroniky, která se uvnitř elektronových mikroskopů nachází, lze analyzovat velké množství dat nejrůznějšího charakteru. Data, která se v této práci vyskytují jako vstupní, jsou typicky elektrické veličiny, zpracované do číslicové podoby pomocí analogově-digitálních převodníků. Zpracovaná data typicky mohou obsahovat šum či parazitní vlivy nejrůznějšího charakteru. Šum je možné odfiltrovat pomocí správně zvolených digitálních filtrů. To lze realizovat jak softwarově (např. na mikroprocesoru), tak i hardwarově (např. na programovatelném hradlovém poli typu FPGA). Obojí má své výhody a nevýhody. Tato práce se zabývá hardwarovou implementací digitálních filtrů v poli typu FPGA, přičemž se bude brát ohled na efektivitu řešení, tak aby bylo možné filtry implementovat i do již existující elektroniky v elektronovém mikroskopu. Cílem práce je implementovat FIR a CIC filtr do programovatelného hradlového pole typu FPGA a prakticky otestovat jejich funkci.

V práci jsou popsány architektury A/D převodníků, čímž se rozumí základní pojmy jako vzorkování a kvantizace a principy funkce základních v dnešní době používaných architektur. Každá architektura se hodí pro použití v konkrétních aplikacích. Stejně tak v případě vývoje pro elektronové mikroskopy se v elektronice nepoužívá pouze jeden typ A/D převodníků.

V práci jsou dále popsány metody pro předzpracování dat. Zejména se jedná o teoretický popis zvolených digitálních filtrů, které jsou v FPGA později prakticky realizovány.

V práci jsou teoreticky popsána programovatelná hradlová pole typu FPGA, zejména jejich architektura, konfigurační technologie a proces návrhu číslicového obvodu. Jsou zde popsána některá úskalí návrhu číslicového obvodu, jako např. problematika hodinových domén. Ta je důležitá pokud spolu mají být propojena dvě zařízení, přičemž každá pracuje s jiným hodinovým kmitočtem. Prakticky to znamená to, že výstup práce je zařízení pracující s vlastním hodinovým kmitočtem, přičemž nadřazené komunikační zařízení má také svůj vlastní hodinový kmitočet. Je vysvětlen koncept předávání dat pomocí AXI-style ready/valid handshake.

Je využíváno sériového periferního rozhraní, což je protokol, který se používá pro komunikaci mezi periferiemi. Jsou zde uvedeny základní vlastnosti rozhraní. Navržený modul, který je výstupem této práce, bude uvedené rozhraní využívat. Konkrétně přes SPI probíhá komunikace s A/D převodníkem a celý modul v programovatelném hradlovém poli se pro nadřazené zařízení chová jako zařízení typu

SPI Slave.

Z praktického hlediska jsou v práci uvedeny prostředky pro realizaci této práce, což znamená použitý A/D převodník, obvod FPGA, přístroj pro komunikaci a generování testovacích signálů pro A/D převodník. Dále je pak uveden i framework pro testování HDL kódu.

V práci je popsán teoreticky i prakticky návrh HDL modulu pro předzpracování, který je realizován v jazyce `VHDL`. Pro modul je uvedeno blokové schéma a rozbor všech jeho entit, včetně popisu jejich funkce a implementace. Jsou uvedeny dostupné konfigurační i stavové registry pro nadřazené zařízení. Jsou uvedeny datové rámce pro komunikaci s nadřazeným zařízením. U některých entit jsou uvedeny i analýzy jejich implementace v FPGA.

Poslední kapitola se zabývá testování HDL modulu na reálné aplikaci. Je testováno několik vybraných konfigurací FIR a CIC filtru, na které je FPGA na kartě nakonfigurováno.

Práce pro společnost TESCAN GROUP a. s je koncipována univerzálně. Výstupem je kód v jazyce `VHDL`, který je přenositelný pro více FPGA. Výstupem práce tedy není jen jedno zařízení, ale univerzální znovupoužitelná komponenta ve `VHDL`, která bude v budoucna součástí jiných nadřazených komponent.

1 A/D převodníky

V následujícím textu jsou uvedeny některé principy funkce, typy a vlastnosti A/D převodníků. Jsou zde popsána v praxi používaná rozhraní pro komunikaci s A/D převodníky.

A/D převodníky se využívají zejména pro převod analogového signálu na signál digitální. Tento digitální signál je dále použitelný v číslicových systémech, které dále získaná data mohou dále zpracovávat, např. digitálními filtry. D/A převodníky se opačně využívají pro získání analogové hodnoty z hodnoty digitální.

1.1 Vzorkování signálu

Vzorkování je proces, pomocí kterého se diskretizuje spojitý signál v časové oblasti. Pro správné vzorkování musí být bráno v úvahu několik aspektů.

Vzorkovací frekvence musí být nejméně dvojnásobkem frekvence původního analogového signálu. Toto je známo jako Nyquistův teorém - rovnice 1.1, který zajišťuje, že původní signál lze správně rekonstruovat z jeho vzorků.

$$f_{vz} > 2f_{max} \tag{1.1}$$

Pokud by vzorkovací teorém dodržen nebyl, tak po vzorkování se složky signálu mohou objevit na různých frekvencích od původního signálu. To může vést k překrývání frekvencí (aliasing) a tudíž nemožnosti rekonstruovat původní signál. Proti alliasingu je třeba použít filtraci (např. typu dolní propust) před vzorkováním.

Použitím převzorkování A/D převodník vzorkuje analogový signál rychlostí výrazně vyšší, než je Nyquistova frekvence. Vzorky mohou procházet procesem tvarování šumu (Noise Shaping), který kvantizační šum přesouvá na vyšší frekvence. Následně je použit filtr typu dolní propust, který zmenšuje šířku pásma signálu a potlačuje vysokofrekvenční složky šumu. Výsledkem je znatelné snížení šumu v požadovaném frekvenčním rozsahu. Poté je často prováděna decimace signálu. Nesprávné vzorkování času může zhoršit poměr odstupu signál šum (SNR). Vzorkování může probíhat synchronně nebo asynchronně s původním signálem. [3]

1.2 Kvantizace signálu

Kvantizace je proces, při kterém se spojitý analogový signál převádí na diskrétní formu tím, že se jeho hodnota zaokrouhlí na jednu z předem stanovených hodnot v závislosti na jeho hodnotě amplitudy. Tento proces je klíčovým krokem při digitalizaci signálů a je základem pro fungování A/D převodníku.

Kvantizační šum je dán nepřesnostmi, které vznikají při tomto procesu. Tento šum je rozdílem mezi skutečnou hodnotou analogového signálu a hodnotou, která byla uložena po kvantizaci. Čím více hodnot je k dispozici pro reprezentaci signálu (tj. vyšší bitové rozlišení), tím menší jsou kvantizační chyby, a tím je i docíleno vyšší kvality digitálního signálu. [3]

Velikost kvantizačního šumu se vyjadřuje v dB a je určena rovnicí 1.2, kdy N je maximální počet kvantizačních úrovní daného A/D převodníku.

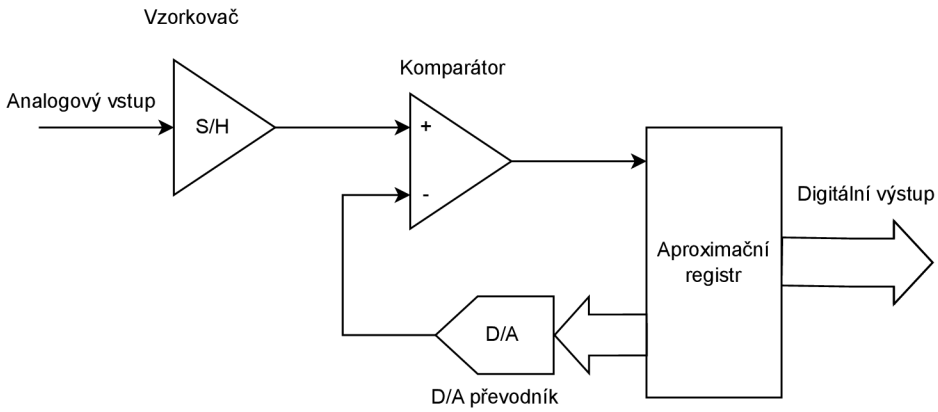
$$SNR = 20 \cdot \log 2^N \approx 6,02 \cdot N \tag{1.2}$$

1.3 Architektury A/D převodníků

V této kapitole jsou stručně popsány základní typy A/D převodníků a stručně vysvětlen jejich hlavní princip.

1.3.1 A/D převodník s postupnou aproximací (SAR)

Typický A/D převodník s postupnou aproximací používá obvod pro vzorkování a vzorkovač signálu, který zaznamená a udržuje okamžitou hodnotu vstupního analogového napětí. Typické blokové schéma SAR je uvedeno na obr. 1.1.



Obr. 1.1: Blokové schéma A/D převodníku s postupnou aproximací

Vnitřní DAC převodník vytváří analogové referenční napětí rovné digitálnímu výstupu vzorkovacího a přidržovacího obvodu. Obě tato napětí jsou přivedena do komparátoru, který výsledek porovnání odešle do A/D převodníku s postupnou aproximací. Tento proces pokračuje n krát po sobě, přičemž n je bitové rozlišení samotného A/D převodníku, dokud není nalezena hodnota nejbližší skutečnému signálu.

A/D převodník s postupnou aproximací nemají žádnou vlastní antialiasingovou filtraci (AAF), takže pokud se nepřidá filtr před A/D převodník, při zvolení příliš nízké vzorkovací frekvence, převodník A/D převodník s postupnou aproximací bude digitalizovat falešné signály (tzv. aliasy). Aliasing je obzvláště problematický, protože jej nelze po digitalizaci opravit. Neexistuje žádný způsob, jak jej softwarově opravit. Musí se mu zabránit buď tím, že se všechny vstupní signály budou vždy vzorkovat rychleji, než je Nyquistova frekvence, nebo filtrováním signálů před vstupem do A/D převodníku a v něm. [8]

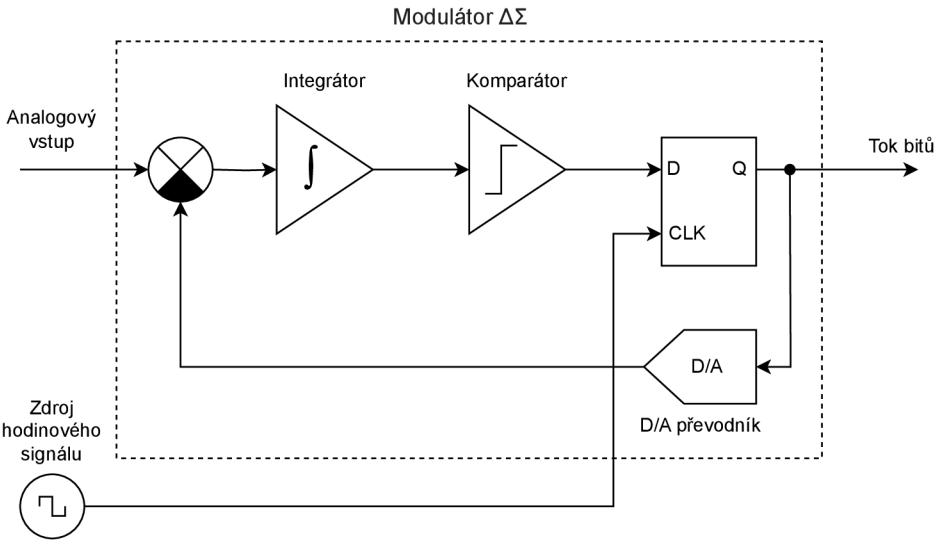
1.3.2 A/D převodník Delta-sigma

A/D převodník Delta-sigma představuje pokročilou variantu A/D převodníku a jeho princip je založen na principu tvarování šumu (noise-shaping). Delta signál přenáší informaci o diferenci vstupního napětí.

Delta-sigma A/D převodník je ideální pro dynamické aplikace, které vyžadují vysoké rozlišení. Proto se běžně využívají v oblastech jako zpracování a analýza zvuku, vibrací a také v aplikacích pro sběr dat. Delta-sigma A/D převodníky jsou hojně využívány v přesných průmyslových měřicích aplikacích. Principiálně eliminují kvantizační šum, což vede k výbornému poměru odstupu signálu od šumu.

Typické blokové schéma Delta-sigma A/D převodníku je znázorněno na obr. 1.2.

Typický $\Delta\Sigma$ modulátor se skládá z diferenčního zesilovače na jehož výstupu je rozdíl vstupního a zpětnovazebního signálu, z integrátoru jehož napětí výstupu roste nebo klesá s rychlostí rovnou výstupu sumátoru, komparátoru, klopného obvodu typu D a jednobitového DAC převodníku pro zpětnovazební přenos na vstup.



Obr. 1.2: Blokové schéma A/D převodníku Delta-sigma ($\Delta\Sigma$)

Delta-sigma A/D převodníky pracují na základě vyššího převzorkování signálu, než je jejich zvolená vzorkovací frekvence. DSP poté zpracovává tato převzorkovaná data a generuje datový tok s vysokým rozlišením a rychlostí. Převzorkování může být až stokrát vyšší než zvolená vzorkovací frekvence. Tento přístup umožňuje dosáhnout velmi vysokého rozlišení (často 24 bitů) a zároveň eliminuje digitalizaci nepřesných signálů pomocí vícestupňového aliasing filtru. To vše však obvykle znamená nižší rychlost ve srovnání s rychlými A/D převodníky s postupnou aproximací. [13]

1.3.3 Integrační A/D převodník

A/D převodníky s dvojitou integrací jsou přesným, ale pomalým typem A/D převodníků. Jejich principem pro přeměnu analogových signálů na digitální je využití integrace. Na začátku procesu A/D převodu je přivedeno analogové napětí, které se nechá postupně integrovat po určitý čas. Poté následuje přivedení známého napětí opačné polarity a tento proces je ještě jednou integrován, ale v opačném směru, dokud nedosáhne zpět nuly. Když dosáhne nuly, systém vypočítá hodnotu vstupního napětí na základě porovnání doby vzestupu s dobou poklesu a znalosti referenční hodnoty.

Tento iterativní proces je spolehlivý, ale zároveň pomalejší a vždy vyžaduje kompromis mezi přesností a rychlostí. Na rozdíl od A/D převodníků typu SAR nebo delta-sigma nemohou A/D převodníky s dvojitou integrací dosáhnout obou těchto parametrů současně. Často se využívají v multimetrech. [8]

1.3.4 Komparační A/D převodník

Komparační A/D převodníky jsou velmi rychlé a pracují prakticky bez zpoždění. Často se používají v případech, kdy je potřeba vysoká vzorkovací frekvence. Princip funkce spočívá v převodu analogových signálů na digitální porovnáváním se známými referenčními hodnotami. Čím více známých referenčních hodnot (komparátorů) se při převodu použije, tím lze docílit vyšší přesnosti. Např. pokud by byl potřeba komparační A/D převodník s 10bitovým rozlišením, musel by se vstupní analogový signál porovnat s 2^{10} známými hodnotami.

Čím vyšší rozlišení je potřeba, tím větší a energeticky náročnější se komparační A/D převodník stává. Z tohoto důvodu je 8bitové rozlišení obecně maximum pro tyto A/D převodníky. Komparační A/D převodníky mohou pracovat v nižších rychlostech GS/s a stále poskytovat 8bitové rozlišení. [8]

2 Vybrané metody předzpracování dat

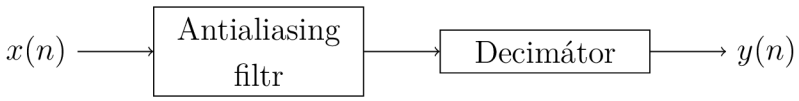
V této kapitole je vysvětlen proces předzpracování získaných dat z A/D převodníku a jsou popsány zvolené metody pro předzpracování dat.

Decimace

Decimace je metoda, kterou lze redukovat množství vzorků signálu a po následné interpolaci na druhé straně přenosového kanálu obnovit původní počet vzorků signálu. Tento postup zajišťuje dokonalou rekonstrukci, pokud frekvenční spektrum užitečného signálu je nižší než vzorkovací frekvence A/D převodníku dělená decimačním koeficientem (číselným násobkem snížení počtu vzorků a datového toku). [9]

Decimace je dobře použitelná tam, kde Nyquistova frekvence signálu je několikrát vyšší než nejvyšší frekvence signálu. Díky decimaci pak lze redukovat vzorkovací frekvenci signálu, čímž lze snížit nároky na zpracování a uchování signálu. Decimační filtry také pomáhají redukovat kvantizační šum po A/D převodu. Před vstupem do decimačního filtru musí být data vyfiltrována pomocí Antialiasing filtru. Pokud by se tak nestalo, tak by mohlo dojít ke změně frekvenčního spektra signálu. [12]

Proces decimace se skládá z antialiasingového filtru a decimátoru. Data $x(n)$ jsou nejdříve přivedena na vstup Antialiasing filtru, kde jsou po filtraci vhodným typem Antialiasing filtru (např. typu dolní propust) vyfiltrována a přivedena na vstup decimátoru, který je nižší vzorkovací frekvencí vzorkuje a vytváří nová data $y(n)$. Diagram decimačního filtru je znázorněno na obr. 2.1.

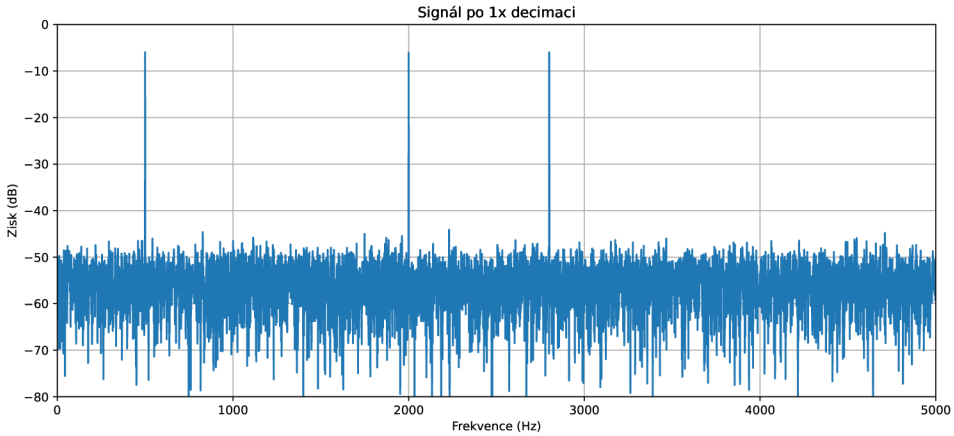


Obr. 2.1: Diagram decimačního filtru

Příklad decimace signálu

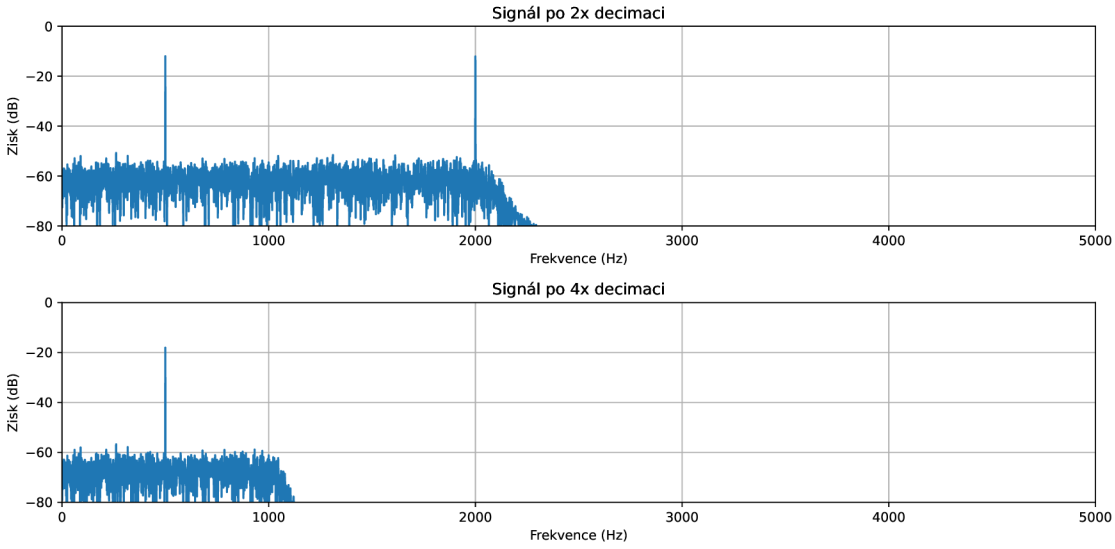
V procesu digitálního zpracování signálu je decimace signálu proces, kdy se z každých N vzorků ponechá pouze jeden a zbývající vzorky jsou zahozeny. Má-li signál vzorkovací frekvenci 3,072 MHz a provedeme decimaci s koeficientem 64, získáme signál s vzorkovací frekvencí 48 kHz. Z Nyquistova teorému vyplývá, že vzorkovací frekvence 3,072 MHz umožňuje rozlišit frekvenční složky s maximální frekvencí 1,576 MHz, zatímco vzorkovací frekvence 48 kHz umožňuje rozlišit frekvenční složky s maximální frekvencí 24 kHz. Při decimaci frekvenční složky mezi 24 kHz a 1,576 MHz nezmizí, ale skládají se do zbývajícího frekvenčního rozsahu od 0 do 24 kHz.

Jako příklad je uveden obr. 2.2 obsahující frekvenční spektrum signálu, který je složen ze tří sinusových vlny s frekvencemi 500 Hz, 2000 Hz a 2800 Hz a bílého šumu. [17]



Obr. 2.2: Původní signál bez decimace [17]

Frekvenční spektra signálů po decimaci jsou patrné z obr. 2.3. Je patrné, že procesem decimace, došlo ke ztrátě informace o signálech na vyšších frekvencích. [17]



Obr. 2.3: Signály po decimaci [17]

2.1 Filtr s konečnou impulzní odezvou (FIR)

Filtry s konečnou impulzní odezvou (FIR) jsou číslicové filtry určené k zpracování různých digitálních signálů přímo na procesoru. Filtrace je realizována číslicově, což znamená provádění programového kódu bez potřeby jakýchkoliv dalších součástek. Tyto filtry pracují s digitálním signálem, což je posloupnost vzorků s určitým bitovým rozlišením (typicky 2^N) po převodu analogového signálu do digitální podoby pomocí A/D převodníku. FIR zkratka znamená Finite Impulse Response, což v překladu znamená, že výstupní odezva má pouze konečný počet nenulových hodnot po aplikaci jednotkového impulzu. Pro jejich praktickou realizaci je zapotřebí procesor nebo FPGA. V analogové technice nemá FIR filtr ekvivalent.

Samotná digitální filtrace signálů není využívána pouze ke změně frekvenčního spektra, ale má také široké využití v jiných oblastech. Lze ji použít k redukci nežádoucího šumu v signálech používaných například pro měření a regulaci. Také se uplatňuje při zvyšování rychlosti přenosu dat nebo při dočasném snížení datového toku s využitím techniky decimace a interpolace.

FIR filtry často nacházejí uplatnění v komplexnějších systémech zpracování signálu, jako jsou adaptivní filtry nebo algoritmy pro potlačení šumu v signálech, kde šum interferuje s požadovaným signálem, např. ve zvukovém prostředí automobilu, kde musí být oddělena řeč od okolního hluku. Jedním z hlavních důvodů použití FIR filtrů jsou jejich vlastnosti, včetně toho, že se jedná o nerekurzivní, lineární, časově invariantní systémy, které jsou vždy stabilní. [9]

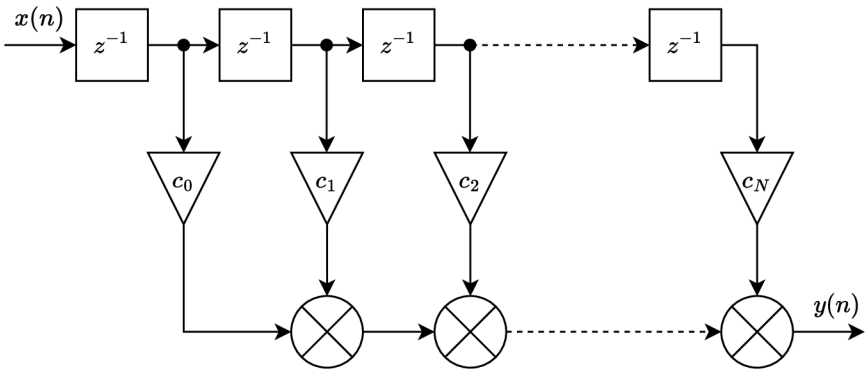
Princip FIR filtru

Pro FIR filtr je třeba určit koeficienty a řád filtru tak, aby splňoval stanovené specifikace pro filtraci. Princip filtru je založen na diskrétní konvoluci mezi jeho koeficienty a vstupem. Konvoluce je operace, kdy se vynásobí prvky na příslušném indexu a následně se jejich součin sečte.

Základní rovnice pro diskrétní konvoluci je znázorněna rovnicí 2.1, kde y je výstup, N je řád filtru, b je pole koeficientů a x jsou vzorkované vstupy. Matematicky lze operaci konvoluce označit pomocí symbolu $*$.

$$y(n) = \sum_{i=0}^{N-1} b(i) \cdot x(n-i) = (b * x)(n) \tag{2.1}$$

Pro lepší názornost lze vyjádřit FIR filtr pomocí bloků v blokovém diagramu, který je uvedený na obr. 2.4.



Obr. 2.4: Blokové schéma FIR filtru

Aplikace FIR filtrů

FIR filtr je univerzální číslicový filtr, který lze modifikovat pro specifickou aplikaci. Vše závisí na nastavení vnitřních koeficientů. Návrh a implementace FIR filtrů se mohou lišit v závislosti na konkrétních požadavcích dané aplikace. Některé praktické aplikace FIR filtru jsou popsány dále v textu.

Filtrace průměrováním

Filtrace průměrováním je jednou ze základních aplikací FIR filtrů. I přes jednoduchost této metody je velmi dobře využitelná pro celou řadu aplikací. Základní rovnice pro výpočet průměru je následovná 2.2.

$$y(n) = \frac{1}{N} \sum_{i=0}^{N-1} x(n-i) \tag{2.2}$$

Z rovnice 2.2 vyplývá, že pro implementaci filtrace pomocí průměru stačí přidělit všem koeficientům FIR filtru hodnotu, která bude rovna převrácené hodnotě řádu filtru (hodnot pro průměrování) N.

Filtrace pomocí průměru se často využívá u dolnopropustních filtrů, pro redukci šumu, příp. pro statistické aplikace. [11]

FIR filtr typu dolní propust

FIR filtr typu dolní propust umožňuje signálům s frekvencí nižší než mezní frekvence projít, zatímco vyšší frekvence jsou potlačeny. Často se používá pro vyhlazování nebo antialiasing.

FIR filtr typu horní propust

FIR filtr typu horní propust umožňuje signálům s frekvencí vyšší než mezní frekvence projít, zatímco nižší frekvence jsou potlačeny. Vysokofrekvenční filtry se často používají k odstranění nízkofrekvenčního šumu nebo k izolaci vysokofrekvenčních složek.

FIR filtr typu pásmová propust

FIR filtr typu pásmová propust umožňuje určitému rozsahu frekvencí projít, zatímco frekvence mimo tento rozsah jsou potlačeny. Pásmové filtry jsou užitečné k izolaci konkrétních frekvenčních složek v signálu.

FIR filtr typu pásmová zádrž

FIR filtr typu pásmová zádrž potlačuje určitý rozsah frekvencí a umožňuje frekvencím mimo tento rozsah propustit. Často se používá k odstranění rušení nebo šumu při konkrétní frekvenci.

Vícepásmový FIR filtr

Vícepásmový FIR filtr kombinuje více filtrů a vytváří filtr s více průchodovými nebo zádržnými pásmy. Umožňuje složitější tvar frekvenční odezvy, splňující konkrétní požadavky ve frekvenčních oblastech.

Lineární fázový FIR filtr

Lineární fázové FIR filtry mají výhodnou vlastnost udržovat konstantní zpoždění napříč všemi frekvencemi. To je dobře využitelné v aplikacích jako např. audio technika nebo zpracování obrazu.

Minimální fázový FIR filtr

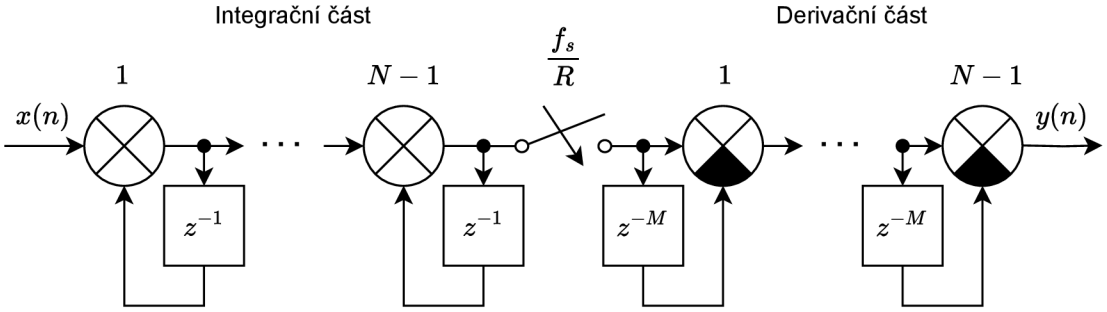
Minimální fázové FIR filtry mají minimální možné posunutí fáze pro danou amplitudovou odezvu. Tyto filtry se často používají v aplikacích, kde je malý a predikovatelný posun fáze.

2.2 Kaskádový filtr s integrátorem

Kaskádové filtry s integrátorem (Cascaded integrator-comb filter (CIC)) se používají pro decimaci nebo interpolaci signálu. CIC rovnou mění poměr vstupní a výstupní vzorkovací frekvence a nelze je použít jen jako filtr, který má na výstupu stejný

počet vzorků jako na vstupu. Narozdíl od FIR filtrů se nenastavují žádné koeficienty a není v jejich případě ani realizována operace násobení, což je pro implementaci v Hardware (zejména pro FPGA) výhodné. Základní struktura CIC filtru se skládá z derivační a integrační části. [16]

Blokové schéma CIC filtru pro decimaci signálu je znázorněno na obr. 2.5.



Obr. 2.5: Blokové schéma CIC filtru

Lze volit počet částí (Stage) filtru pomocí parametru N, čímž se defakto určuje řád CIC filtru. Lze si všimnout umístění decimátoru mezi integrační a derivační částí. Decimátor by bylo možné přesunout i za derivační část - na výstup. To by však přineslo nevýhodu v podobě zbytečného počítání vzorků, které budou stejně zahozeny. Implementace decimátoru uprostřed přináší výhodu v tom, že derivační část (tedy sčítačky resp. odčítačky) jsou využívány pouze $\frac{1}{R}$ celkového času. To přináší výhodu ve sdílení zdrojů, tedy že sčítačky resp. odčítačky lze využívat konkrétně v FPGA pro jinou operaci v daný čas. Rovnice pro vypočítání výstupu z decimačního CIC filtru 1. řádu je následovná 2.3, přičemž konstanta R určuje decimační, nebo interpolační poměr a M je počet zpožděných vzorků se kterými se pracuje.

$$y(n) = y(n-1) + x(n) - x(n-RM) \tag{2.3}$$

3 Programovatelná hradlová pole (FPGA)

V této kapitole jsou popsána FPGA a uvedeny vybrané problematiky při návrhu číslicového obvodu, které se v rámci této práce musí zohlednit. Zároveň tato kapitola slouží jako doplňující text pro vysvětlení závěrečného zhodnocení výsledků implementace při řešení této práce.

Programovatelná hradlová pole (FPGA) představují alternativu k zákaznickým integrovaným obvodům (ASIC). Mezi významné vlastnosti FPGA patří rychlý návrhový cyklus, možnost opakovaného využití (u FPGA využívajících technologii SRAM) a schopnost dynamické změny konfigurace celého FPGA nebo jeho částí během provozu. FPGA konstrukce jsou obecně méně výkonné vzhledem k maximální pracovní taktovací frekvenci a příkonu. [14]

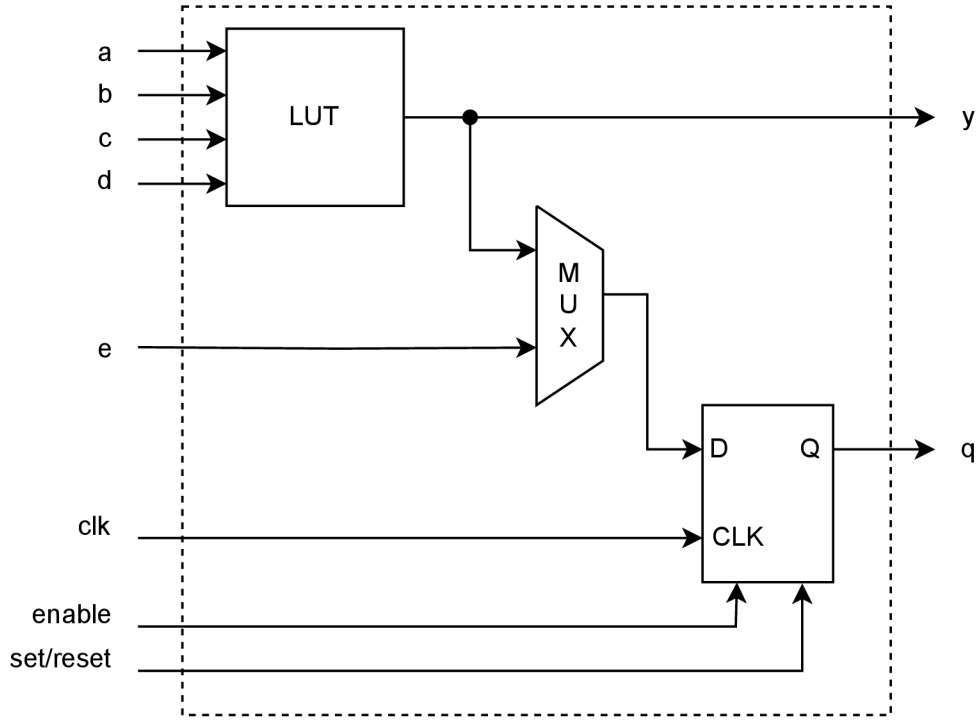
3.1 Architektura FPGA

V této podkapitole je obecně popsána architektura FPGA, popsány programovatelné a vstupně-výstupní logické bloky FPGA a jejich propojovací struktura. Jsou uvedeny konfigurační technologie FPGA. Je vysvětlen proces návrhu konfigurace pro FPGA, včetně procesu syntézy a simulace. Je vysvětlen fenomén hodinových domén v FPGA.

3.1.1 Programovatelný logický blok FPGA

Každé FPGA obsahuje programovatelnou strukturu, kterou lze reprezentovat pomocí programovatelných logických bloků. Logický blok se typicky skládá z pravdivostní tabulky (LUT), multiplexoru a registru. Typické blokové schéma programovatelného logického bloku je uvedeno na obr. 3.1. [15]

LUT tabulka na obr. 3.1 je uživatelsky konfigurovatelná tak, aby plnila jakoukoliv logickou funkci. LUT není explicitně jen 4 vstupá, ale konkrétní počet vstupů se vždy odvíjí od typu FPGA. Za LUT se nachází multiplexor pomocí kterého lze volit vstupní signál do registru (klopného obvodu typu D). Každý registr lze konfigurovat tak aby fungoval jako klopný obvod typu D (řízený hranou) nebo Latch (řízený úrovní). Nicméně používání asynchronní logiky u klopného obvodu typu Latch se při návrhu konfigurace pro FPGA velmi nedoporučuje, kvůli jejich špatné časové analýze. [15]



Obr. 3.1: Blokové schéma programovatelného logického bloku FPGA [15]

3.1.2 Vstupně-výstupní blok FPGA

Je to blok, který propojuje vstupně/výstupní piny a interní propojovací prvky. Obsahuje také některé řídicí obvody, jako jsou pull-up, pull-down, směr vstupu/výstupu, slewrate a open drain. Obsahuje obvod pro udržování hodnot, jako jsou klopné obvody typu D. V komerčních FPGA jsou podporovány různé standardy, jako jsou LVTTL, PCI, PCIe a SSTL, které jsou standardy jednočinných vstupně/výstupních obvodů, LVDS pro diferenční standardy vstupně/výstupních obvodů. [6]

3.1.3 Propojovací struktura FPGA

Skládá se z drátových kanálů, bloků připojení a přepínacích bloků na spojení mezi logickými bloky a mezi logickými bloky a I/O bloky. Kromě stylu, uspořádaného ve vzorku mřížky, existují drátové kanály s hierarchickými strukturami a ty, které tvoří H-stromy. Každý přepínač je programovatelný, a je možné vytvořit libovolnou drátovou trasu pomocí vestavěných drátových zdrojů. [6]

3.2 Konfigurační technologie FPGA

Konfigurování programovatelných logických bloků se provádí pomocí konfiguračních buněk. Typů konfiguračních buněk je několik a liší se podle typu FPGA.

Konfigurační technologie Antifuse

Antifuse je typ buňky, která je vytvořena dvěma kovovými vrstvami vyplněnými amorfním křemíkem. Při průchodu proudu většího než cca 17 mA změní křemík svojí strukturu a stane se vodivým. Antifuse se vyznačuje trvalým naprogramováním, malým odporem přechodu, malými rozměry a úplnou testovatelností při výrobě nebo v programátoru. Její nevýhodou je, že je naprogramována jednou provždy. Naopak potenciální výhodou tohoto způsobu je, že výsledný obvod neobsahuje paměť se souborem konfiguračních bitů (bitstream), což je důležité z hlediska obrany proti tzv. reversnímu inženýrství. Jsou výhodné z hlediska radiační odolnosti pro aerospace aplikace. [14]

Konfigurační technologie Flash

Stejně jako v případě Antifuse jsou buňky založené na technologii Flash energeticky nezávislé. Na rozdíl od technologie Antifuse lze flash buňky podle potřeby přeprogramovat. Nevýhodami je velká náročnost na výrobní technologii, rozměry čipu a jejich nízká spolehlivost. [15]

Konfigurační technologie SRAM

Šest tranzistorů tvoří paměťovou buňku SRAM, která řídí programovatelný spínač, a ten následně nastavuje logický blok nebo topologii propojovacích vodičů. Díky těmto typům buněk lze opakovaně konfigurovat FPGA, u některých typů dokonce za chodu. Funkci buňky lze plně ověřit při výrobě FPGA.

Programovací buňka SRAM zabírá více místa než antifuse. Technologie výroby FPGA sleduje technologii výroby SRAM. Při ztrátě napájecího napětí se u této metody ztrácí informace o konfiguraci, a tak musí být tato FPGA doplněna trvalou, na dodávce energie nezávislou pamětí obsahující soubor konfiguračních bitů (bitstream), který se použije při zapnutí obvodu nebo při obnovení jeho činnosti po výpadku energie. [14]

3.3 Proces návrhu číslicového obvodu

Je nutné zvážit každou fázi návrhu digitálního obvodu. Je nezbytná dobře vytvořená specifikace, která definuje požadované funkce navrženého obvodu. Je třeba vzít v úvahu provozní požadavky, jako je rozsah provozních teplot nebo provozní frekvence. V dalším kroku je navržený systém strukturován do funkčních bloků, kde každý blok obvykle plní jedinečnou funkci, která se liší od ostatních. Tímto způsobem můžete vytvořit srozumitelný, snadno upravitelný a dobře spravovaný kód. K

popisu takového kódu se často používají jazyky pro popis hardware, jako je VHDL, Verilog, SystemC nebo SystemVerilog. Je důležité mít pro každý funkční blok testovací kód (Testbench). Testování by mělo začít u jednotlivých bloků a teprve poté pokračovat celým konceptem, který zahrnuje propojené funkční bloky. Tyto dvě funkce často vykonávají samostatné vývojové týmy, což umožňuje nezávislé testování a odhaluje potenciální mezery v porozumění navrhovaného číslicového obvodu. Pokud jsou funkční simulace úspěšné a navržený koncept funguje podle očekávání, může syntéza kódu pokračovat. V opačném případě je nutné původní kód upravit. RTL (Register-Transfer Level) kód představuje úroveň hardwarového popisu digitálního návrhu používaného k popisu funkčnosti digitálního systému. Tato úroveň se často používá při návrhu hardwaru a popisuje příkazy hardwaru, funkční bloky a jejich vztahy. Během syntézy je RTL kód převeden na propojení logických bloků v hradlovém poli, což vyžaduje definování parametrů časování a přiřazení signálů pinům FPGA. Úspěšná syntéza vytvoří Netlist, což je soubor obsahující propojení logických bloků. Dalším krokem je rozložení a propojení, které optimalizuje logiku a rozložení prvků FPGA. Bloky FPGA jsou vzájemně propojeny. Následuje časová simulace, která zajišťuje splnění požadavků na řešení pomocí přednastavených interních frekvencí signálu. Výstupy simulace časování jsou přesné, protože rozložení logických bloků FPGA je známé, což umožňuje přesné určení časových zpoždění. Úspěšná simulace časování naznačuje, že výsledný koncept bude pravděpodobně správně funkční na skutečném FPGA. Výsledky navrženého modulu mohou zahrnovat počet logických bloků, registrů, požadované I/O piny a maximální frekvenci hodinového signálu komponenty. [7]

3.3.1 Logická syntéza

Logická syntéza je proces, který generuje logický obvod a sekvenční obvod z RTL popisu navrženého objektu. Výsledek logické syntézy se nazývá netlist. Netlist je soubor logických prvků, včetně logických bran a klopných obvodů typu D a propojeních mezi těmito prvky. Mapování technologie je proces, který přiděluje logické prvky v netlistu skutečným logickým prvkům v cílovém FPGA. Mnoho FPGA využívá programovatelné logické prvky nazývané pravdivostní tabulky (LUTs). V současných vývojových prostředích jsou procesy logické syntézy a mapování technologie z velké části automatizovány.

Integrované vývojové prostředí optimalizuje rychlost operací a velikost obvodu v procesech logické syntézy a mapování technologie. V optimalizačním procesu syntézy jsou neměnné vstupy a nepoužívané výstupy z procesu vyjmuty. Vstupy bez zdroje a konfliktní výstupy, pokud existují, mohou způsobit chyby nebo varování ve vývojovém prostředí. Chyba vzhledem k daným specifikacím by mohla být auto-

maticky vyřešena procesem logické syntézy, a tak je správné pečlivě ověřit zprávy a hlášky z vývojového prostředí. Logika může být modifikována v optimalizačním procesu syntézy. Je třeba vzít na vědomí skutečnost, že některé komponenty mohou ve výsledném zdrojovém kódu v konečném netlistu zmizet. [6]

3.3.2 RTL simulace

Je prováděna k ověření, zda se chování navrženého obvodu shoduje s danými specifikacemi. Pro efektivní ladění funkce a chování a správné vyhodnocení výkonu se musí připravit testovací soubor (testbench). Simulační nástroje obvykle zahrnují překladač strojového kódu pro simulaci a simulační software. Překladač analyzuje zdrojový kód a generuje mezistupňový kód pro efektivní provedení simulace. Simulační engine generuje a zpracovává časové posloupnosti událostí testovaného obvodu podle testovacího kódu. Při simulaci lze typicky nastavit čas ukončení, body přerušení, a řídit přerušení. Zprávy z výstupních příkazů ve zdrojovém kódu simulace jsou zobrazeny na konzoli a mohou být uloženy do logovacího souboru. Výsledek simulace je uložen jako časová řada přechodů signálů. Prohlížeč signálů zobrazuje jejich průběh v čase. Lze vybrat signály (objekty) z hierarchie instancí a nastavit reprezentaci pro daný signál (objekt). Příklady reprezentací signálů mohou být: binární, desítkové, hexadecimální, počet číslic. [6]

3.3.3 Rozmístění signálových cest v FPGA

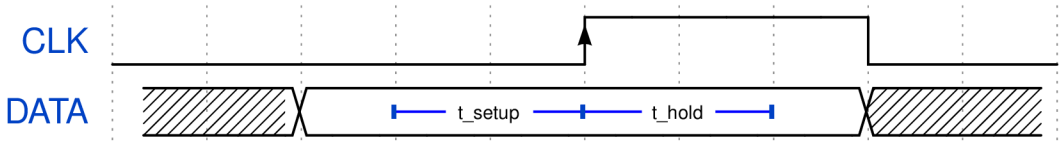
Během procesů rozmístění signálových cest se mapují objekty na zdroje na FPGA. Obvykle se nejprve zpracovávají logické bloky a poté jejich propojení. I když nástroj pro návrh provede umístění za předpokladu odhadu zpoždění šíření signálu, lze se setkat s případy, které blokují trasování v některém spojení a porušují požadavek na zpoždění šíření signálu. V takovém případě musí návrhář znovu zkusit procesy umístění a trasy, změnit parametry pro řízení procesů na základě selhání spojení, aby se vyhnul nežádoucímu zpoždění, a přiřadil vyšší prioritu spojení, které je náchylné k zpoždění. Procesy umístění a trasy vyžadují dlouhou dobu výpočtu. Větší obvod způsobuje menší prostorovou rezervu a nakonec mnohem delší dobu výpočtu. V nejhorších případech to může vést k selhání v trasování. Pokud opakované pokusy o umístění a trasování selžou při hledání proveditelného řešení, tak návrhář musí zkoumat možnosti optimalizace, přepsat zdrojový kód s vylepšenou architekturou a algoritmem nebo nahradit cílové FPGA zařízení jiným. [6]

3.3.4 Hodinové domény v FPGA

Často se při návrhu číslicového obvodu lze setkat s přechody mezi více hodinovými doménami. Tuto problematiku je nutno správně řešit tak, aby v návrhu nedocházelo k problémům se stabilitou FPGA, či přijímání metastabilních signálů.

Podmínky nutné pro eliminaci metastabilního signálu

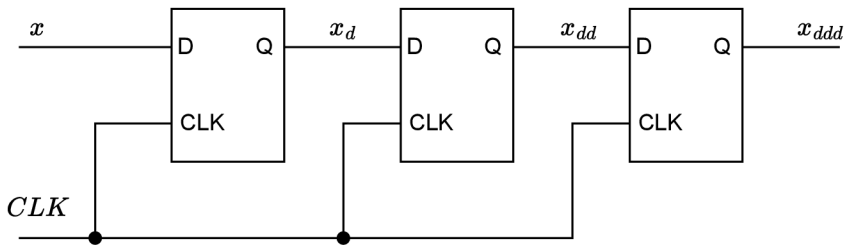
Metastabilitou se rozumí stav, kdy nelze určit, ve kterém logickém stavu se signál nachází. Nastává při nesplnění časových podmínek (označovaných jako t_{setup} a t_{hold}), které definují, jak dlouho musí být signál neměnný před a po překlopení klopného obvodu. Pro lepší názornost je uveden obr. 3.2.



Obr. 3.2: Metastabilita signálu

Použití klopných obvodů typu D

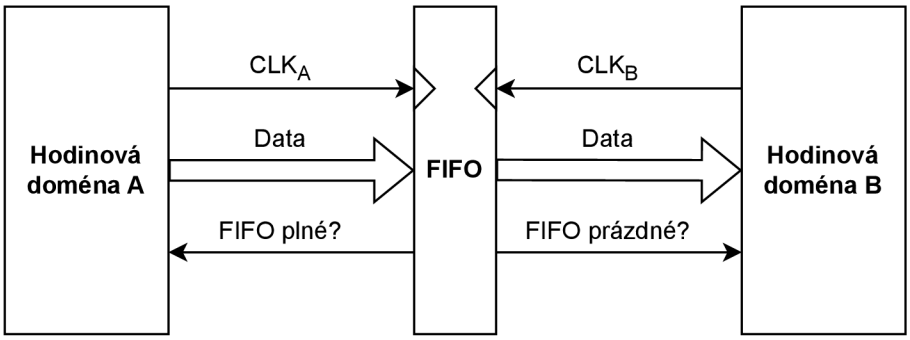
Nejjednodušší řešení pro přechod mezi hodinovými doménami je použití dvou nebo tří klopných obvodů (KO) typu D. Pro názornost je uvedeno jednoduché schéma propojení 3.3, kde x je vstup z jiné hodinové domény a x_{ddd} je výstup v aktuální hodinové doméně. Takové řešení se nejvíce hodí pro řešení, kdy je k dispozici hodinový kmitočet vyšší, než jakým byl vzorkován přijímaný signál.



Obr. 3.3: Použití klopných obvodů typu D pro přechod mezi hodinovými doménami

Použití datového typu fronta (FIFO)

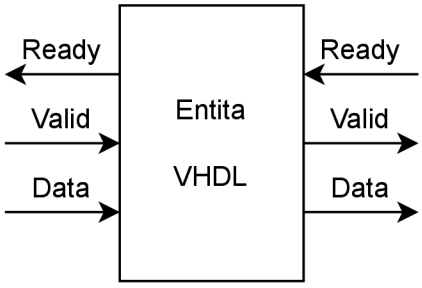
Složitějším, ale za to více robustnějším řešením je využití datového typu fronty (FIFO). Pro zápis dat do FIFO lze využít hodinový signál z původní domény a pro čtení z FIFO lze využít hodinový signál z hodinové domény, kam signál přenášíme. Takové řešení se hodí pro signály, které mají velkou šířku. Navíc narozdíl od použití klopných obvodů typu D lze toto řešení univerzálně použít na přechod mezi jakkoliv rychlými doménami bez ztráty dat. Blokový nákres použití FIFO mezi hodinovými doménami je na obr. 3.4.



Obr. 3.4: Použití FIFO pro přechod mezi hodinovými doménami

3.3.5 Předávání signálu pomocí AXI-style ready/valid handshake

AXI-style ready/valid handshake je jedním ze způsobů pro výměnu dat mezi zařízeními. Entity mezi sebou musí pracovat synchronně, a číst signál na stejnou hranu signálu. Nelze jej použít pro přechod mezi hodinovými doménami. Rozhraní se skládá ze tří signálů: `Ready`, `Valid` a `Data`. Nákres entity se signály AXI-style ready/valid handshake je uveden na obr. 3.5. Přenos probíhá pouze právě pokud jsou signály `Ready` a `Valid` v log. 1. [18]



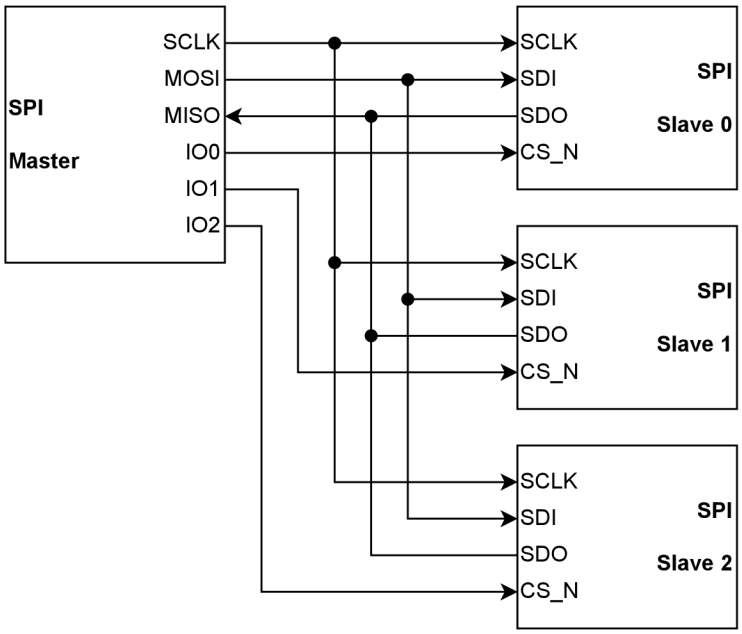
Obr. 3.5: Nákres entity pro demonstraci předávání signálu pomocí AXI-style ready-/valid handshake

4 Sériové periferní rozhraní (SPI)

V této kapitole je popsán princip a vlastnosti komunikace pomocí SPI. SPI je typ komunikace, který se využívá pro komunikaci mezi zařízeními a jejich periferiemi. Periferiemi mohou být senzory, A/D převodníky a další. V textu je pro zařízení řídící komunikaci používán výraz Master a pro zařízení odpovídající na sběrnici výraz Slave. Tyto pojmy nelze přímo nahradit politicky korektnější alternativou jako např. Leader a Follower, nebo Initiator a Follower, protože přímo z definice SPI jsou signály označeny jako Master nebo Slave.

4.1 Specifikace rozhraní SPI

Rozhraní SPI se typicky skládá z Master a Slave, přičemž Master vždy řídí komunikaci. SPI pracuje v synchronním režimu komunikace, což znamená, že Master i Slave pracují ve stejné hodinové doméně s jedním společným hodinovým signálem. Typicky se zařízení pro komunikaci po SPI propojují čtyřmi vodiči - SCLK (hodinový signál), CS_N (signál pro výběr zařízení), MOSI (Master Out Slave In - datový signál), MISO (Master In Slave Out - datový signál). Typické propojení Master a tří Slave zařízení je znázorněno na obr. 4.1.



Obr. 4.1: SPI zapojení Master a Slave zařízení

SCLK

SCLK je hodinový signál, který generuje Master a slouží k synchronizaci komunikace. Tento signál určuje rychlost přenosu dat mezi Master a Slave. Data jsou vzorkována nebo zapisována na náběžnou nebo sestupnou hranu signálu SCLK (podle režimu komunikace SPI), což zajišťuje synchronní a řízený přenos dat.

CS_N

CS_N je signál, který Master používá k označení konkrétního zařízení Slave, se kterým bude komunikovat. Aktivací tohoto signálu Master informuje vybraného Slave o začátku komunikace a deaktivací signalizuje její konec. Tímto způsobem se identifikuje, které Slave zařízení bude účastníkem aktuální komunikace.

MOSI

MOSI je výstupní signál z Master a vstupní signál pro Slave. Master tímto signálem posílá data, která jsou následně přijímána a zpracovávána Slave zařízením. Tímto způsobem je zajištěn přenos informací z Master na Slave.

MISO

MISO funguje opačně než MOSI. Je to výstupní signál ze Slave a vstupní signál pro Master. Slave na tento pin zapisuje data, která jsou následně čtena Master zařízením. Tímto způsobem lze umožnit přenos informací ze Slave zpět k Master.

Uvedené signály společně tvoří kompletní rozhraní pro SPI komunikaci mezi Master a Slave. Synchronizace pomocí hodinového signálu, posílání dat signály MOSI a MISO, a výběr konkrétního Slave zařízení pomocí CS signálu jsou klíčové vlastnosti komunikace SPI.

4.2 Komunikační režimy sběrnice SPI

SPI podporuje čtyři režimy. Každý režim je charakterizován nastavením dvou parametrů, a to Clock Polarity (CPOL) a Clock Phase (CPHA). Parametr CPOL určuje úroveň hodinového signálu (SCLK) v klidovém stavu, kdy CPOL = 0 znamená klidovou úroveň 0 a CPOL = 1 znamená klidovou úroveň 1. Parametr CPHA určuje, kdy jsou data platná a kdy jsou čtena ve vztahu k hranám SCLK. Režimy SPI jsou klíčové pro specifikaci synchronizace datového přenosu mezi Master a Slave v systému. Pro lepší názornost režimů komunikace SPI je uveden obr. 4.2.

SPI Mode 0

CPOL = 0 a CPHA = 0 Data jsou platná na náběžné hraně SCLK a klidová úroveň SCLK je v log. 0.

SPI Mode 1

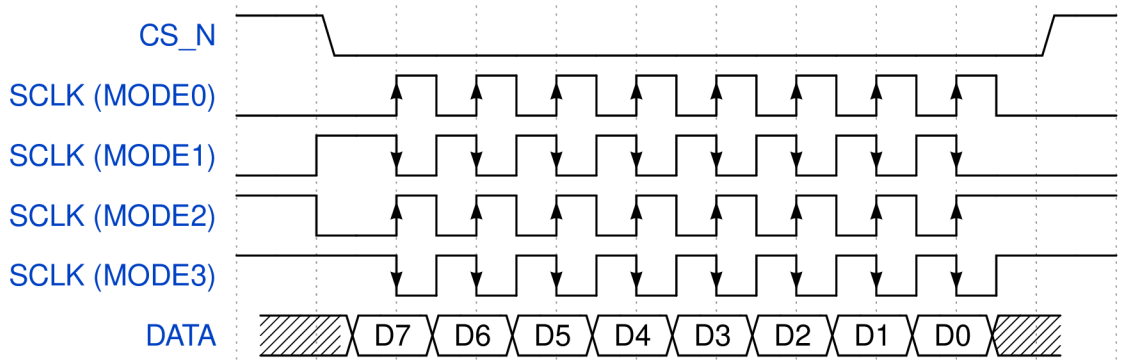
CPOL = 0 a CPHA = 1 Data jsou platná na sestupné hraně SCLK a klidová úroveň SCLK je v log. 1.

SPI Mode 2

CPOL = 1 a CPHA = 0 Data jsou platná na náběžné hraně SCLK a klidová úroveň SCLK je v log. 1.

SPI Mode 3

CPOL = 1 a CPHA = 1 Data jsou platná na sestupné hraně SCLK a klidová úroveň SCLK je v log. 0.



Obr. 4.2: Režimy komunikace SPI

4.3 Průběh komunikace SPI

V rámci SPI probíhá komunikace na základě synchronní sériové metody přenosu dat, kdy informace putují bit po bitu. Celý proces začíná aktivací CS signálu Master zařízením pro vybrané Slave, což oznamuje začátek komunikace a specifikuje, s kterým Slave má Master komunikovat.

Během přenosu dat každý bit datového slova putuje mezi zařízeními. U Master jsou data postupně posílána, začínající nejčastěji nejvýznamnějším bitem (MSB) a končí nejméně významným bitem (LSB). Na straně Slave se přijímání dat děje v

opačném pořadí, kde první je přijímán MSB a následuje postupné zaznamenávání jednotlivých bitů.

SPI podporuje full-duplex komunikaci, což umožňuje Master posílat data Slave a současně přijímat data od Slave.

Komunikace je zakončena deaktivací CS signálu, což signalizuje konec komunikace s daným Slave zařízením. Celkově lze konstatovat, že SPI přináší efektivní sériovou komunikaci s využitím synchronizovaného hodinového signálu, což umožňuje přesný a rychlý přenos dat bit po bitu mezi Master a Slave.

4.4 Praktické aplikace použití SPI

Použití a aplikace sériového periferního rozhraní (SPI) jsou široké. SPI se běžně využívá v elektronických zařízeních pro propojení mikrokontrolérů s různými periferními zařízeními, jako jsou senzory, displeje, a další. Toto rozhraní nachází uplatnění v embedded systémech a IoT (Internet of things) zařízeních, kde slouží pro efektivní komunikaci mezi různými čipy a moduly.

Příklady konkrétních aplikací zahrnují použití SPI pro čtení a zápis dat na SD karty v digitálních fotoaparátech, mobilních telefonech a jiných zařízeních. Dále se využívá pro komunikaci s barevnými LCD displeji.

V oblasti senzorů a akcelerometrů se SPI používá pro přenos dat v chytrých zařízeních a robotice. Taktéž slouží k vzájemné komunikaci mezi mikrokontroléry v rámci jednoho systému. V komunikačních modulech, jako jsou WiFi nebo Bluetooth moduly, umožňuje SPI bezdrátovou komunikaci v elektronických zařízeních.

4.5 Výhody a nevýhody použití SPI

SPI nabízí rychlý přenos dat, jednoduchost implementace a možnost full-duplex komunikace. Na druhé straně, počet CS signálů může omezit počet připojitelných slave zařízení, a SPI je navrženo pouze pro Master-Slave topologii, což může být omezením v komplexnějších systémech. Přesto je SPI široce využíváno tam, kde je potřeba rychlá a spolehlivá sériová komunikace. Nevýhodou SPI může být to, že se jedná o synchronní přenos dat, což se příliš nehodí na dlouhá spojení. Při komunikaci musí dojít signál ze Slave do Master ve specifikovaném čase, což se v případě dlouhých spojení nemusí stát.

5 Prostředky pro realizaci práce

V této kapitole jsou stručně popsány komponenty používané pro praktickou realizaci práce. Je detailněji popsána karta firmy TESCAN, A/D převodník TLA2518 včetně vysvětlení principu komunikace. Dále jsou uvedeny technické parametry FPGA MachXO2, zařízení Digilent ADP3250 Analog Discovery Pro. V závěru je pak uveden používaný framework pro testování `VHDL` kódu VUnit.

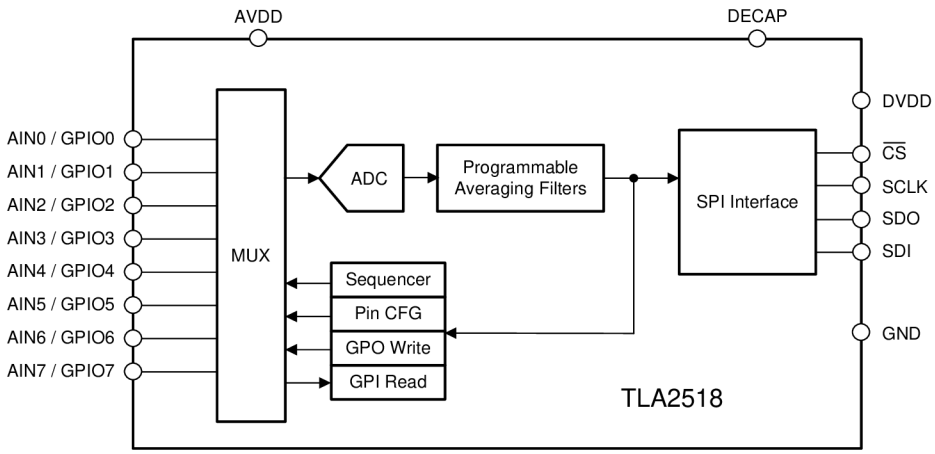
5.1 Popis hardwarové karty firmy TESCAN využité při testovaní implementace této práce

Karta slouží jako ovládací karta pro polovodičový detektor, což je zařízení používané k detekci a měření elektromagnetického záření, jako jsou rentgenové paprsky, gama záření, nebo světlo. Princip fungování polovodičového detektoru spočívá v absorpci energie záření v polovodičovém materiálu, což vytváří elektron-děrové páry. Tyto páry jsou poté zachyceny elektrickým polem uvnitř detektoru a vytvářejí elektrický signál, který lze detekovat a měřit. [21]

Pro tuto práci je relevantní nejvíce to, že se na kartě nachází FPGA MachXO2 a A/D převodník TLA2518 a bude se na kartě testovat navržená implementace. Na kartě jsou vyvedeny debug piny pro napojení logického analyzátoru přímo na piny FPGA a je i vyveden kanál č. 7 A/D převodníku TLA2518 pro generování testovacího signálu z generátoru, což je pro testy implementace velmi výhodné.

5.2 A/D převodník TLA2518

Pro testování je použit A/D převodník od Texas Instruments s označením TLA2518. TLA2518 je 8 kanálový A/D převodník s postupnou aproximací (SAR) s rozlišením 12 bitů. Blokové schéma A/D převodníku TLA2518 je uvedeno na obr. 5.1. Z blokového schéma je patrné, že 8 vstupů do A/D převodníku je multiplexováno a A/D převod probíhá vždy jen pro jeden z nich. Rozhraní pro komunikaci s komponentou a vyčítání dat po A/D převodu probíhá pomocí komunikace SPI. [4]



Obr. 5.1: Blokové schéma A/D převodníku TLA2518 [4]

Pro lepší přehlednost parametrů A/D převodníku je uvedena tab. 5.1. Volba A/D převodníku je důležitá pro pozdější implementaci v FPGA, kde je třeba implementovat rozhraní takové, aby správně pracovalo s konkrétním typem převodníku. Komunikační rozhraní se u různých převodníků často liší.

Tab. 5.1: Přehled parametrů A/D převodníku TLA2518 [4]

Parametr	**A/D převodník TLA2518**
Typ převodu	S postupnou aproximací
Rozlišení	12bitové
Vzorkovací kmitočet	1 Msps (Mega sample per second)
Počet vstupních kanálů	8
Způsob komunikace	SPI (rychlost komunikace až 60 MHz)

Komunikační rozhraní A/D převodníku TLA2518

Pomocí SPI komunikace se přistupuje do 8bitových registrů. Registry jsou uvedeny v tab. 5.2. Bližší popis registrů a podrobnější vysvětlení funkce lze nalézt v literatuře [4].

Tab. 5.2: Registry A/D převodníku TLA2518 [4]

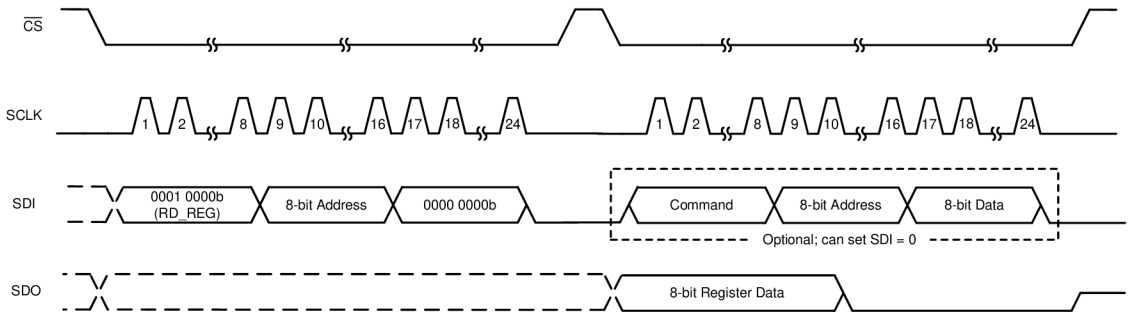
Adresa	Popis registru
0x00	SYSTEM_STATUS
0x01	GENERAL_CFG
0x02	DATA_CFG
0x03	OSR_CFG
0x04	OPMODE_CFG
0x05	PIN_CFG
0x07	GPIO_CFG
0x09	GPO_DRIVE_CFG
0x0B	GPO_VALUE
0x0D	GPI_VALUE
0x10	SEQUENCE_CFG
0x11	CHANNEL_SEL
0x12	AUTO_SEQ_CH_SEL

Zápis do registru TLA2518

Pro zápis do registru je odesílán 24bitový rámec po SPI. Prvních 8 bitů obsahuje informaci o zápisovém příkazu (`No operation`, `Single register read`, `Single register write`, `Set bit`, `Clear bit`). Následuje 8bitová adresa registru (uvedeno v tab. 5.2. Posledních 8 bitů obsahuje samotná posílaná data. Takto je správně provedena komunikace s TLA2518. [4]

Čtení z registru TLA2518

Čtení z registru TLA2518 se skládá ze zápisu (správně nastavený zápisový příkaz určuje čtení) a z druhého 24bitového rámce, který data čte. Pro lepší názornost je uveden obr. 5.2. [4]



Obr. 5.2: Čtení z registru TLA2518 [4]

5.3 FPGA MachXO2 CMXO2-2000HC

Jako FPGA pro realizaci této práce je použit LCMXO2-2000HC. Jedná se o malé FPGA výrobce Lattice Semiconductor z rodiny obvodů MachXO2. Některé klíčové vlastnosti tohoto FPGA: [5]

- Počet logických bloků: 2112
- Maximální provozní frekvence: 269 MHz
- Distribuovaná RAM: 16 kbit
- Integrovaný blok RAM - EBR: 74 kbit
- Počet vstupně-výstupních pinů: 79

Jako výchozí prostředí pro syntézu a implementaci kódu pro tuto práci bude používáno vývojové prostředí Lattice Diamond 3.13. Syntéza kódu je prováděna pomocí Synplify Pro.

5.4 Zařízení pro testování Digilent ADP3250 Analog Discovery Pro

Zařízení Digilent ADP3250 Analog Discovery Pro je dvoukanálový osciloskop. Má 14bitové rozlišení a vzorkovací kmitočet až 0,5 GS/s. Obsahuje 16 digitálních kanálů. Lze jej použít jako logický analyzátor, nebo pro generování definovaných digitálních signálů (SPI, I2C, UART). Pro komunikaci se zařízením se používá program Waveforms. Některé vlastnosti zařízení: [20]

- 16 digitálních kanálů
- 2-kanálový generátor libovolných analogových signálů
- Maximální vzorkovací frekvence: 0,5 GS/s analogový a 125 MS/s digitální

5.5 Framework pro testování VHDL kódu VUnit

VUnit je open-source nástroj navržený pro provádění Unit testů pro VHDL nebo SystemVerilog. Poskytuje potřebnou funkcionalitu pro realizaci kontinuálního a automatizovaného testování HDL kódu. Zvyšuje efektivitu vývoje možností rozdělení rozsáhlých testbench na menší a nezávislé testy, čímž může zlepšit kvalitu a rychlost testování projektů umožněním spouštění rozsáhlého množství testů z jednoho skriptu.

Ověřovací metodologie je plně univerzální a není nijak omezena. Výhody VUnit lze využít při psaní testů jak na začátku, tak na konci vývoje, ať už se jedná o dlouhodobé testy či krátkodobé jednotkové testy. Unit umožňuje testy spouštět vícejádrově, což je velká výhoda při provádění velkého množství různých testů v jednom čase. [19]

Struktura testovacího skriptu VUnit

Ve struktuře testovacího skriptu v jazyce Python je potřeba vytvořit nový objekt pro VUnit. Je potřeba importovat soubory z VHDL a skript spustit. Lze nastavit i ostatní parametry jako nastavení generické proměnné, nebo načtení Waveform souboru pro Modelsim apod. Jednoduchý příklad skriptu je následovný: [19]

```
1 from vunit import VUnit
2 vu = VUnit.from_argv()
3 lib = vu.add_library("lib")
4 lib.add_source_files("*.vhd")
5 vu.main()
```

Výpis 1: Příklad testovacího skriptu v Python (framework VUnit)

Struktura testbench VUnit

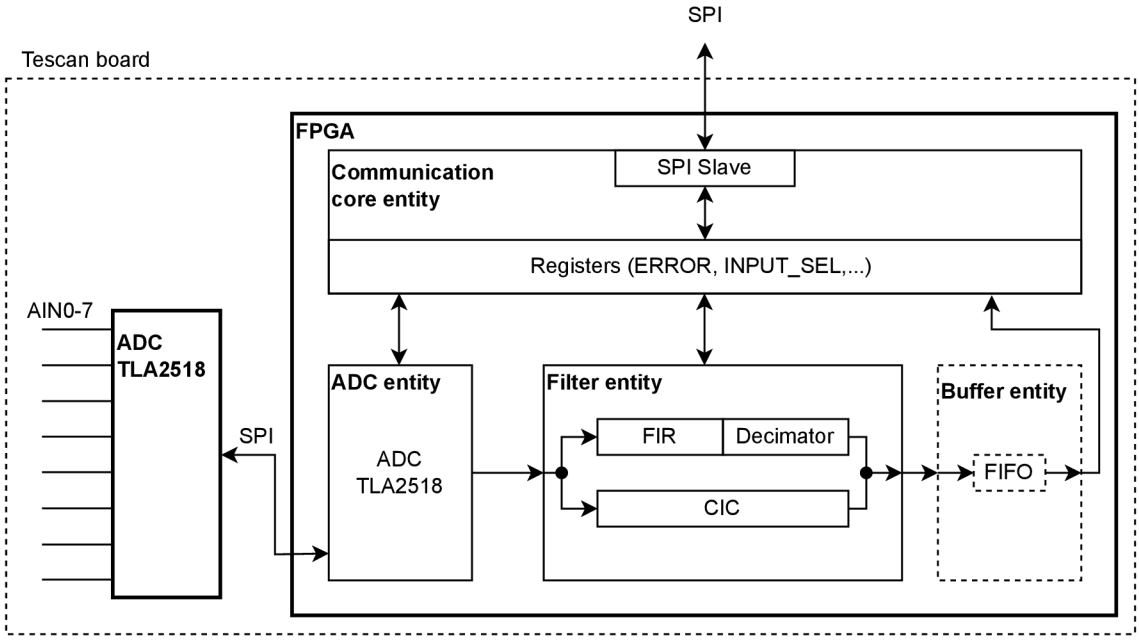
Ve struktuře testbench v jazyce VHDL je potřeba definovat generickou proměnnou runner_cfg a strukturu testu. Ta se typicky skládá z cyklu while ve kterém jsou vnořeny testy. Jednoduchý příklad testbench je následovný: [19]

```
1  library vunit_lib;
2  context vunit_lib.vunit_context;
3
4  entity vunit_testbench is
5    generic (runner_cfg : string);
6  end entity;
7
8  architecture tb of vunit_testbench is
9  begin
10   main : process
11   begin
12     test_runner_setup(runner, runner_cfg);
13     while test_suite loop
14       if run("test_001") then
15         -- test 1
16       elsif run("test_002") then
17         -- test 2
18       end if;
19     end loop;
20     test_runner_cleanup(runner);
21   end process;
22 end architecture;
```

Výpis 2: Příklad testbench ve VHDL (framework VUnit)

6 Návrh modulu pro předzpracování dat ve VHDL

V této kapitole je popsán praktický návrh modulu pro předzpracování dat v jazyce `VHDL`. Na obr. 6.1 je ideově znázorněno blokové schéma modulu pro předzpracování dat. Jednotlivé části blokového schéma jsou v kapitole dále detailněji popsány. Všechny vysvětlované zdrojové kódy jsou k dispozici na elektronické příloze (struktura souborů je uvedena v příloze A).

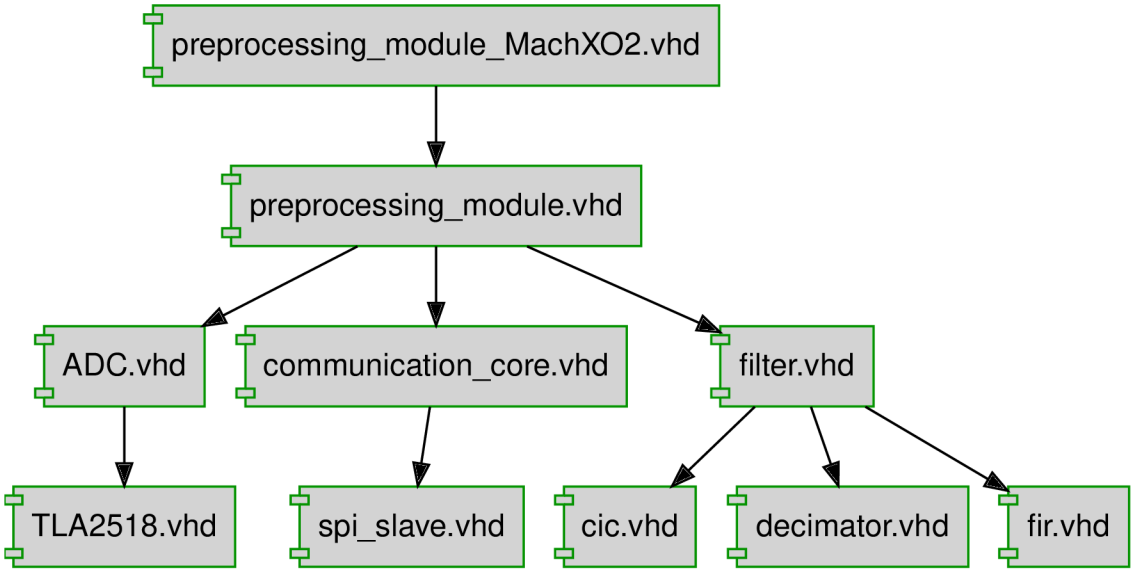


Obr. 6.1: Blokové schéma HDL modulu pro předzpracování

HDL modul se pro nadřazený systém chová jako zařízení typu SPI Slave. Pro A/D převodník se chová jako zařízení typu SPI Master. Celý modul obsahuje 8 a 16bitové registry, které lze pomocí komunikačního rozhraní nastavovat nebo vyčítat. Modul je důležité na začátku promyšleně rozdělit do jednotlivých entit, tak aby bylo docíleno systematického kódu a v budoucnu efektivnější spravovatelnosti a modifikovatelnosti kódu. Každá entita plní určitou činnost a je nezávislá na ostatních entitách.

Entita `preprocessing_module`, která slouží jako Top Level Entity (platformě nezávislá), propojuje všechny entity mezi sebou. Ve svém Generic parametru obsahuje informace o verzi modulu, informaci zda je verze testovací (Release Candidate), svojí Hardwarovou ID. Dalšími parametry jsou frekvence hodinového signálu (ten je generovaný přímo v FPGA), požadovaná frekvence komunikace po SPI s A/D převodníkem, šířka vstupního signálu (závisí na rozlišení a volbě konkrétního A/D

převodníku), šířka výstupního signálu (po filtraci a decimaci) a požadovaná vzorkovací frekvence vstupního a výstupního signálu. Graf hierarchické závislosti entit je uveden na obr. 6.2.



Obr. 6.2: Blokové schéma závislosti `VHDL` entit

Z obr. 6.2 je patrné, že modul obsahuje entitu `preprocessing_module_MachXO2`. Tato entita je platformě závislá, což znamená, že je syntetizovatelná pouze pro určitá FPGA od společnosti Lattice. Nachází se v ní instance vnitřního hodinového signálu, což je specifická funkce pro konkrétní FPGA. Dále pak entita při spuštění konfigurace generuje asynchronní reset, tak aby byly stavy inicializovány na správné počáteční hodnoty.

6.1 Rozhraní pro A/D převodník

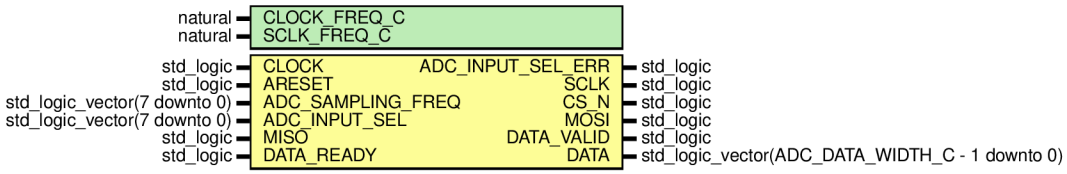
Rozhraní pro A/D převodník slouží pro komunikaci s A/D převodníkem a pro zpracování dat z A/D převodníku. Koncepce entity je volena genericky tak, aby bylo možné v budoucnu použít jiný A/D převodník.

6.1.1 Základní principy fungování entity `ADC`

Entita `ADC` obsahuje `if-generate`, který na základě generického parametru zvolí v syntéze daný typ A/D převodníku. V budoucnosti se předpokládá použití různých typů A/D převodníků. Tímto lze docílit přenositelnosti a modulárnosti kódu. Entita slouží pouze jako propojovací a neobsahuje žádnou číslicovou logiku.

6.1.2 Základní principy fungování entity `TLA2518`

Vstupně výstupní rozhraní entity `TLA2518` je popsané na obr. 6.3. Entita obsahuje generické vstupy pro frekvenci hodinového signálu a frekvenci hodinového signálu SPI.



Obr. 6.3: Diagram vstupů a výstupů entity `TLA2518`

Popis funkce `TLA2518`

Entita s vzorkovací frekvencí nastavenou v registru `ADC_SAMPLING_FREQ` neustále vyčítá data z A/D převodníku. Zároveň musí být zohledněn požadavek na změnu vstupního kanálu, přičemž při změně obsahu registru `ADC_INPUT_SEL` entita nakonfiguruje pomocí SPI A/D převodník na požadovaný kanál. Musí být respektován čas potřebný pro dokončení konverze signálu A/D převodníkem definovaným v katalogovém listu [4] jako `t_conv`. Výstupní data jsou z entity předávána pomocí korespondenčního protokolu AXI-Style ready/valid handshake realizovaného pomocí signálů `DATA`, `DATA_VALID` a `DATA_READY`.

Popis logiky `TLA2518`

Logika zapisování a čtení A/D převodníku je popsána v Moorově stavovém automatu, který se v FPGA vhodně implementuje. Obsahuje celkem 4 stavy (`IDLE_S, SPI_TRANSACTION_S, ADC_DATA_VALID_S, CONV_WAIT_S`).

Ve výchozím stavu `IDLE_S` se volí transakce, která proběhne. Transakcí mohou být celkem tři druhy - změna kanálu, změna vzorkovací frekvence, nebo vyčtení dat. Změna kanálu nebo vzorkovací frekvence proběhne prioritně, pokud se změní obsah v jejich registru. V entitě se nachází volně běžící čítač, který vyšle pulz, právě pokud jeho hodnota dosáhla nastavené vzorkovací frekvence. Na základě pulzu pak začíná transakce vyčítání dat z A/D převodníku. V tomto stavu se podle typu transakce nastaví vnitřní registry obsahující data na odeslání a délka komunikace. Pokud není požadavek na transakci, tak se signál `CS_N` se nachází ve stavu log. 1, jinak v log. 0.

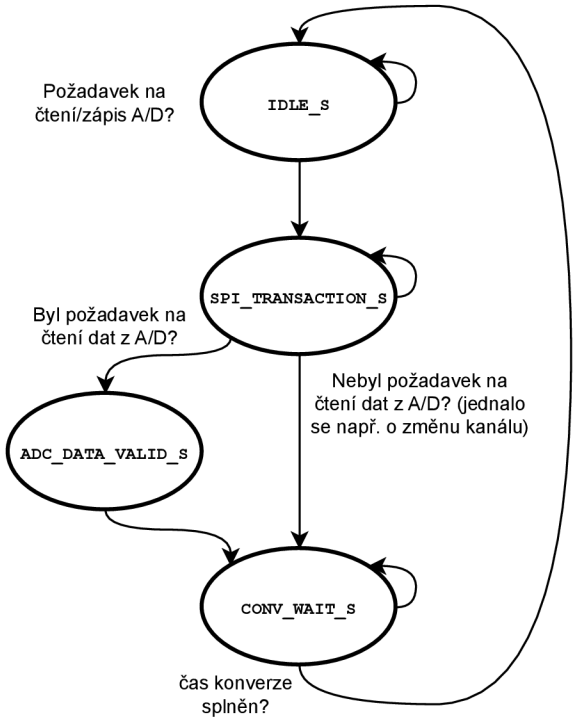
Ve stavu `SPI_TRANSACTION_S` probíhá SPI komunikace. SPI komunikace probíhá v režimu 0. Při náběžné hraně signálu `SCLK` probíhá vzorkování dat ze signálu `MISO`

do interního posuvného registru, pří sestupné hraně se pak z jiného interního posuvného registru nastavují data na signál `MOSI`. Pokud komunikace proběhla, tak se přechází do buď do stavu `ADC_DATA_VALID_S` pokud probíhalo vyčítání dat, jinak do stavu `CONV_WAIT_S`.

Ve stavu `ADC_DATA_VALID_S` se předávají data pomocí AXI-style ready/valid handshake. Tím se zaručí předání dat ve správný čas jiné entitě. Poté se přechází do stavu `CONV_WAIT_S`.

Ve stavu `CONV_WAIT_S` se zabezpečuje splnění podmínky `t_conv = 600 ns`, což je maximální čas potřebný pro převod analogové hodnoty na digitální převodníku TLA2518. To je realizováno čítačem aktivním při `CS_N = '1'`. Pokud čítač dočítá do maxima, objeví se příznak čítače `done` v log. 1 a přejde se do výchozího stavu `IDLE_S`.

Pro názornost je uveden nákres stavového automatu na obr. 6.4.



Obr. 6.4: Stavový automat v entitě `TLA2518`

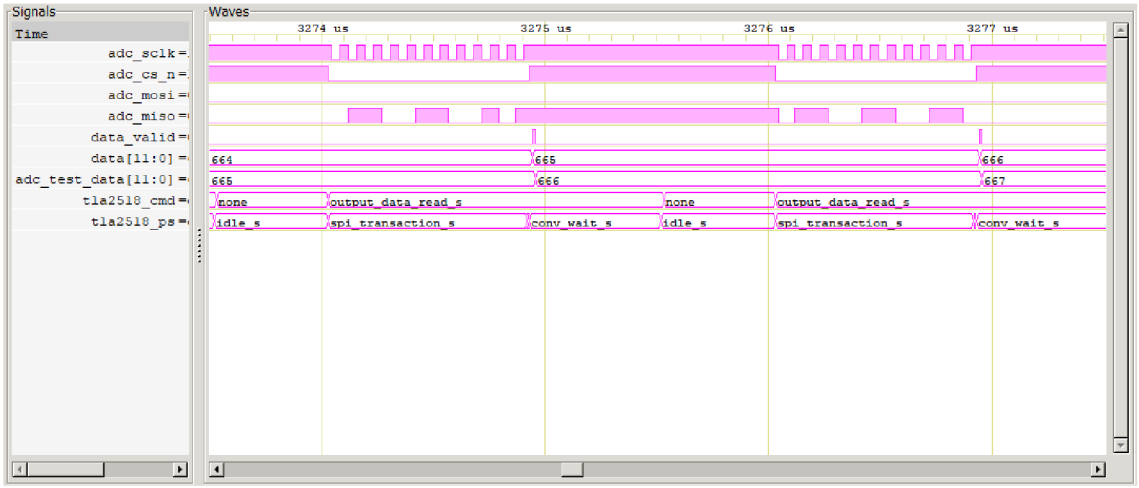
Analýza implementace `TLA2518`

Pro implementaci je potřeba 154 LUT, 126 registrů. Maximální frekvence hodinového signálu je stanovena na 118,4 MHz.

Test implementace `TLA2518`

Pro entitu `TLA2518` byly napsány celkem tři VUnit testy, přičemž každý se soustředí na jinou požadovanou funkcionalitu entity (`TLA2518 Continues data reading`, `TLA2518 Channel Change`, `TLA2518 Sample rate Change`).

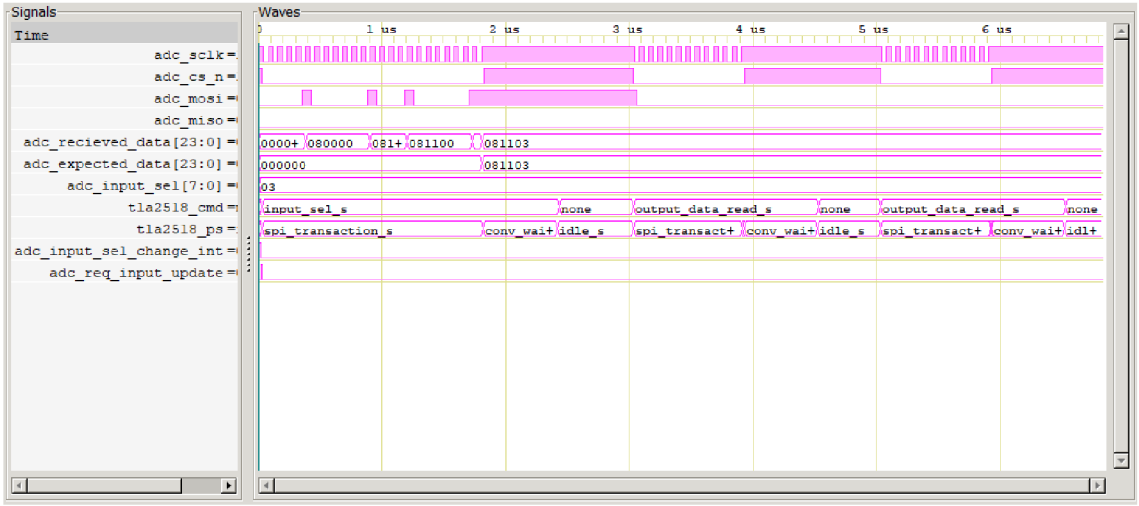
Testem `TLA2518 Continues data reading` se testuje příjem dat z A/D převodníku. Přijímaná data jsou 12bitová a jsou generována simulovaným A/D převodníkem TLA2518 v testbench. Na začátku testu se provádí reset zařízení, a pak v cyklu for jsou postupně vyčítána data ze simulovaného A/D převodníku. Na konci cyklu for se čeká na `DATA_VALID` v log. 1 a následně jsou přijatá data kontrolována s poslanými daty simulovaným A/D převodníkem. Tento cyklus testuje všechny možné hodnoty z A/D převodníku, tedy 2^{12} hodnot. Pokud by jakákoliv z těchto hodnot nebyla rovna očekávané hodnotě, tak test skončí s chybou. Náhled na testovací signály je na obr. 6.5, kde je patrná SPI komunikace, očekávaná data (`adc_test_data`) a skutečně přijatá data `data`. Jsou patrné i stavy Moorova stavového automatu uvnitř entity. Vzorky jsou v tomto konkrétním případě vzorkovány s periodou 500 kHz, což je z obr. 6.5 patrné. Je tedy i ověřeno to, že komunikace neprobíhá sporadicky, ale pouze v přesně definovaných vzorkovacích intervalech.



Obr. 6.5: Náhled na VUnit test `TLA2518 Continues data reading`

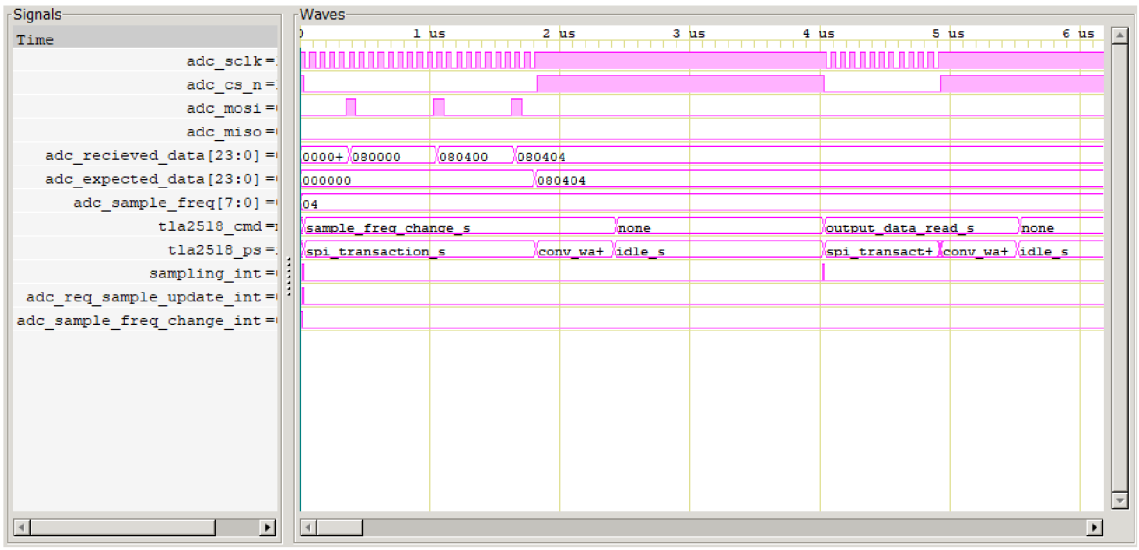
Testem `TLA2518 Channel Change` se testuje změna kanálu A/D převodníku. Na začátku testu se provádí reset zařízení, a pak je posílán datový rámec pro změnu kanálu A/D převodníku TLA2518, tak jak je definován v datovém listu [4]. Na konci testu se kontroluje, jestli je odeslaný datový rámec roven očekávanému datovému rámci. Náhled na testovací signály je na obr. 6.6, kde je patrná 24bitová SPI komunikace, oproti standardní 12bitové v případě vyčítání dat. Jsou patrná očekávaná data

(`adc_test_data`) a skutečně přijatá data `data`. Jsou patrné i stavy Moorova stavového automatu uvnitř entity. Po změně kanálu se přechází do stavu standardního vyčítání A/D převodníku v přesně definovaných vzorkovacích intervalech.



Obr. 6.6: Náhled na VUnit test `TLA2518 Channel Change`

Testem `TLA2518 Sample rate Change` se testuje změna vzorkovací frekvence A/D převodníku. Test je analogický jako test `TLA2518 Channel Change`, protože se jedná o stejně dlouhý 24bitový zápis, jen do jiného registru. Na začátku testu se provádí reset zařízení, a pak je posílán datový rámec pro změnu vzorkovací frekvence A/D převodníku TLA2518, tak jak je definován v datovém listu [4]. Na konci testu se kontroluje, jestli je odeslaný datový rámec roven očekávanému datovému rámci. Náhled na testovací signály je na obr. 6.7, kde je patrná 24bitová SPI komunikace, oproti standardní 12bitové v případě vyčítání dat. Jsou patrná očekávaná data (`adc_test_data`) a skutečně přijatá data `data`. Jsou patrné i stavy Moorova stavového automatu uvnitř entity. Po změně vzorkovací frekvence se přechází do stavu standardního vyčítání A/D převodníku v přesně definovaných nových vzorkovacích intervalech. Hodnota vzorkování totiž narozdíl od volby vstupního kanálu neovlivňuje jen A/D převodník, ale i interval SPI komunikace generované z FPGA.



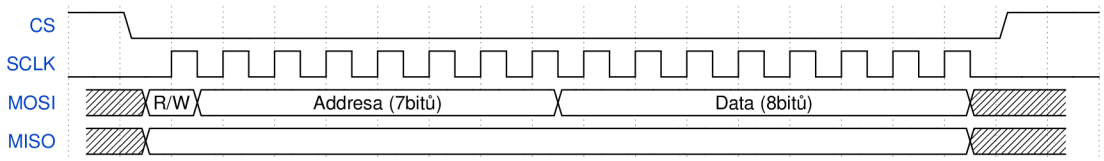
Obr. 6.7: Náhled na VUnit test `TLA2518 Sample rate Change`

6.2 Komunikační rozhraní

Entita komunikačního rozhraní obsahuje definici a implementaci všech konfiguračních a stavových registrů (`communication_core`). Obsahuje rozhraní pro komunikaci s vnějším zařízením (`spi_slave`). Entita se pro nadřazený systém chová jako zařízení typu SPI Slave a komunikuje s ním podobně jako paměťový prostor.

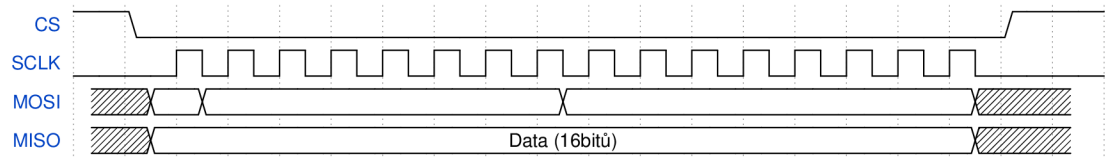
Parametry SPI komunikace

Pro komunikaci s modulem bude využívána sběrnice SPI v režimu 0. Pro korektní komunikaci je nutné stanovit parametry komunikace. SPI Slave očekává 16bitový datové datové rámce. První MSB bit komunikace určuje typ příkazu, zda se jedná o čtení, nebo zápis. Druhý až sedmý bit slouží pro nastavení adresy registru, na kterém je operace prováděna. Pokud se jedná o zápis, tak se po korektním ukončení komunikace zapíše hodnota do příslušného registru, pro názornost je uveden obr. 6.8.



Obr. 6.8: Datové rámce pro komunikaci s HDL modulem

Pokud se jedná o čtení, tak se obsah registru vypíše při probíhající následující komunikaci po SPI, pro názornost je uveden obr. 6.9.



Obr. 6.9: Datové rámce pro komunikaci s HDL modulem znázorňující cyklus čtení

Přístupné konfigurační a stavové registry

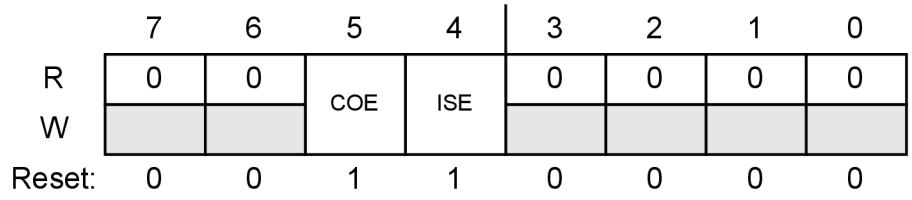
V této sekci jsou detailně popsány a vysvětleny konfigurační a stavové registry. Souhrnný popis konfiguračních registrů modulu pro předzpracování je uveden v tab. č. 6.1

Tab. 6.1: Konfigurační a stavové registry HDL modulu pro předzpracování

Adresa	Název registru	Popis registru
0x07	`ERROR`	Stav. reg. chybových příznaků
0x0A	`INPUT_SEL`	Konf. reg. pro volbu kanálu A/D převodníku
0x0B	`DECIMATION_RATE`	Konf. reg. pro určení decimačního poměru
0x0C	`ADC_DATA`	Stav. reg. pro vyčítání dat z A/D převodníku
0x0D	`OUTPUT_DATA`	Stav. reg. pro vyčítání vyfiltrovaných dat
0x0E	`SAMPLING_FREQUENCY`	Konf. reg. pro nastavení vzorkovací frekvence

Stavový registr `ERROR`

Stavový registr `ERROR` slouží pro indikaci chyb. Umístění bitů v registru je uvedeno na obr. 6.10. Popis jednotlivých bitů v registru je uveden v tab. 6.2.



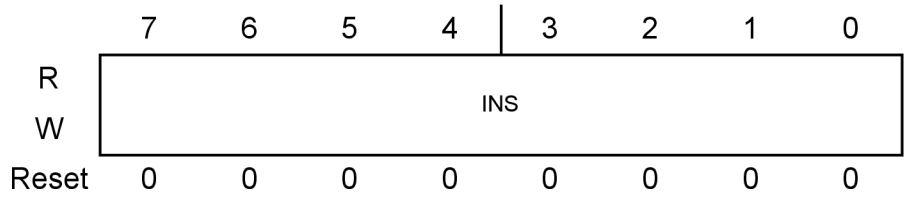
Obr. 6.10: Stavový registr `ERROR`

Bit	Popis
5 COE	**Communication Error** - bit indikuje chybu komunikace nadřazeného systému s modulem. 0 komunikace proběhla v pořádku 1 komunikace proběhla s chybou
4 ISE	**Input Select Error** - bit indikuje chybu volby vstupního kanálu, byl zvolen kanál mimo podporovaný rozsah A/D převodníkem 0 volba vstupního kanálu proběhla v pořádku 1 volba vstupního kanálu proběhla s chybou

Tab. 6.2: Popis stavového registru `ERROR`

Konfigurační registr `INPUT_SEL`

Konfigurační registr `INPUT_SEL` slouží pro volbu vstupního kanálu do A/D převodníku. Na základě informace v registru FPGA nakonfiguruje A/D převodník na daný kanál, pakliže není pro daný A/D převodník mimo rozsah. Umístění bitů v registru je uvedeno na obr. 6.11. Popis jednotlivých bitů v registru je uveden v tab. 6.3.



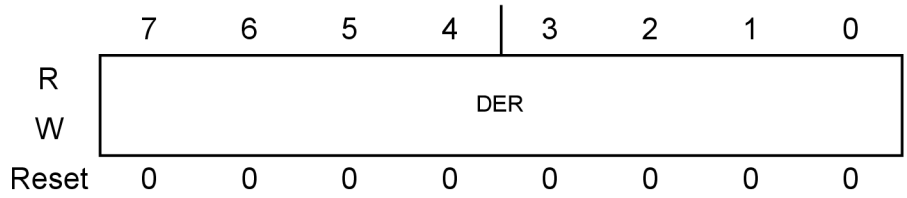
Obr. 6.11: Konfigurační registr `INPUT_SEL`

Bit	Popis
7:0 INS	**Input select** - Volba vstupního kanálu A/D převodníku 0-255 odpovídá číslu vstupního kanálu

Tab. 6.3: Popis konfiguračního registru `INPUT_SEL`

Konfigurační registr `DECIMATION_RATE`

Konfigurační registr `DECIMATION_RATE` slouží pro volbu decimačního poměru. Na základě informace v registru FPGA nakonfiguruje decimaci na danou hodnotu. Umístění bitů v registru je uvedeno na obr. 6.12. Popis jednotlivých bitů v registru je uveden v tab. 6.4.



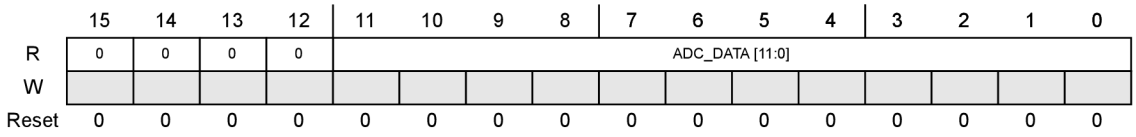
Obr. 6.12: Konfigurační registr `DECIMATION_RATE`

Bit	Popis
7:0 DER	**Decimation rate** - Volba decimačního poměru 0-255 odpovídá decimačnímu poměru 0 až 255

Tab. 6.4: Popis konfiguračního registru `DECIMATION_RATE`

Stavový registr `ADC_DATA`

Stavový registr `ADC_DATA` slouží pro vyčítání přijatých dat z A/D převodníku před vstupem do entity filtrace. Celková velikost registru jsou 2 byty, data jsou z A/D převodníku jsou v posledních 12 bitech registru. Registr je určen pouze pro čtení a nelze do něj zapisovat. Umístění bitů v registru je uvedeno na obr. 6.13. Popis jednotlivých bitů v registru je uveden v tab. 6.5.



Obr. 6.13: Stavový registr `ADC_DATA`

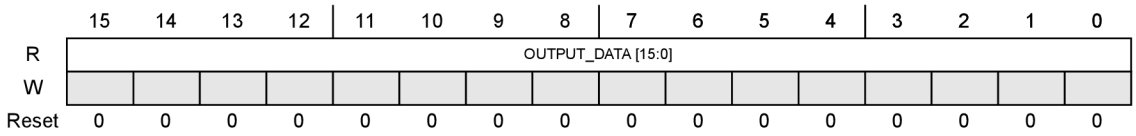
Bit	Popis
11:0 ADC_DATA	**ADC Data** - přijatá data z A/D převodníku 0-4095 je hodnota reprezentovaná ve dvojkovém doplňku

Tab. 6.5: Popis stavového registru `ADC_DATA`

Stavový registr `OUTPUT_DATA`

Stavový registr `OUTPUT_DATA` slouží pro vyčítání dat z entity filtrace. Celková velikost registru jsou 2 byty, velikost dat je 16 bitů. Registr je určen pouze pro čtení a nelze

do něj zapisovat. Umístění bitů v registru je uvedeno na obr. 6.14. Popis jednotlivých bitů v registru je uveden v tab. 6.6.



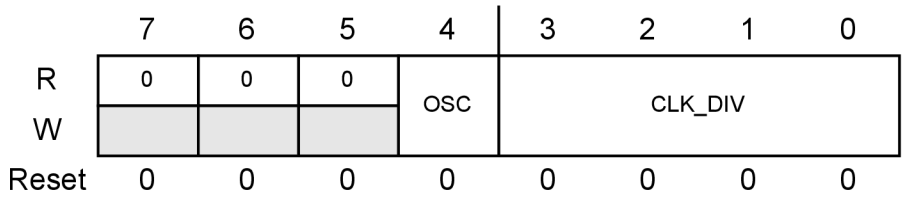
Obr. 6.14: Stavový registr `OUTPUT_DATA`

Bit	Popis
15:0 OUT_DATA	**OUTPUT Data** - data z entity filtrace 0-65535 je hodnota reprezentovaná ve dvojkovém doplňku

Tab. 6.6: Popis stavového registru `OUTPUT_DATA`

Konfigurační registr `SAMPLING_RATE`

Konfigurační registr `SAMPLING_RATE` slouží pro nastavení vzorkovací frekvence. Na základě informace v registru FPGA nakonfiguruje A/D převodník na danou vzorkovací frekvenci a s touto vzorkovací frekvencí i FPGA nastaví interval vyčítání hodnot z A/D převodníku. Vzorkovací frekvence vycházejí přímo z dokumentace ([4]) A/D převodníku TLA2518 a pro jiný typ by bylo nutné registr předefinovat na jiné frekvence pro daný typ A/D převodníku. Umístění bitů v registru je uvedeno na obr. 6.15. Popis jednotlivých bitů v registru je uveden v tab. 6.7. Přehledná tabulka hodnoty vzorkovací frekvence podle nastavení registru je uvedena na obr. 6.16.



Obr. 6.15: Konfigurační registr `SAMPLING_RATE`

Bit	Popis
3:0 CLK_DIV	**Clock divider** - Nastavení děličky hodinového signálu v A/D převodníku
4:4 OSC_SEL	**Oscillator select** - Nastavení oscilátoru v A/D převodníku

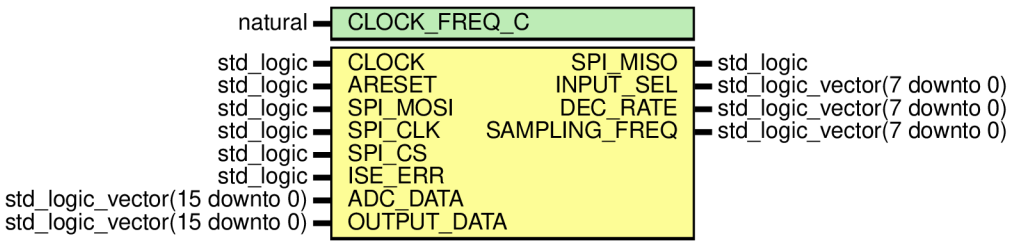
Tab. 6.7: Popis konfiguračního registru `SAMPLING_RATE`

CLK_DIV[3:0]	OSC_SEL = 0		OSC_SEL = 1	
	SAMPLING FREQUENCY, f_{CYCLE} (kSPS)	CYCLE TIME, t_{CYCLE} (µs)	SAMPLING FREQUENCY, f_{CYCLE} (kSPS)	CYCLE TIME, t_{CYCLE} (µs)
0000b	1000	1	31.25	32
0001b	666.7	1.5	20.83	48
0010b	500	2	15.63	64
0011b	333.3	3	10.42	96
0100b	250	4	7.81	128
0101b	166.7	6	5.21	192
0110b	125	8	3.91	256
0111b	83	12	2.60	384
1000b	62.5	16	1.95	512
1001b	41.7	24	1.3	768
1010b	31.3	32	0.98	1024
1011b	20.8	48	0.65	1536
1100b	15.6	64	0.49	2048
1101b	10.4	96	0.33	3072
1110b	7.8	128	0.24	4096
1111b	5.2	192	0.16	6144

Obr. 6.16: Konfigurace vzorkovací frekvence TLA2518 [4]

6.2.1 Základní principy fungování entity `communication_core`

Entita `communication_core` sdružuje v textu výše popsané registry a zajišťuje správné předávání informací z a do entity `spi_slave`. Vstupně výstupní rozhraní entity `decimator` je popsané na obr. 6.17.



Obr. 6.17: Diagram vstupů a výstupů entity `communication_core`

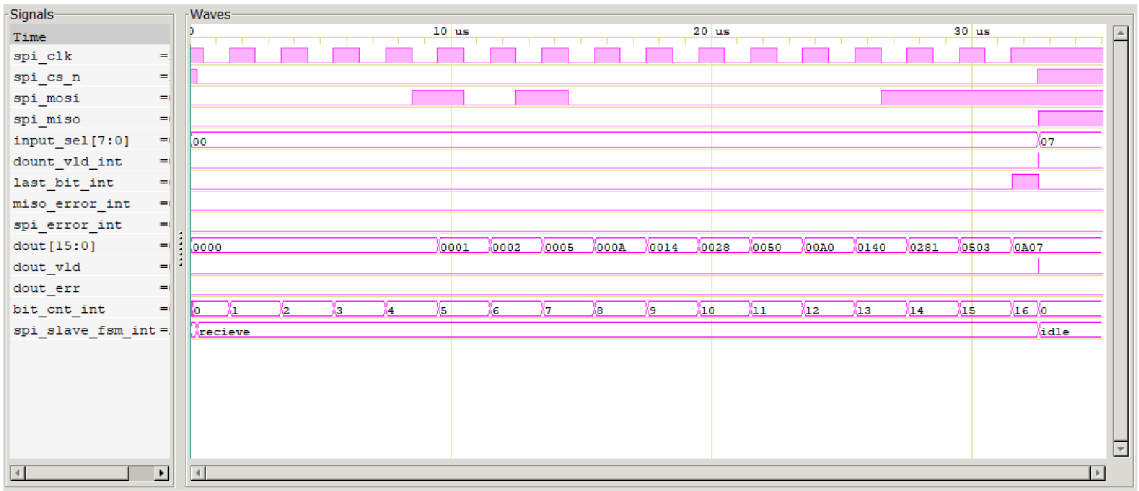
Entita očekává validní příjem dat z `spi_slave`, pokud validní data obdrží, tak na základě znalosti rozmístění bitů v datovém rámci (řídicí bit, adresové bity a datové bity) správně přiřadí (parsuje) data do konkrétního registru. V případě čtení registru, nastaví multiplexor na konkrétní registr, ze kterého se čte a tato data propojí na vstup SPI Slave, tak aby byla při další transakci připravena pro odesílání.

Analýza implementace `communication_core`

Pro implementaci je potřeba 110 LUT, 134 registrů. Maximální frekvence hodinového signálu je stanovena na 141,1 MHz.

Test implementace `communication_core`

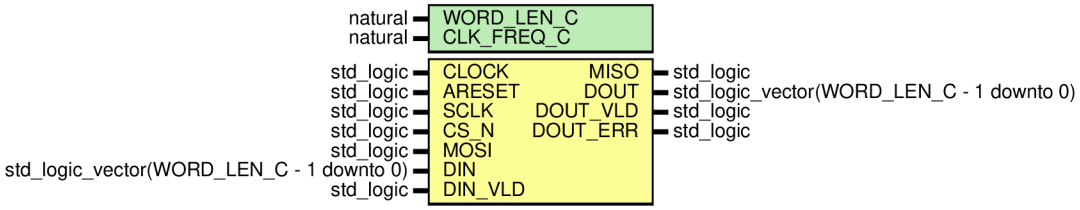
Pro entitu byl vytvořen testbench `communication_core_tb` s využitím framework VUnit. Testuje se zápis do registru `INPUT_SEL`, který volí vstupní kanál A/D převodníku TLA2518. Procedura testu je prostá, nejdříve se zařízení resetuje, a pak SPI Master (který je napsán jako procedura v testbench) vysílá datový rámec, požadující nastavení vstupního kanálu A/D převodníku na 7. Jakmile komunikace proběhne, tak se testuje, jestli je hodnota v registru totožná s očekávanou hodnotou. V případě, že by zápis do registru správně nebyl realizován, tak se do konzole vypíše očekávaná hodnota a skutečná hodnota. Na obr. 6.18 je patrná SPI komunikace a na konci komunikace je patrná změna obsahu registru `input_sel`, tak jak bylo očekáváno. Pokud by ke změně nedošlo, tak by simulace skončila neúspěšně. Implementace ostatních registrů je provedena obdobně, proto je komplexnější testování čtení a zápisu do registrů později prováděno až na reálném zařízení.



Obr. 6.18: Náhled na VUnit test `Write to INPUT_SEL`

6.2.2 Základní principy fungování entity `spi_slave`

Entita `spi_slave` zabezpečuje korektní přenos dat z nadřazeného systému do FPGA. Vstupně výstupní rozhraní entity `spi_slave` je popsané na obr. 6.19.



Obr. 6.19: Diagram vstupů a výstupů entity `spi_slave`

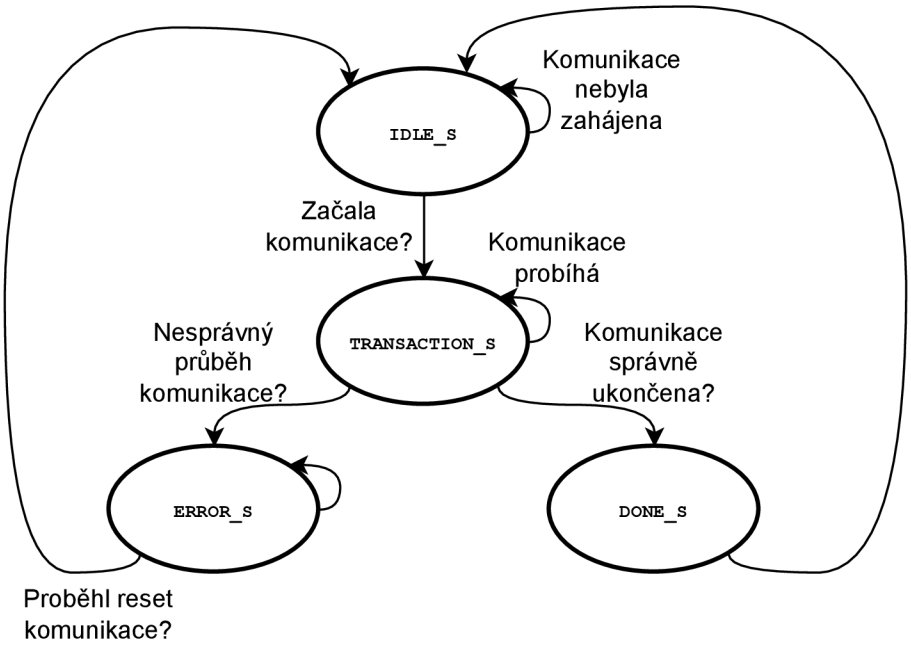
Protože se jedná o přechod mezi hodinovými doménami, kde je možná metastabilita vstupních signálů, tak je potřeba zvážit metodu návrhu pro řešení tohoto fenoménu. Možnosti přechodu mezi hodinovými doménami byly popsány v textu dříve. Pro tuto implementaci je pro signály zvoleno řešení pomocí dvou klopných obvodů typu D pro signály `SCLK`, `CS_N` a `MOSI`. Tím je docíleno velké pravděpodobnosti, že signály nebudou v metastabilitě. Entita obsahuje nezávislý čítač náběžných hran signálu `SCLK`, tak aby bylo možné detekovat chybu, kdy přijde jiný datový rámec, než který je očekávaný. V takovém případě zařízení bude chybovém stavu a bude čekat na novou komunikaci, či reset zařízení. Hlavní příjem a odesílání dat je implementováno v Moorově stavovém automatu znázorněném na obr. 6.20, obsahující celkem 4 stavy (`IDLE_S, TRANSACTION_S, ERROR_S a DONE_S`)

Ve stavu `IDLE_S` se čeká na sestupnou hranu signálu `CS_N`, kdy se přechází do stavu `TRANSACTION_S`.

Ve stavu `TRANSACTION_S` probíhá příjem dat. Při každé náběžné hraně signálu `SCLK` jsou přijímána data a vkládána do posuvného registru. Při každé sestupné hraně signálu `SCLK` jsou vzorkována data výstup `MISO` z posuvného registru. Pokud by ve stavu `TRANSACTION_S` došlo k indikaci chybového bitu, tak přejde do stavu `ERROR_S`. Pokud je komunikace korektně ukončena, tak jak je očekáváno, tak přejde do stavu `DONE_S`.

Ve stavu `ERROR_S` se čeká na reset zařízení, nebo na signál `CS_N` v log. 1.

Ve stavu `DONE_S` je vyslán puls o validních výstupních datech a zařízení přejde do svého výchozího stavu `IDLE_S` a je připraveno na další komunikaci.



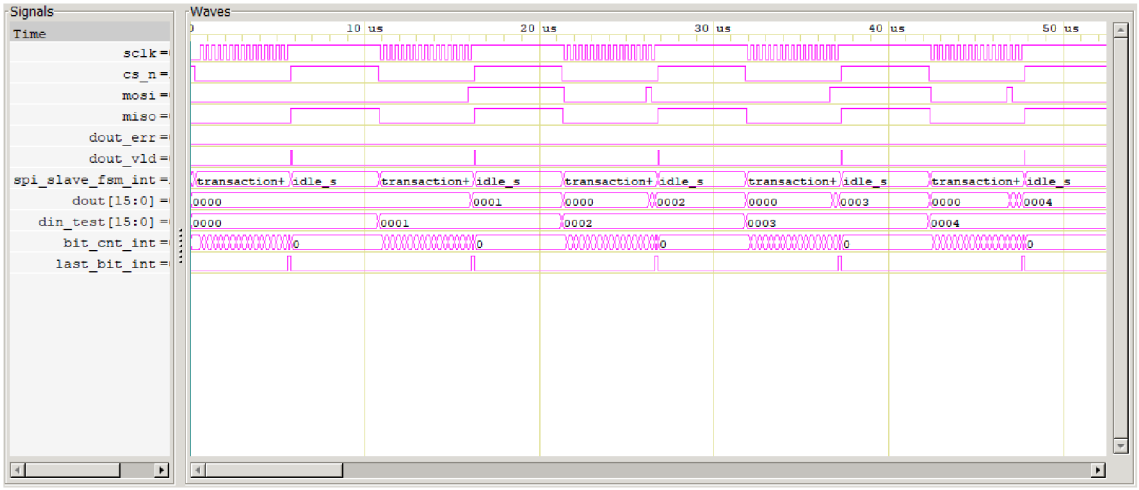
Obr. 6.20: Stavový automat v entitě `spi_slave`

Analýza implementace `spi_slave`

Pro implementaci je potřeba 49 LUT, 65 registrů. Maximální frekvence hodinového signálu je stanovena na 195,6 MHz.

Test implementace `spi_slave`

Pro entitu byl vytvořen testbench `spi_slave_tb` s využitím framework VUnit. V testu `SPI Slave Continues data read test` se porovnávají přijatá a odesílaná data. Procedura testu se skládá z resetu zařízení a poté přijímáním dat. SPI Master je v tomto případě napsán pomocí for-cyklu v testbench. Data jsou testována v celém rozsahu možných hodnot v datovém rámci, v tomto případě je testováno 2^{16} hodnot. Z obr. 6.21 jsou patrné některé důležité signály. Vektor `din_test` jsou testovací data odesílaná SPI Master a vektor `dout` jsou přijatá data pomocí SPI Slave. Lze si všimnout, že při validním pulzu `dout_vld` jsou si data rovna, tak jak bylo očekáváno. Pokud by rovna nebyla, tak se test ukončí chybovým výpisem.



Obr. 6.21: Náhled na VUnit test `SPI Slave Continues data read test`

6.3 Filtrace dat a zpracování dat

Entita Filtrace dat provádí samotnou filtraci signálu. Vstup do bloku je realizován pomocí AXI-style ready/valid handshake. Modul obsahuje implementované filtry popsané v kapitole pro předzpracování dat.

6.3.1 Základní principy fungování entity `filter`

Entita `filter` obsahuje generické rozhraní, které volí konkrétní typ filtru. Ten může nyní být FIR nebo CIC, ale v budoucnu může dojít k rozšíření o další typ digitálního filtru.

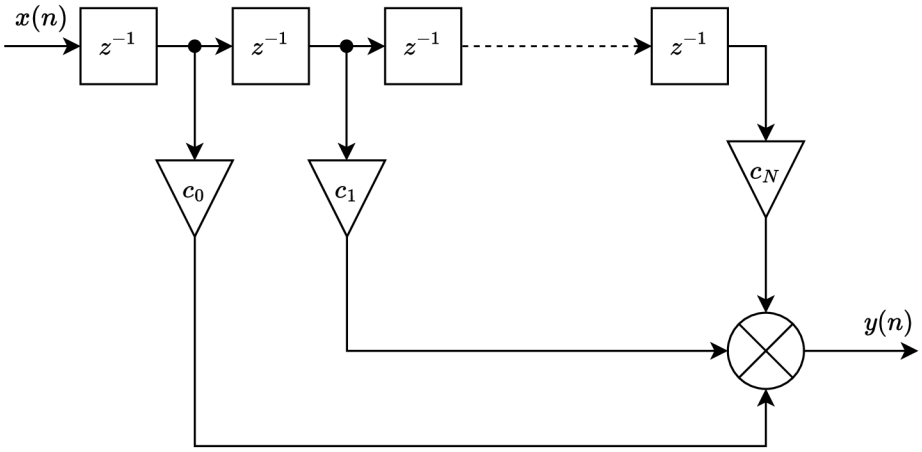
6.3.2 Základní principy fungování entity `fir`

FIR filtr v FPGA lze implementovat různými způsoby. Dvěma základními přístupy jsou paralelní a sériová implementace. Obvykle pokud je v FPGA k dispozici více násobiček, či přímo určených bloků pro výpočty (typicky v případě výrobce Xilinx a dedikovaných DSP bloků), tak můžeme volit paralelní implementaci, pokud jich však je jen omezené množství, nebo je třeba šetřit zdroje, tak se zvolí implementace sériová a složitost výpočtu se pak prodlužuje v čase a ne v nároku na zdroje FPGA.

Paralelní implementace

Obecné blokové schéma paralelní implementace FIR filtru řádu N je znázorněno na obr. 6.22. Blokové schéma se skládá ze zpožďovacích členů, které reprezentují různě

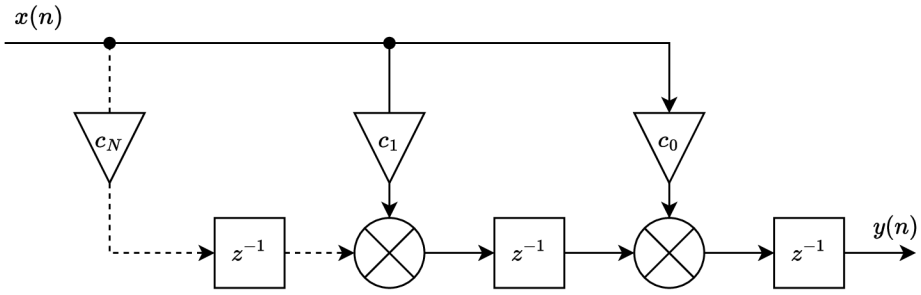
zpožděné vzorky signálů, z násobících členů, které násobí příslušný vzorek signálu koeficientem a nakonec ze sumátoru, který všechny výsledné signály sčítá.



Obr. 6.22: Blokové schéma paralelní implementace FIR filtru

Znázorněné blokové schéma na obr. 6.22 by však bylo pro FPGA použitelné pouze při nízkém řádu filtru, právě kvůli přítomnosti součtového členu. Při praktické implementaci by totiž mohlo docházet k problémům s časováním mezi vstupním a výstupním registrem, proto je vhodné použít zřetězení (pipelining). [10]

Upravené blokové schéma (tzv. Transposed FIR filter), které tento problém řeší je znázorněno na obr. 6.23.



Obr. 6.23: Blokové schéma zřetězené paralelní implementace FIR filtru

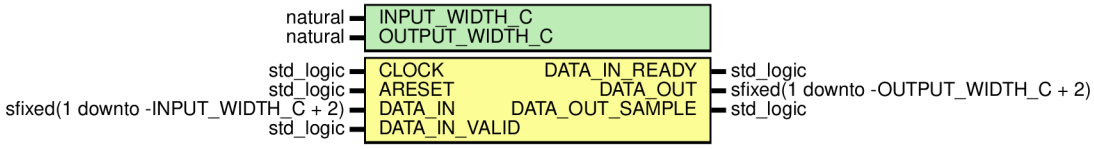
Sériová implementace

Sériová implementace je vhodná použít v případě, kdy je vnitřní hodinový signál v FPGA násobně rychlejší než vzorkovací frekvence dat. Vhodné je rozdělit konvoluci do několika kroků, kdy se každý krok bude cyklicky provádět (např. prostřednictvím stavového automatu). Taková implementace přináší velkou výhodu v ušetření zdrojů v FPGA tím, že se opakovaně využívají (časový multiplex). Prakticky to znamená

to, že namísto použití n (řád filtru) násobiček (v případě paralelní implementace) se použije pouze jediná, pro kterou se bude v každém taktu měnit vstup (např. pomocí multiplexeru). Další výhoda může spočívat v efektivním využívání RAM bloků pro uchovávání dat. [10]

Implementace FIR filtru v FPGA pro modul pro předzpracování dat

Jako nejvhodnější implementace FIR filtru v FPGA pro tuto práci se jeví sériová implementace, z důvodu menšího množství zdrojů ve vybraném FPGA. FIR filtr je napsán v jazyce `VHDL` genericky, tak aby jej bylo možné univerzálně používat. Genericky lze nastavit šířku vstupního a výstupního signálu z FIR filtru. Dalšími parametry FIR filtru jsou vnitřní koeficienty, které se nastavují v definici architektury. Lze nastavit jejich šířku, podle čehož se odvíjí využití zdrojů v FPGA a přesnost filtrace. Vstupy a výstupy do entity FIR filtru jsou znázorněny na obr. 6.24.



Obr. 6.24: Nákres entity realizující FIR filtr ve `VHDL`

Samotný proces výpočtu odezvy filtru s danými koeficienty je vykonáván ve stavovém automatu. Ten se skládá ze tří stavů `IDLE_S`, `ACTIVE_S` a `DONE_S`. Ve stavu `IDLE_S` se čeká na validní vstupní data. Poté se přechází do stavu `ACTIVE_S`, kde se pro efektivní výpočet náročný na nejméně možných zdrojů využívá časového multiplexování signálů. Ve stavu se dále nachází násobení a sumace. Tyto kroky jsou vhodně zřetězeny, tak aby bylo možné dosáhnout lepší časové analýzy. V případě implementování kódu do FPGA, které obsahuje dedikované DSP bloky by pravděpodobně došlo k jejich přímému využití, protože obvykle přímo podporují typ operace `Multiply-accumulate`. Tato skutečnost by rozhodně byla ku prospěchu k časové analýze obvodu. Počet časových multiplexů signálu je roven řádu filtru, což je nutno brát v úvahu a přizpůsobit vzorkovací kmitočet, popř. řád filtru, protože se od této skutečnosti odvíjí čas výpočtu výstupu FIR filtru. Na základě této znalosti lze určit, že počet hodinových cyklů potřebných pro výpočet výstupního vzorku je roven počtu řádu filtru + 3 cykly, což je počáteční zpoždění registrů typu D určených pro zřetězení (pipelining).

`VHDL` kód hlavního výpočtu výstupu filtru ve stavu `ACTIVE_S` je ukázána na výpisu č. 3.

```
-- main time multiplexed FIR filter process
when ACTIVE_S =>

    -- do multiply and accumulate stages for specified cycles
    if cnt = ACTIVE_STATE_D_C - 1 then
        cnt   <= 0;
        state <= DONE_S;
    else
        cnt <= cnt + 1;
    end if;

    -- data multiplex (resource sharing)
    if cnt < TAPS_C - 1 then
        data_in_d(TAPS_C - 1 - cnt) <= data_in_d(TAPS_C - 1 - cnt - 1);
        mult_inp_sig                <= data_in_d(TAPS_C - 1 - cnt);
        mult_inp_coeff              <= COEFF_C(TAPS_C - 1 - cnt);
    end if;

    -- multiply, use fixed rounding and saturation
    mult <= resize(mult_inp_sig * mult_inp_coeff,
                   mult,
                   round_style    => fixed_round,
                   overflow_style => fixed_saturate);

    -- accumulate, use fixed rounding and saturation
    acc <= resize(acc + mult,
                  acc,
                  round_style    => fixed_round,
                  overflow_style => fixed_saturate);
```

Výpis 3: Implementace hlavního výpočtu FIR filtru ve `VHDL`

Jakmile je stav (výpočet FIR filtru) dokončen, přechází se do stavu `DONE_S`, kdy se data překlopí do výstupního registru a modul vyšle puls, který indikuje nově vypočtená data.

Analýza implementace `fir`

Konkrétní velikost implementace FIR filtru je závislá na volbě a počtu koeficientů, takže nelze obecně říci počty zdrojů v FPGA. V kapitole, která se týká reálného testování zařízení je velikost implementace pro jednotlivé testované konfigurace FIR filtru uvedena.

Test implementace `fir`

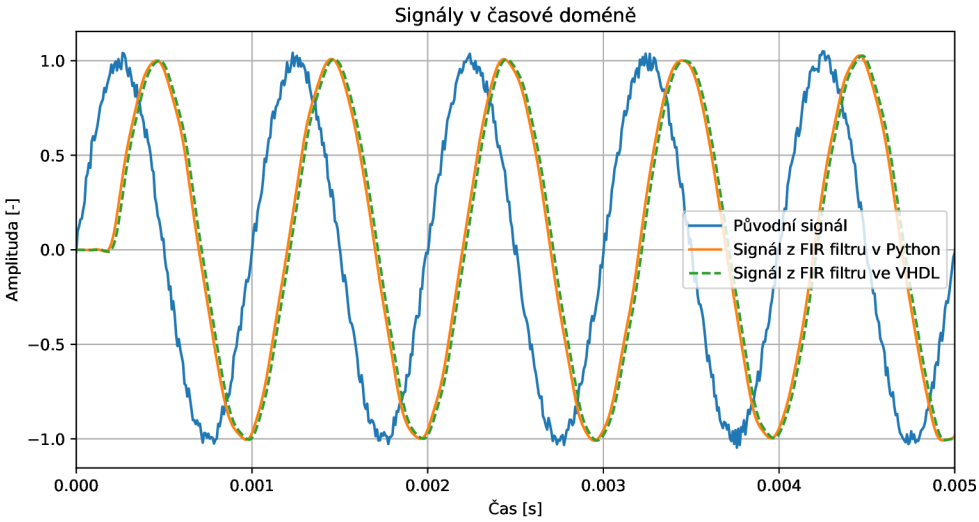
Pro testování FIR filtru v FPGA pro HDL modul byl vytvořen testbench. Pro zjištění, zda je FIR filtr korektně funkční bylo využito srovnání odezvy FIR filtru v FPGA a FIR filtru dostupného v knihovně pro programovací jazyk Python `SciPy`, což je sada volně dostupných nástrojů pro práci s daty, pro vědeckou a technickou práci. Pomocí volně dostupné knihovny `SciPy` byl navržen FIR filtr (pomocí metody váhových oken) typu dolní propust. Narozdíl od využití programového prostředí MATLAB má Python a jeho knihovny výhodu v tom, že je Open source přičemž obsahuje takřka všechny nástroje a knihovny, které nabízí i MATLAB. Jako parametry pro návrh filtru byly zvoleny:

- Vzorkovací frekvence: 125 kHz
- Mezní frekvence: 10 kHz
- Řád filtru: 51

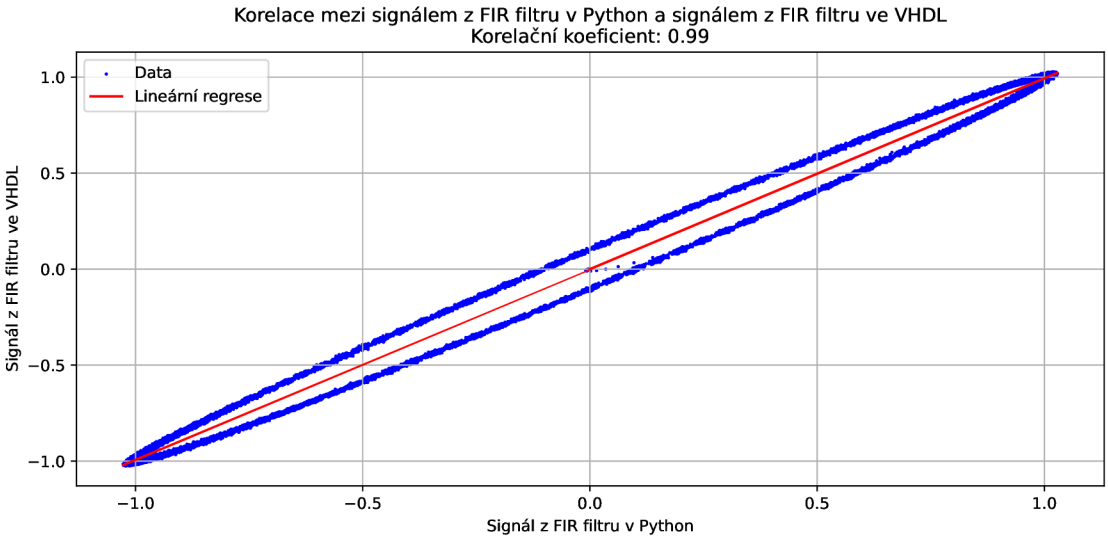
Na základě parametrů byly pomocí funkce `firwin` a Hammingova okna vypočteny koeficienty a spočítána odezva (konvenčně ta přesná pomocí Python) na testovací signál. Koeficienty z Python jsou typicky v datovém typu `float64`, nicméně pro FPGA se používá 16bitová šířka koeficientu ve `fixed_point`. Z toho je zřejmé, že přesnost výpočtu filtru nebude v FPGA tak přesná, jako v případě výpočtu v Python. Reprezentace čísla ve `VHDL` je ve dvojkovém doplňku. Pro přepočet koeficientů z `float64` na `fixed_point` byla napsána funkce v Python, která přímo vygeneruje kus kódu ve VHDL, obsahující hodnotu koeficientů.

Jako testovací signál byl zvolen signál, který se skládá z harmonické složky (na frekvenci 1 kHz) a bílého šumu. Signál se šumem, společně s filtrovanými signály v časové doméně jsou znázorněny v grafu č. 6.25.

Z časové domény je patrné, že skutečně došlo k filtraci šumu. Dále je zřejmé, že jsou signály z filtru v `Python` a v `VHDL` téměř identické, jsou jen o dva vzorky od sebe fázově posunuté, což je způsobeno zpožděním výpočtu ve `VHDL` oproti `Python`. Pro skutečné porovnání podobnosti signálu lze využít korelaci. Pro výpočet korelace mezi dvěma signály v `Python` lze použít funkci `numpy.corrcoef()` z knihovny `NumPy`. Tato funkce spočítá Pearsonovu korelační matici mezi dvěma signály. Pearsonova korelace měří lineární vztah mezi dvěma proměnnými. Lineární regrese získaných dat určuje korelaci mezi daty. Pokud se směrnice lineární regrese rovná číslu 1, pak lze říci, že mezi zkoumanými daty je přímá úměrnost, tedy že korelace mezi daty je 100 %. Pokud je rovna číslu 0 znamená to, že mezi daty není žádná závislost, pokud -1, tak mezi daty je nepřímá úměrnost. Vizualizace korelace je znázorněna na obr. 6.26. Korelační koeficient mezi signály z filtru v `Python` a v `VHDL` vyšel 0,99477, což defakto indikuje 99,5 % korelaci.



Obr. 6.25: Odezva FIR filtru na vstupní signál a jeho frekvenční charakteristika



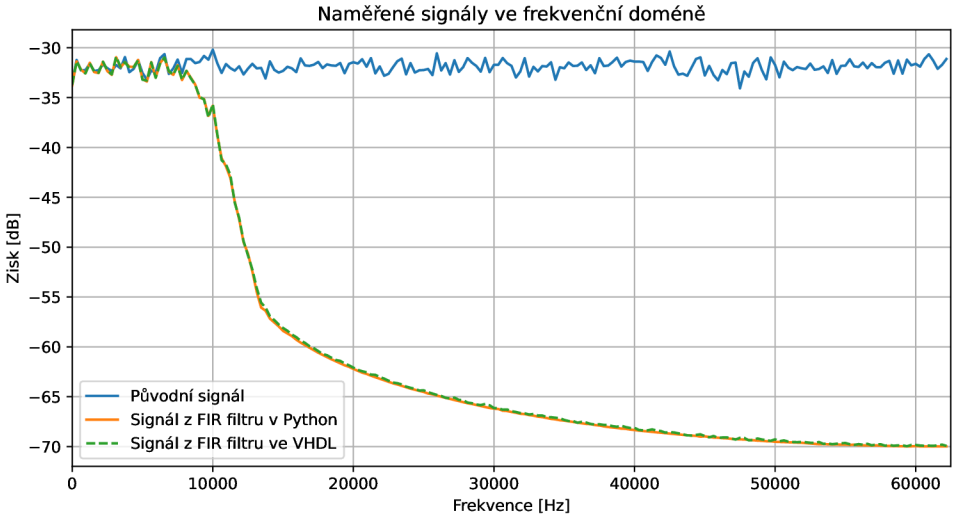
Obr. 6.26: Korelace mezi výstupy vypočítanými pomocí FIR filtru v `Python` a ve `VHDL`

Další způsob, jak porovnat chování filtrů je vykreslení frekvenční charakteristiky filtru. Ta byla změřena pomocí bílého šumu a následně vypočtena jako průměrná FFT na základě 50 měření, kdy každé měření obsahovalo 400 vzorků.

Z grafu č. 6.25 je patrné, že odezva FIR filtru ze SciPy je téměř totožná s filtrem napsaným pro FPGA. Nepřesnosti, které jsou mezi signály patrné, jsou způ-

sobeny zaokrouhlováním hodnot koeficientů a signálu, které vstupují do FPGA. Z důvodu optimalizace je šířka koeficientů implementovaných v FPGA 16 bitů, přičemž knihovna `SciPy` v Python používá datový typ `float64`. Z grafu je patrné, že skutečně dochází k vyfiltrování nízko frekvenční složky signálu podle očekávání.

Na základě simulace a následné zpracování dat, zde popsané, lze očekávat, že se FIR filtr bude chovat korektně i v reálné implementaci.



Obr. 6.27: Frekvenční charakteristika signálů vypočítanými pomocí FIR filtru v `Python` a ve `VHDL`

6.3.3 Základní principy fungování entity `cic`

Implementace CIC filtru v FPGA je podstatně méně náročná na zdroje v FPGA, než FIR filtr. CIC filtr defakto provádí filtrace pomocí plovoucího průměrování.

Pro jeho funkci není potřeba násobičky, ale jen sčítaček, podle kroků CIC filtru. Jednou z vhodnou implementací může být rozdělit samotný filtr do více kroků. V prvním kroku se provádí integrační část (využije se sčítačky pro akumulování hodnoty). Pokud je více kroků, tak je vhodné cesty mezi sčítačkami zřetězit, čímž sice výpočet bude trvat více hodinových cyklů, ale časová analýza výrazně zlepší. `VHDL` kód integrační části CIC filtru je na výpisu č. 4.

```vhdl
-- integrator stage needs to be done for STAGES_C times
when INTEGRATOR_S =>
if cnt = STAGES_C - 1 + 1 then
    cnt     <= 0;
    state   <= COMB_S;
    integ_d <= integ;
else
    cnt <= cnt + 1;
    -- cascade accumulation, first element is connected to input
    for i in 0 to STAGES_C - 1 loop
        if i = 0 then
            integ(i) <= resize(integ_inp + integ_d(i), integ(i));
        else
            integ(i) <= resize(integ(i - 1) + integ_d(i), integ(i));
        end if;
    end loop;
end if;
```

Výpis 4: VHDL kód integrační části CIC filtru

V druhém kroku se nachází derivační část (využije se sčítačky (resp. odčítačky)). Pokud je více kroků jsou opět cesty mezi odčítačkami zřetězeny. Dále se v tomto kroku nachází genericky nastavitelné zpoždění signálu do odčítačky, což určuje chování filtru a jeho frekvenční odezvu. VHDL kód derivační části CIC filtru je na výpisu č. 5.

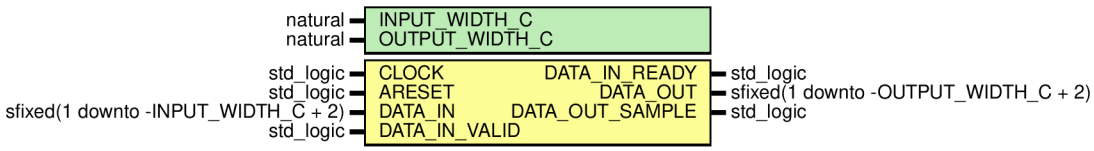
V jazyce VHDL je využito cyklu for-loop pro generování zřetězených sčítaček a odčítaček. Je nutno brát v úvahu výslednou velikost signálu po sčítání, zvolit ji dostatečně velkou tak, aby nedocházelo k přetečení, nebo podtečení. Je nutné brát v potaz bitový růst (označený jako BG_{CIC}), který lze vypočítat pomocí následujícího vztahu, kde D určuje počet zpožďujících členů derivační části a N je počet kroků CIC filtru:

$$BG_{CIC} = \log_2(D^N) \tag{6.1}$$

```vhdl
-- comb stage needs to be done for STAGES_C times
when COMB_S =>
if cnt = STAGES_C - 1 + 1 then
    cnt   <= 0;
    state <= DONE_S;
    -- generate delay line
    for i in 0 to STAGES_C - 1 loop
        if i = 0 then
            comb_d(i) <= comb_d(i)(comb_d(i)'high - 1 downto comb_d(i)'low) &
                         integ_d(STAGES_C - 1);
        else
            comb_d(i) <= comb_d(i)(comb_d(i)'high - 1 downto comb_d(i)'low) &
                         comb(i - 1);
        end if;
    end loop;
else
    cnt <= cnt + 1;
    -- cascade subtraction, first element is connected to input from last
    -- integrator stage
    for i in 0 to STAGES_C - 1 loop
        if i = 0 then
            comb(i) <= resize(integ(STAGES_C - 1) - comb_d(i)(comb_d(i)'high),
                              comb(i));
        else
            comb(i) <= resize(comb(i - 1) - comb_d(i)(comb_d(i)'high), comb(i));
        end if;
    end loop;
end if;
```

Výpis 5: VHDL kód derivační části CIC filtru

Je nutné, aby počet zpožďujících členů derivační části D byla mocnina čísla 2. Poté lze efektivně využít dělení pomocí bitového posunu. Vstupy a výstupy do entity CIC filtru jsou znázorněny na obr. 6.28.



Obr. 6.28: Nákres entity realizující CIC filtr ve VHDL

Analýza implementace `cic`

Konkrétní velikost implementace CIC filtru je závislá na volbě a počtu kroků (stage) a dalších parametrech CIC filtru, takže nelze obecně říci počty zdrojů v FPGA. V kapitole, která se týká reálného testování zařízení je velikost implementace testovaného CIC filtr uvedena.

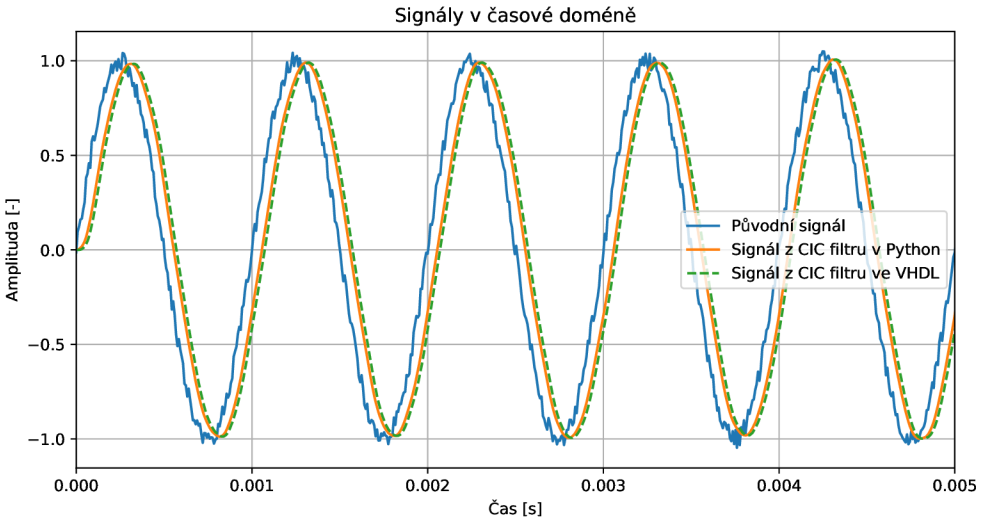
Test implementace `cic`

Pro testování CIC filtru v FPGA pro HDL modul byl vytvořen testbench. Testování filtru probíhalo na totožných vstupních datech jako v případě FIR filtru. Pro zjištění, zda je CIC filtr korektně funkční bylo využito srovnání odezvy CIC filtru v FPGA a CIC filtru v `Python`. Byl navržen CIC filtr (pomocí metody váhových oken) typu dolní propust. Jako parametry pro návrh filtru byly zvoleny:

- Počet kroků filtru (Stages) = 2
- Zpoždění vstupujících do derivační části = 8
- Decimace = 1 (bez decimace)

Na základě parametrů byla spočítána odezva (konvenčně ta přesná pomocí Python) na testovací signál.

Jako testovací signál byl zvolen signál, který se skládá z harmonické složky (na frekvenci 1 kHz) a bílého šumu. Signál se šumem, společně s filtrovanými signály v časové doméně jsou znázorněny v grafu č. 6.29.



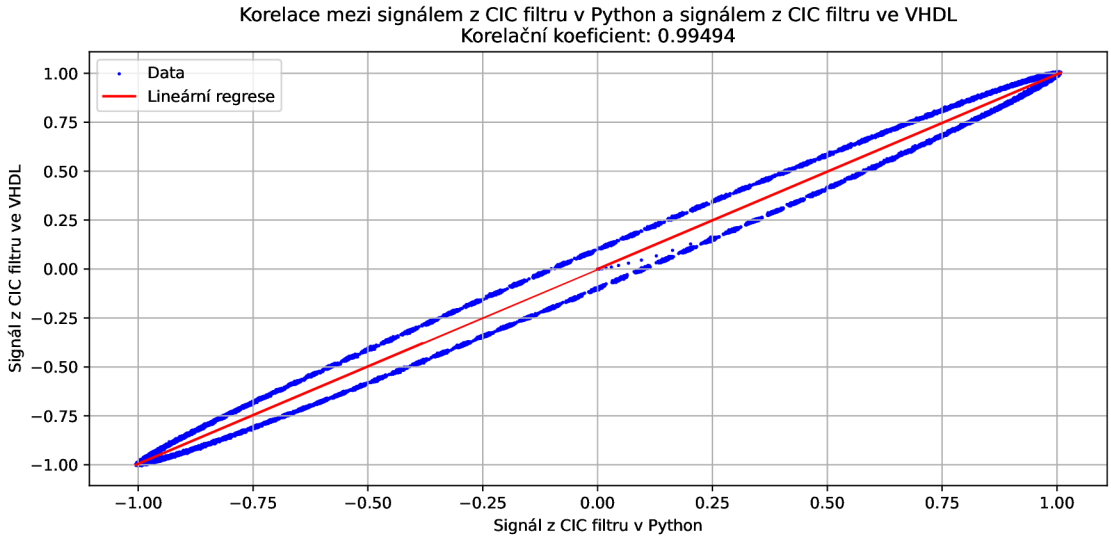
Obr. 6.29: Odezva CIC filtru na vstupní signál a jeho frekvenční charakteristika

Z časové domény je patrné, že skutečně došlo k filtraci šumu. Dále je zřejmé, že jsou signály z filtru v `Python` a v `VHDL` téměř identické, jsou jen o dva vzorky od sebe fázově posunuté, což je způsobeno zpožděním výpočtu ve `VHDL` oproti `Python`. Pro skutečné porovnání podobnosti signálu lze využít korelaci. Pro výpočet korelace mezi dvěma signály v `Python` lze použít funkci `numpy.corrcoef()` z knihovny `NumPy`. Tato funkce spočítá Pearsonovu korelační matici mezi dvěma signály. Pearsonova korelace měří lineární vztah mezi dvěma proměnnými. Vizualizace korelace je znázorněna na obr. 6.30. Korelační koeficient mezi signály z filtru v `Python` a v `VHDL` vyšel 0,99494, což defakto indikuje 99,3 % přesnost.

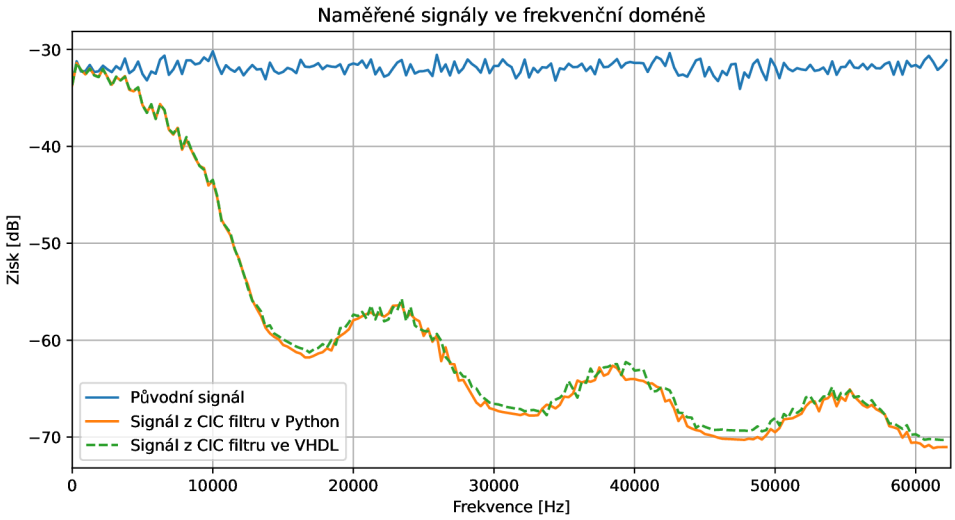
Další způsob, jak porovnat chování filtrů je vykreslení frekvenční charakteristiky filtru. Ta byla změřena pomocí bílého šumu a následně vypočtena jako průměrná FFT na základě 50 měření, kdy každé měření obsahovalo 400 vzorků.

Z grafu č. 6.29 je patrné, že odezva CIC filtru z `Python` je téměř totožná s CIC filtrem napsaným pro FPGA. Nepřesnosti, které jsou mezi signály patrné, jsou způsobeny zaokrouhlováním hodnot koeficientů a signálu, které vstupují do FPGA. Z grafu je patrné, že skutečně dochází k vyfiltrování nízko frekvenční složky signálu podle očekávání.

Na základě simulace a následné zpracování dat, zde popsané, lze očekávat, že se CIC filtr bude chovat korektně i v reálné implementaci.



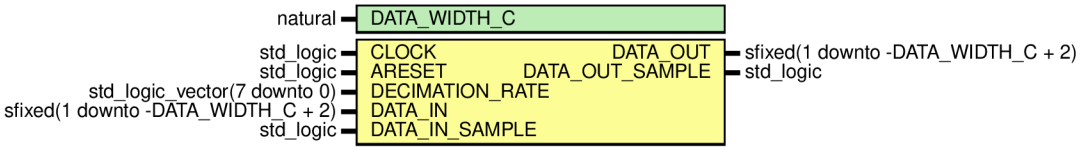
Obr. 6.30: Korelace mezi výstupy vypočítanými pomocí CIC filtru v `Python` a ve `VHDL`



Obr. 6.31: Frekvenční charakteristika signálů vypočítanými pomocí CIC filtru v `Python` a `VHDL`

6.3.4 Základní principy fungování entity `decimator`

Entita `decimator` provádí decimaci signálu. Tato entita je použita pouze pro typ filtru FIR. Není používána pro CIC filtr, protože CIC principiálně provádí decimaci. Vstupně výstupní rozhraní entity `decimator` je popsané na obr. 6.32. Entita obsahuje generický vstup pro volbu šířky vstupních dat.



Obr. 6.32: Diagram vstupů a výstupů entity `decimator`

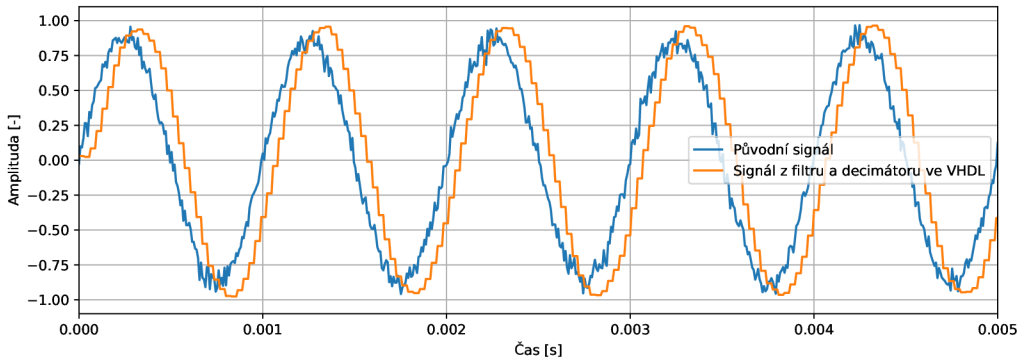
Logika entity je prostá. Entita s definovaným decimačním poměrem zahazuje vzorky dat. To je realizováno pomocí čítače, který při každém dočítání do hodnoty stanovené v registru `DECIMATION_RATE` vloží data na výstup. Tímto způsobem lze převést rychlý tok dat na pomalejší, a tedy snížit požadavky na rychlost komunikace.

Analýza implementace `decimator`

Pro implementaci je potřeba 15 LUT, 11 registrů. Maximální frekvence hodinového signálu je stanovena na 178,9 MHz.

Test implementace `decimator`

Testování entity probíhalo následujícím způsobem. Nejdříve byl vyfiltrován signál se šumem vhodným filtrem typu dolní propust (v tomto případě FIR) a poté přiveden do bloku decimace. Původní signál byl vzorkován vzorkovací frekvencí 125 kHz. Sledovaný signál má frekvenci 1 kHz, kdy podle Nyquistova teorému je nutné takový signál vzorkovat frekvencí větší než $2 \cdot 1kHz$, tak aby bylo možné zpětně signál správně rekonstruovat. Na obr. 6.33 je patrný signál po filtraci antialliasing filtrem a 4x decimaci.



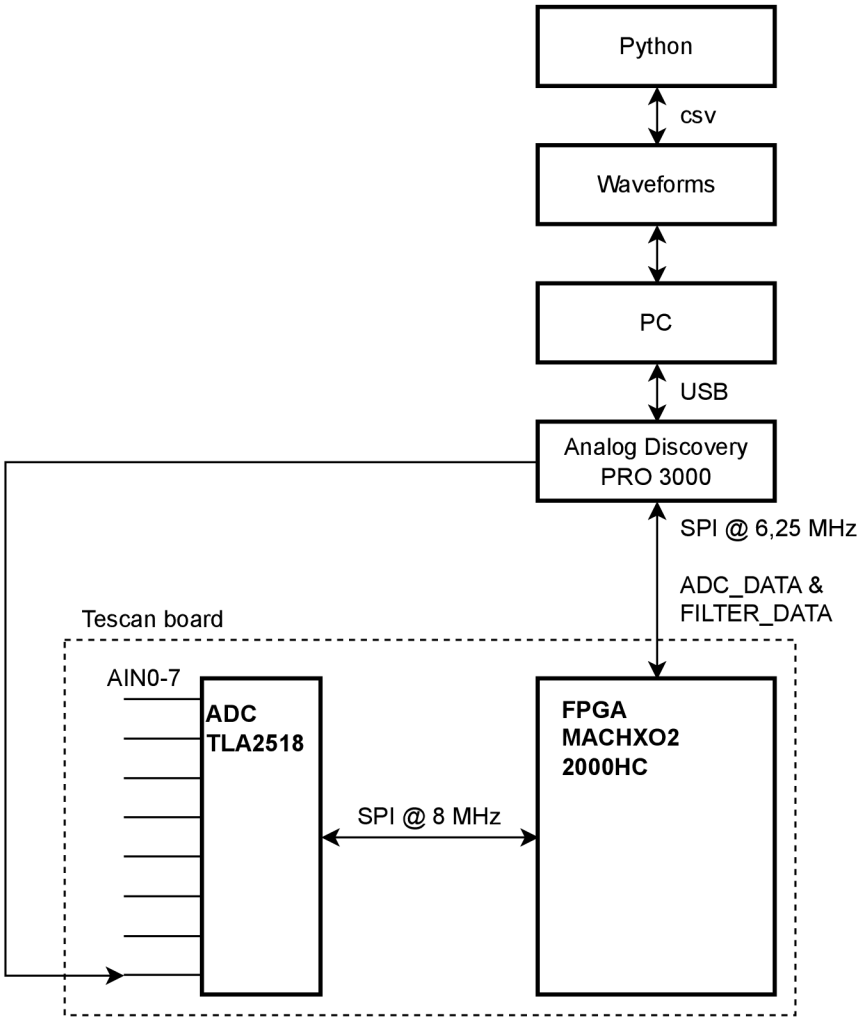
Obr. 6.33: Diagram vstupů a výstupů entity `decimator`

6.4 Vyrovnávací paměť

V blokovém schéma 6.1 je ideově znázorněna entita `Buffer`. Entita nebyla implementována pro tuto aplikaci (v komunikaci z nadřazeného systému je totiž zaručena včasnost vyčítání vzorků). Data jsou z výstupu filtru přímo zapisována do registru s danou frekvencí. Pokud by včasnost zaručena nebyla, tak by bylo vhodné implementovat entitu vyrovnávací paměti např. datovým typem FIFO. Nicméně i tak by muselo být zaručeno dostatečně rychlé vyčítání vzorků z nadřazeného systému, tak aby nedošlo k přeplnění vyrovnávací paměti.

7 Testování HDL modulu pro předzpracování dat na reálné aplikaci

V této kapitole jsou uvedeny příklady testování navrženého HDL modulu pro předzpracování dat. Jsou uvedena konkrétní testovací konfigurace FIR filtru a CIC filtru. Všechny vysvětlované zdrojové kódy jsou k dispozici na elektronické příloze (struktura souborů je uvedena v příloze A). Měřící řetězec je blokově uveden na obr. 7.1.

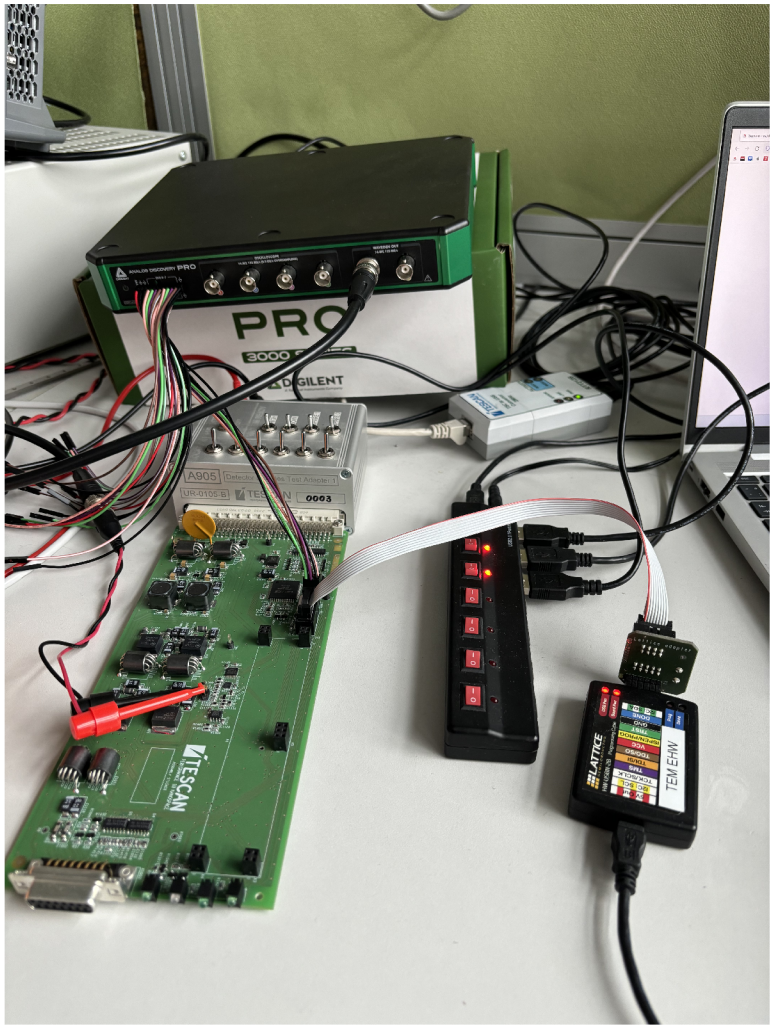


Obr. 7.1: Blokové schéma měřicího řetězce pro testování modulu pro předzpracování dat

Měřící řetězec se skládá z karty firmy Tescan na které se nachází FPGA MachXO2 s nahranou konfigurací pro předzpracování dat a A/D převodník TLA2518. Přístroj Analog Discovery PRO 3000 slouží pro generování signálu pro A/D převodník (na kanále 7) a pro komunikaci s FPGA jako SPI Master. Pomocí obslužného software Analog Discovery PRO 3000 je napsán měřící skript, který vyčítá data z

registrů `ADC_DATA` a `FILTER_DATA` a ukládá je do .csv souboru. Následně jsou data z .csv souboru analyzována pomocí `Python` a knihovny `SciPy` a je vykreslen průběh změřených signálů v časové i frekvenční doméně.

Pro testování je použitý kanál AIN7 A/D převodníku TLA2518 a vzorkovací frekvence 125 kS/s. Frekvenční charakteristika filtrů je měřena pomocí přivedení bílého šumu na vstupní kanál A/D převodníku. Měření je prováděno pro 200 tisíc vzorků, což je relativně dostatečný počet vzorků na otestování filtru. Na obr. 7.2 jsou patrné prostředky použité pro měření, karta firmy TESCAN a Lattice programátor pro FPGA připojen přes JTAG.



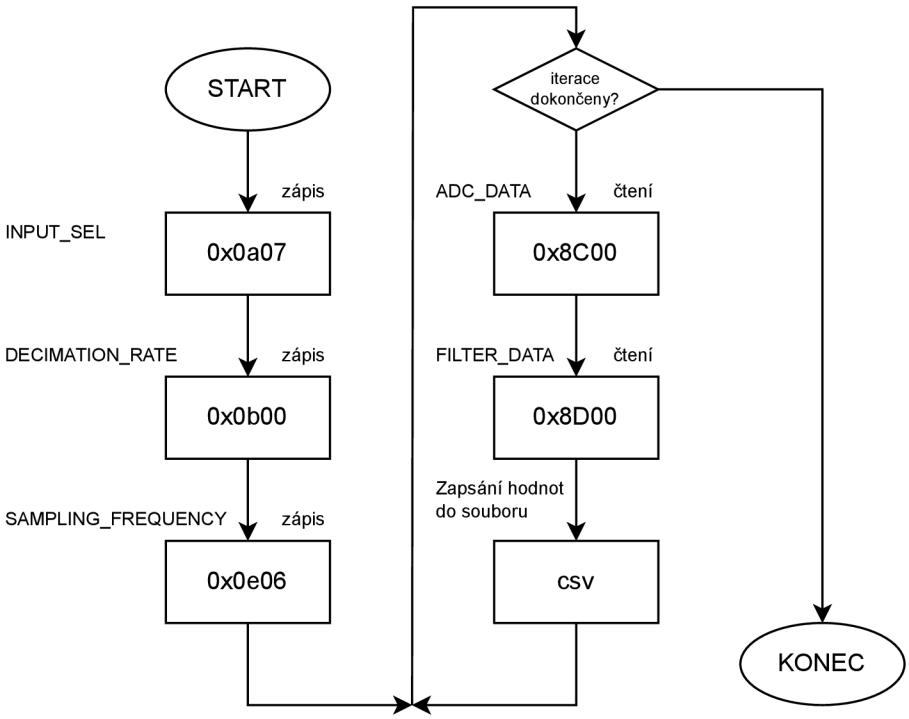
Obr. 7.2: Foto zapojení při testování modulu pro předzpracování dat

Měřící skript pro Analog Discovery PRO 3000

Software Waveforms umožňuje uživateli psát vlastní měřicí skripty pro zařízení, které podporují daný typ komunikace. Skript je složen ze dvou částí - inicializační a smyčkové. Inicializační část je provedena pouze na začátku a smyčková část je

prováděna neustále po určitý počet iterací. V inicializační části se typicky nachází data pro konfiguraci zařízení a v smyčkové je realizováno vyčítání dat s určitou periodou. Prostředí Waveforms dokáže zaručit provádění operací v reálném čase, což znamená, že data lze vyčítat přesně s určitou přesně stanovenou periodou. To je v případě provádění příkazů přímo ze systému Windows nereálné. Vyčítání dat z A/D převodníků je časově kritické, takže je nutné na to brát ohled.

Algoritmus sběru dat z modulu pro předzpracování dat včetně hodnot, které jsou po SPI odesílány je uveden na obr. 7.3.



Obr. 7.3: Algoritmus sběru dat z modulu pro předzpracování dat

Na začátku inicializačního skriptu probíhá SPI komunikace, která nastavuje vstupní kanál A/D převodníku, decimační poměr a vzorkovací frekvenci. Poté se ve smyčce vyčítají data z FPGA z registru s daty z A/D převodníku a z registru s daty z filtru. Data jsou ukládána průběžně do .csv souboru, který je poté analyzován.

7.1 Testování FIR filtru typu průměrování 11. řádu

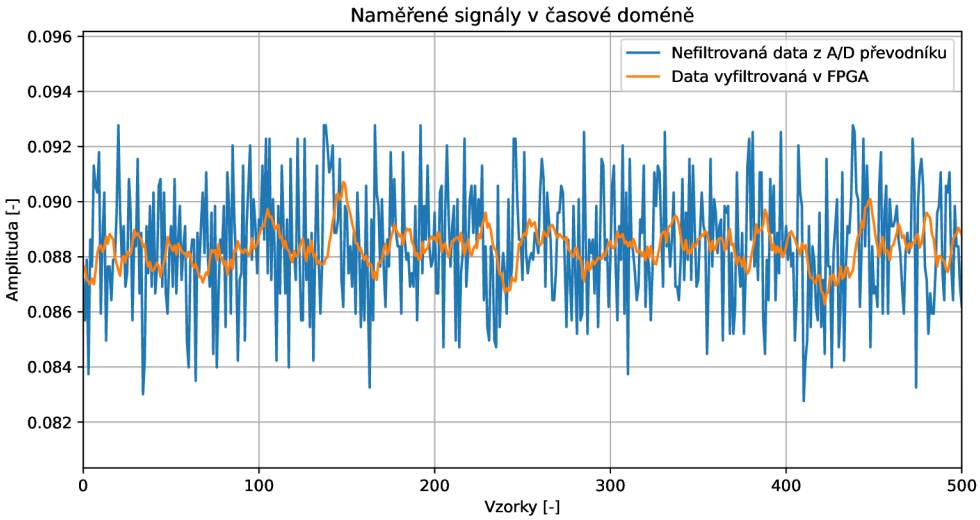
V této konfiguraci má FIR filtr nastaveny koeficienty pro průměrování 11. řádu. To znamená, že všechny koeficienty mají hodnotu $\frac{1}{11}$. Koeficienty vytvořené pomocí skriptu v `Python` ve `fixed_point` reprezentaci s velikostí 16 bitů pro `VHDL` kód jsou uvedeny ve výpisu č. 6.

```
--- GENERATED FIR FILTER COEFFICIENTS FOR VHDL ---
constant FILTER_ORDER_C      :   natural := 11;
constant COEFF_C             :   COEFF_T := (
to_sfixed(0.09090909090909091, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.09090909090909091, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.09090909090909091, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.09090909090909091, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.09090909090909091, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.09090909090909091, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.09090909090909091, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.09090909090909091, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.09090909090909091, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.09090909090909091, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.09090909090909091, 1, -COEFF_WIDTH_C + 2));
```

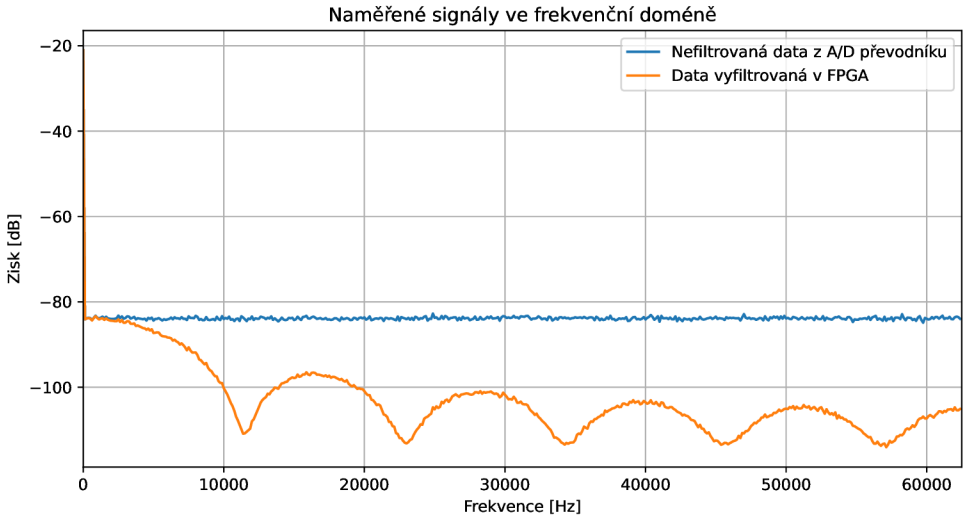
Výpis 6: Koeficienty ve `VHDL` pro FIR filtr typu průměrování 11. řádu

Jako metoda měření frekvenční charakteristiky FIR filtru typu průměrování 11. řádu byla zvolena metoda, kdy je na 7. kanál A/D převodník generován signál typu bílý šum. Následně jsou data za A/D převodníkem a za filtrem vyčítána z FPGA. Časová a frekvenční doména změřených signálů je patrná z obr. 7.4.

Pro lepší vykreslení frekvenční charakteristiky je lepší provést měření několik, z každého vypočítat FFT a tyto FFT pak zprůměrovat a vykreslit pouze výslednou frekvenční charakteristiku. Díky tomu je výsledná křivka hladší a čitelnější. Pro tento případ bylo 200 tisíc naměřených vzorků rozděleno na 200 měření a na základě nich byla stanovená průměrná FFT. Z grafů na obr. 7.4 a 7.5 je pak dobře patrná frekvenční charakteristika filtru, to má tvar přesně takový, jaká je teoreticky očekávána. Lze konstatovat, že praktická implementace FIR filtru typu průměrování 11. řádu je plně funkční.



Obr. 7.4: Časová doména signálu při použití FIR filtru typu průměrování 11. řádu



Obr. 7.5: Frekvenční doména signálu při použití FIR filtru typu průměrování 11. řádu

Analýza implementace FIR filtru typu průměrování 11. řádu

Využití zdrojů pro implementaci entity `fir.vhd` je následující: 206 LUT, 227 registrů. Maximální frekvence hodinového signálu je stanovena na 66,1 MHz.

Celkové využití zdrojů FPGA po implementaci pro konkrétní konfiguraci je následující: 391 LUT, 536 registrů. Maximální frekvence hodinového signálu je stanovena na 104,8 MHz, přičemž použitá hodnota hodinového signálu je 38 MHz. Počet potřebných vstupně-výstupních pinů je 12.

Je nutno podotknout, že pokud by byla zapotřebí skutečně efektivní implementace filtru typu průměrování, tak lze takový filtr navrhnout v FPGA daleko více optimalizovaně, než pomocí násobení. Lze jej navrhnout jen za pomocí sčítaní a odčítání. Výsledkem by pak bylo výrazné ušetření zdrojů oproti jeho implementaci ve FIR filtru. Nicméně tento typ filtru byl zvolen pro testování implementace této práce jako demonstrace funkčnosti navrženého řešení.

7.2 Testování FIR filtru typu dolní propust 11. řádu

V této konfiguraci má FIR filtr nastaveny koeficienty pro typ dolní propusti 11. řádu. Filtr byl navržen v `Python` funkcí `firwin` z balíčku `scipy.signal`. Koeficienty FIR filtru byly navrhovány pomocí Hammingova okna. Parametry návrhu jsou následující:

- Vzorkovací frekvence: 125 kHz
- Mezní frekvence: 10 kHz
- Řád filtru: 11

Koeficienty vytvořené pomocí skriptu v `Python` ve `fixed_point` reprezentaci s velikostí 16 bitů pro `VHDL` kód jsou uvedeny ve výpisu č. 7.

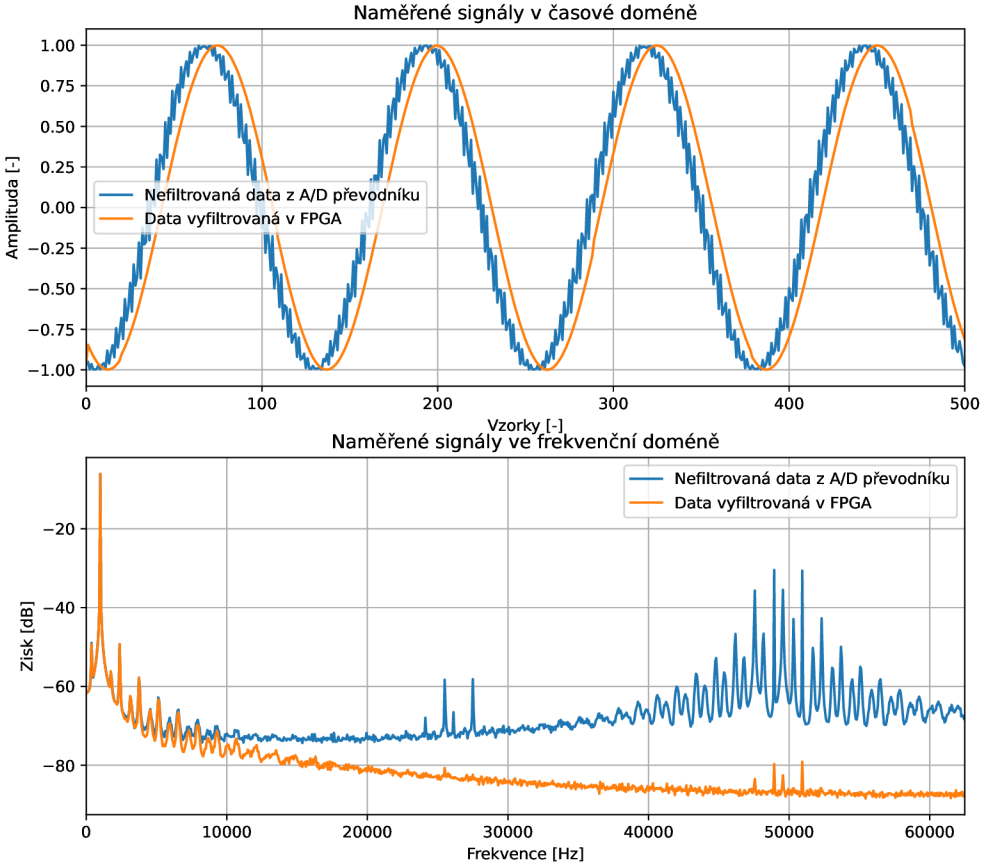
```
--- GENERATED FIR FILTER COEFFICIENTS FOR VHDL ---
constant FILTER_ORDER_C     :   natural := 11;
constant COEFF_C            :   COEFF_T := (
to_sfixed(0.004059059965798471, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.016387751788843455, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.057125427346512554, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.12429293720894577, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.18966057293847272, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.21694850150285394, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.18966057293847272, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.12429293720894577, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.057125427346512554, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.016387751788843455, 1, -COEFF_WIDTH_C + 2),
to_sfixed(0.004059059965798471, 1, -COEFF_WIDTH_C + 2));
```

Výpis 7: Koeficienty ve `VHDL` pro FIR filtr typu dolní propust 11. řádu

Pro demonstraci funkčnosti FIR filtr typu dolní propust 11. řádu byl tentokrát zvolen vstupní signál, který obsahuje na vyšší frekvencí signál určený na filtraci.

Opět jako v předchozím případě bylo provedeno 200 tisíc měření vzorků a poté rozděleno na 200 měření a na základě nich byla stanovená průměrná FFT. Časová a frekvenční doména změřených signálů je patrná z obr. 7.6.

Je teoreticky očekáváno, že podle definice bude filtr nízké frekvence propouštět a vyšší eliminovat (mezní frekvence je v tomto případě 10 kHz), což je přesně patrné, jak na časové (signál za filtrem je vyhlazený), tak i na frekvenční charakteristice. Lze konstatovat, že praktická implementace FIR filtru typu dolní propust 11. řádu je plně funkční.



Obr. 7.6: Časová a frekvenční doména signálu při použití FIR filtr typu dolní propust 11. řádu

Analýza implementace FIR filtru typu dolní propust 11. řádu

Využití zdrojů pro implementaci entity `fir.vhd` je následující: 218 LUT, 244 registrů. Maximální frekvence hodinového signálu je stanovena na 64,8 MHz.

Celkové využití zdrojů FPGA po implementaci pro konkrétní konfiguraci je následující: 412 LUT, 520 registrů. Maximální frekvence hodinového signálu je stanovena na 64,4 MHz, přičemž použitá hodnota hodinového signálu je 38 MHz. Počet potřebných vstupně-výstupních pinů je 12.

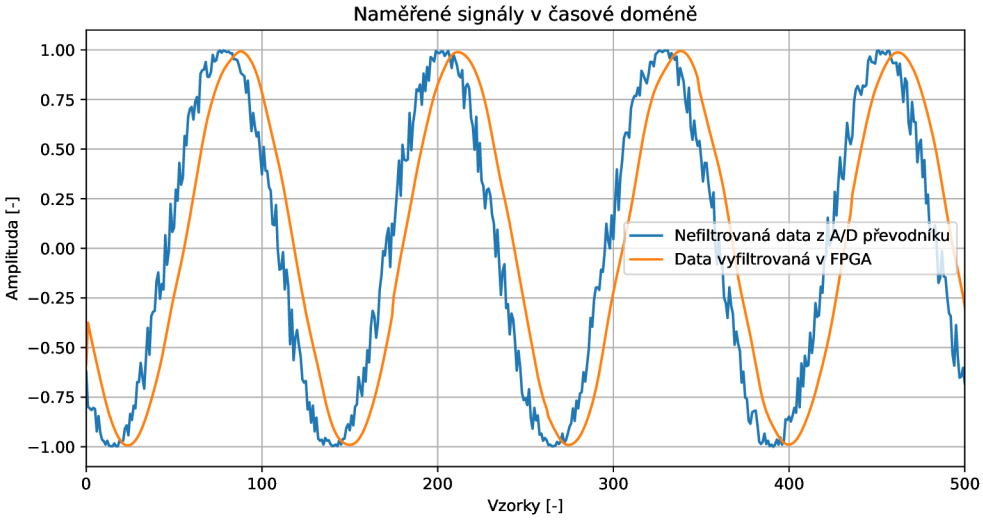
7.3 Testování CIC filtru

V této konfiguraci je CIC filtr nastaven na 3 kroky a zpoždění členů do derivační části je 8. Parametry nastavení filtru ve VHDL jsou uvedeny ne výpise č. 8. Časová doména naměřených signálů je patrná z obr. 7.7.

```
1 constant DECIMATION_RATE_C   : natural := 1;
2 constant STAGES_C            : natural := 3;
3 constant COMB_DELAY_COUNT_C : natural := 8;
```

Výpis 8: Nastavení ve VHDL pro CIC filtr

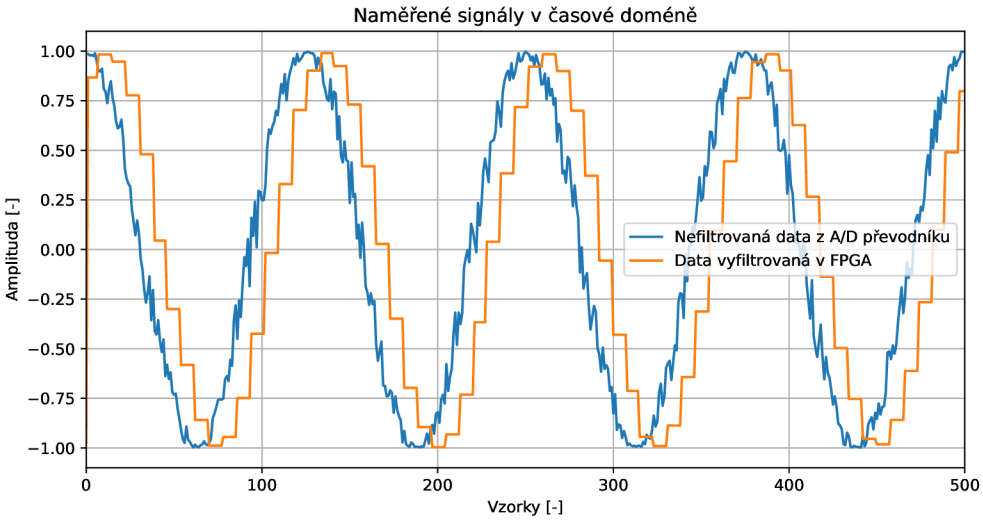
Pro demonstraci funkčnosti CIC filtru byl zvolen vstupní signál, který obsahuje na vyšší frekvencí signál určený na filtraci. Opět jako v předchozím případě bylo provedeno 200 tisíc měření vzorků a poté rozděleno na 200 měření a na základě nich byla stanovená průměrná FFT.



Obr. 7.7: Časová doména signálu při použití CIC filtru bez decimace

V tomto případě byl využit CIC filtr bez decimace. Je teoreticky očekáváno, že podle definice bude filtr nízké frekvence propouštět a vyšší eliminovat, což je přesně

patrné, jak na časové (signál za filtrem je vyhlazený), tak i na frekvenční charakteristice. Lze konstatovat, že praktická implementace CIC filtru je plně funkční. Na obr. 7.8 je znázorněn signál po filtraci CIC filtrem s decimací 8. Tímto způsobem lze prakticky efektivně CIC filtry využívat. Z původní vzorkovací frekvence 125 kHz, je nyní signál převzorkován na frekvenci 15,625 kHz.



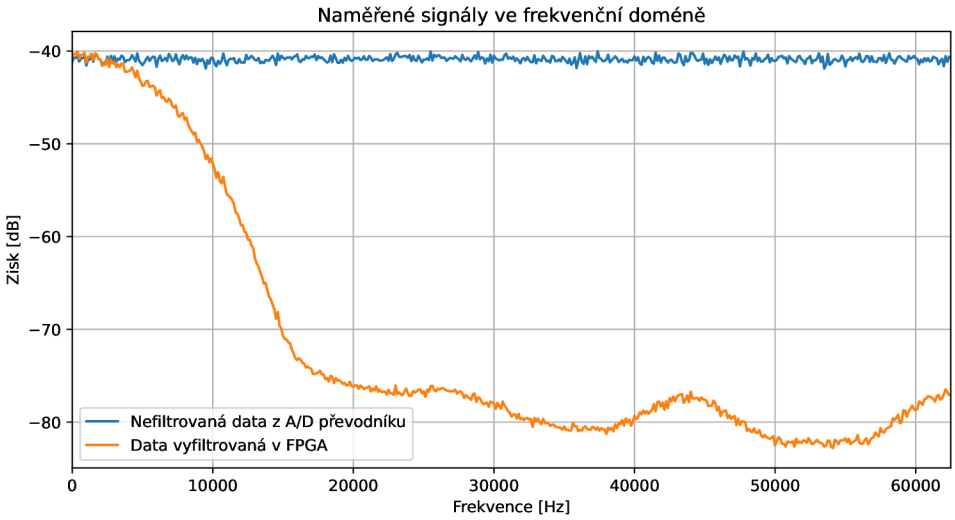
Obr. 7.8: Časová doména signálu při použití CIC filtru s decimací 8

Frekvenční charakteristiku filtru lze naměřit pomocí přivedení bílého šumu na vstup, podobně jako v předchozích praktických testech v této kapitole. Byla vypočítána jako průměrná FFT na základě 200 měření. Frekvenční charakteristika CIC filtru je patrná na obr. 7.9.

Analýza implementace CIC filtru

Využití zdrojů pro implementaci entity `cic.vhd` je následující: 28 LUT, 344 registrů. Maximální frekvence hodinového signálu je stanovena na 104,8 MHz.

Celkové využití zdrojů FPGA po implementaci pro konkrétní konfiguraci je následující: 269 LUT, 464 registrů. Maximální frekvence hodinového signálu je stanovena na 104,8 MHz, přičemž použitá hodnota hodinového signálu je 38 MHz. Počet potřebných vstupně-výstupních pinů je 12. Oproti implementacím filtrů FIR si lze všimnout, že využití zdrojů podstatně nižší, pro implementaci je využito pouze 28 LUT, výpočet filtru je rychlejší i maximální frekvenci hodinového signálu lze použít vyšší.



Obr. 7.9: Frekvenční doména CIC filtru

7.4 Shrnutí a srovnání výsledků testování HDL modulu pro předzpracování dat

V této kapitole byly uvedeny tři konfigurace filtrů implementovaných v FPGA, byly otestovány s reálným signálem, změřeny, uloženy do souboru a následně pomocí `Python` a knihovny `Scipy` analyzovány. Konfigurace byly:

- FIR filtr typu průměrování 11. řádu
- FIR filtr typu dolní propust 11. řádu s mezní frekvencí 10 kHz
- CIC filtr se 3 kroky a zpožděním členů derivační části 7

Každý filtr byl testován určitým signálem a pak vykreslena jeho časová i frekvenční doména. Srovnání využití zdrojů FPGA je uvedeno v tab. 7.1.

Filtr	LUT	Registr D	f_{clk}
FIR průměrování 11. řádu	206	227	66,1 MHz
FIR dolní propust 11. řádu	218	244	64,8 MHz
CIC 3. řád, zpoždění 7	28	344	104,8 MHz

Tab. 7.1: Porovnání využití zdrojů FPGA filtry

Je patrné, že CIC spotřebuje výrazně méně LUT, než FIR filtr. To je dáno tím, že není potřeba implementovat násobičku jako v případě FIR filtru. Na spotřebě zdrojů FPGA rozhoduje konkrétní konfigurace filtru. V případě CIC filtru je využito více registrů než v případě FIR filtru, což je dáno potřebou implementace registrů pro

zpoždění signálu do derivační části. Nicméně tato skutečnost neovlivňuje tolik časování, jako implementace násobičky a tudíž poměrně velkého množství kombinační logiky v případě FIR filtru. Vyšší frekvence hodinového signálu lze dosáhnout u CIC filtru, v tomto konkrétním případě až 178,6 MHz. V FPGA MachXO2 CMXO2-2000HC je násobička implementována pomocí LUT, protože neobsahuje hardwarovou násobičku. Pro násobičku složenou z LUT pro násobení signálů o velikost 12 a 16 je využito 192 LUT. Kdyby tato implementace filtru byla implementována pro FPGA obsahující DSP bloky (vhodnější FPGA pro aplikace signálového zpracování) podporující operace multiply and accumulate, tak by došlo k využití těchto bloků a výsledné časování by bylo mnohem lepší.

Co se týče limitů FPGA MachXO2 CMXO2-2000HC, tak maximální implementovatelný počet koeficientů pro FIR filtr se se pohybuje okolo 91 koeficientů, na větší počet není v FPGA dostatečné množství zdrojů. Záleží však na hodnotě koeficientů a případně následné optimalizaci, kdy například koeficienty s hodnotou blížící se nule lze optimalizovat, protože lze říci rovnou, že výsledek bude nula. Nicméně taková optimalizace by byla vhodná pouze pro určité koeficienty a nastavení FIR filtrů, tudíž optimalizací by zároveň mohlo dojít k zhoršení univerzálnosti použití FIR filtru.

Pro konkrétní FPGA jsou od jejich výrobců dostupné IP bloky, ve kterých jsou jak FIR, tak CIC filtry implementovány. Nicméně pro FPGA použité v této práci výrobce tyto IP bloky nedodává. FPGA MachXO2 CMXO2-2000HC není přímo vhodné pro zpracování signálu právě z důvodu absence DSP bloků, avšak pro jednoduchou filtraci signálu jej lze klidně použít.

Závěrem lze konstatovat, že se podařilo jak teoreticky, tak i prakticky implementovat a otestovat filtry, které mohou sloužit pro filtraci signálu s nastavitelnými parametry. FIR filtry lze nakonfigurovat tak, aby je bylo možné využít jako dolnopropustní, hornopropustní filtry, nebo filtry ostatního typu. Nastavení záleží na koeficientech. CIC filtry jsou dolnopropustní filtry, které jsou pro implementaci vyžadují minimální počet zdrojů. Často jsou využívány při decimování nebo interpolování signálu.

Závěr

Cílem diplomové práce bylo teoreticky navrhnout koncepci modulu v FPGA pro předzpracování dat.

V první kapitole byly teoreticky popsány architektury A/D převodníků. To zahrnovalo uvedení konkrétních principů A/D převodníků s postupnou aproximací, A/D převodník Delta-sigma, převodníky integrační a komparační. V kapitole jsou také popsány základní pojmy jako vzorkování nebo kvantizace, které se týkají digitálního zpracování analogových dat.

Druhá kapitola se věnuje zvoleným metodám pro předzpracování dat. Je zde uvedena funkce filtru s konečnou impluzní odezvou (FIR), včetně jeho principu a možných aplikací. Dále je vysvětlena funkce káskádového filtru s integrátorem (CIC). Uvedené filtry jsou v práci implementovány a otestovány v programovatelném hradlovém poli. Je demonstrována jejich funkce na praktické úloze, jejíž popis je uveden v kapitole č. 7. Typicky je na vstup přiveden signál obsahující šum, který je vhodným způsobem odfiltrován pomocí FPGA.

Ve třetí kapitole jsou teoreticky popsány obvody FPGA. To zahrnuje popis jejich architektur a vnitřní struktury. Jsou zde uvedeny i typy konfiguračních technologií FPGA. Kapitola se věnuje průběhu návrhu digitálního obvodu pro FPGA a jsou vysvětleny základní pojmy jako syntéza, simulace a proces rozmístění signálových cest uvnitř FPGA. V kapitole je stručně popsána problematika přechodu mezi hodinovými doménami. Dále se v kapitole nachází způsob pro předávání signálů pomocí AXI-style ready/valid handshake. Kapitola popisuje vybrané problematiky, které musely být zohledněny při návrhu konfigurace do FPGA.

Čtvrtá kapitola se věnuje sériovému rozhraní pro komunikaci SPI, která je v práci používána. Věnuje se základním principům komunikace, popisu rozhraní, včetně popisu komunikačních režimů SPI. Jsou uvedeny praktické aplikace i výhody a nevýhody komunikace pomocí této sběrnice. Mezi výhody patří rychlý přenos dat, jednoduchost implementace a možnost full-duplex komunikace. Nevýhodou je pak to, že se jedná o synchronní přenos dat, který je nevhodný pro přenos na velké vzdálenosti.

V páté kapitole jsou uvedeny používané prostředky pro realizaci práce. To zahrnuje výběr A/D převodníku TLA2518 a vhodného FPGA MachXO2 CMXO2-2000HC. Pro hardware jsou uvedeny základní parametry včetně frekvencí komunikace mezi FPGA a A/D převodníkem a mezi FPGA a vnějším zařízením pro vyčítání dat z FPGA. V kapitole je uveden i používaný software pro syntézu `VHDL` kódu Symplify Pro. Je uveden testovací framework pro jazyk HDL VUnit. Je popsána zařízení, které slouží jako logický analyzátor a generátor signálu pro testování FPGA konfigurace Digilent ADP3250 Analog Discovery Pro.

Šestá kapitola zahrnuje teoretický popis navrhovaného zařízení ve VHDL. Obsahuje přehledné blokové schéma entit modulu (obr. 6.1). Je zde uveden detailnější popis entit, čímž se rozumí detailní popis jejich funkce, včetně nákresu stavových automatů implementovaných ve VHDL. Popsána je komunikace nadřazeného systému s modulem. Jsou uvedeny konfigurační a stavové registry, které slouží pro konfiguraci zařízení. Jsou uvedeny 8bitové registry, které jsou uvedeny v tab. č. 6.1. Zahrnuje popis možných řešení a popis zvoleného řešení. Pro implementaci FIR filtru byl zvolen sériový přístup tak, aby bylo docíleno sdílení zdrojů v FPGA. Správná implementace je ověřena pomocí srovnání odezev z Python knihovny SciPy a odezvy ze simulace implementace ve VHDL. Odezvy byly téměř totožné. Nepřesnosti byly způsobeny zaokrouhlováním, přičemž Python počítá s datovým typem float64 a VHDL implementace počítá s datovým typem fixedpoint16. Výstupem kapitoly je funkční implementace modulu pro předzpracování dat.

V sedmé kapitole je uvedena demonstrace funkčnosti celého řešení na praktické úloze. Jsou uvedeny tři konfigurace FIR filtru - průměrování 11. řádu, dolní propust 11. řádu horní propust 11. řádu. Je uvedena i konfigurace CIC filtru. Měřicí úloha se skládala z logického analyzátoru a generátoru signálu s šumem, který byl následně pomocí vytvořeného skriptu v jazyce Python zpracován a byla vykreslena jeho časová a frekvenční odezva (pomocí algoritmu FFT). Kapitola obsahuje grafy, které jsou vygenerovány na základě velkého množství naměřených dat tak, aby byly věrně demonstrovány vlastnosti navržených filtrů. Je uvedeno shrnutí výsledků testování HDL modulu pro předzpracování dat, což zahranuje počty využitých zdrojů v FPGA. V elektronické příloze je přiloženo doprovodné video (demo.mp4), které demonstruje praktickou implementaci FIR filtru typu dolní propust. Ve videu je ukázán postup sběru dat a jejich analýzy.

Během realizace diplomové práce vzniklo celkem 15 souborů se zdrojovým kódem v jazyce VHDL s celkovou velikostí 84,9 kB a 4 soubory se zdrojovým kódem v jazyce Python s celkovou velikostí 10,7 kB. Přehled souborů je uveden v příloze A. V souborech se nachází jak samotný kód zdrojový kód pro FPGA, tak i soubory pro simulaci VHDL kód. V souborech jsou i skripty v jazyce Python, které byly použity pro testování navrhovaných řešení a analýzu řešení diplomové práce.

Budoucnost a další navazující rozšíření této práce

Jak již bylo zmíněno, práce je koncipována tak, aby byla široce využitelná v dalších aplikacích v FPGA společností TESCAN GROUP, a. s. Co se týče dalších metod pro předzpracování dat, tak lze v budoucnu implementovat další filtry typu polyfázový FIR filtr (Polyphase FIR filter), což je podobně jako CIC filtr určen k efektivní konverzi vzorkovacího kmitočtu. Stejně tak lze implementovat i Notch filtr. Každý filtr má specifické vlastnosti a je vhodný pro určitou aplikaci v oblasti digitálního zpracování signálů. Pro využití výhod paralelní implementace FIR filtru by bylo nutné využít hradlové pole s více logickými bloky, ideálně s DSP bloky, ve kterých je přímo hardwarově implementována sčítačka a násobička (v blocích typu DSP), což tak umožní velmi rychlý a efektivní výpočet. Stejně tak by bylo možné navýšit řád FIR filtru několikanásobně a mít tím specifičtější filtraci signálu, aby bylo možné přesněji vyhovět požadavkům na zpracování signálů.

Literatura

[1] PINKER, J., POUPA, M. *Číslicové systémy a jazyk VHDL.* Praha: 2009. ISBN 80-7300-198-5.

[2] ŠŤASTNÝ, J. *FPGA prakticky.* Praha: 2011. ISBN 978-70-7300-2.

[3] WAHO, T. *Introduction to Analog-to-Digital Converters Principles and Circuit Implementation.* River Publishers, 2019. 288 s. ISBN 978-87-7022-101-6.

[4] TEXAS INSTRUMENTS INC. *TLA2518 Small, 8-Channel, 12-Bit ADC with SPI Interface and GPIOs datasheet (Rev. B).* 2020. Datasheet.

[5] Lattice Semiconductor Corporation *MachXO2 Family Data Sheet* 2023. Datasheet.

[6] AMANO, H. *Principles and Structures of FPGAs* Springer Singapore, 2018. ISBN 978-981-13-0824-6.

[7] MATOUŠEK, P. *Rozšíření rozhraní pro řízení buňkového měniče pomocí FPGA.* Brno, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2022, 70 s. Bakalářská práce. Vedoucí práce: Ing. Petr Petyovský, Ph.D.

[8] SMITH, Grant Maloy. *Types of ADC Converters - The Ultimate Guide.* Online. DEWESoft. Dostupné z: https://dewesoft.com/blog/types-of-adc-converters. [cit. 2023-10-07].

[9] VOJÁČEK, A. *Použití filtrů FIR v digitálním zpracování signálů.* Online. Dostupné z: https://automatizace.hw.cz/clanek/2005110801. [cit. 2023-10-11].

[10] MARINOV, D. *Part 2: Finite impulse response (FIR) filters.* Online. Dostupné z: https://vhdlwhiz.com/part-2-finite-impulse-response-fir-filters/. [cit. 2023-10-16].

[11] MARINOV, D. *Part 3: FIR filter types.* Online. Dostupné z: https://vhdlwhiz.com/part-3-fir-filter-types/. [cit. 2023-10-19].

[12] National Instruments. *Decimation (Digital Filter Design Toolkit)* Online. Dostupné z: https://www.ni.com/docs/en-US/bundle/labview-digital-filter-design-toolkit-api-ref/page/lvdfdtconcepts/dfd_decimation.html. [cit. 2023-10-16].

[13] KAFKA, V. *Sigma-delta modulátor.* Online. Dostupné z: https://hippo.feld.cvut.cz/UserFiles/File/ads/Kafka.pdf. [cit. 2023-10-23].

[14] DANĚK, M. *Programovatelná hradlová pole – FPGA.* Online. Dostupné z: https://automa.cz/cz/casopis-clanky/programovatelna-hradlova-pole-fpga-2006_02_30930_672/. [cit. 2023-10-23].

[15] MAXFIELD, C. M. *Fundamentals of FPGAs: What Are FPGAs and Why Are They Needed? .* Online. Dostupné z: https://www.digikey.cz/en/articles/fundamentals-of-fpgas-what-are-fpgas-and-why-are-they-needed. [cit. 2023-10-23].

[16] LYONS, R. *A Beginner's Guide To Cascaded Integrator-Comb (CIC) Filters.* Online. Dostupné z: https://www.dsprelated.com/showarticle/1337.php. [cit. 2023-10-30].

[17] VERBEURE, T. *An Intuitive Look at Moving Average and CIC Filters* Online. Dostupné z: https://tomverbeure.github.io/2020/09/30/Moving-Average-and-CIC-Filters.html. [cit. 2023-12-19].

[18] JENSEN, J. J. *How the AXI-style ready/valid handshake works* Online. Dostupné z: https://vhdlwhiz.com/how-the-axi-style-ready-valid-handshake-works/. [cit. 2024-02-19].

[19] ASPLUND, L. *VUnit User Guide* Online. Dostupné z: https://vunit.github.io/user_guide.html. [cit. 2024-02-19].

[20] DIGILENT *Analog Discovery Pro 3000 Series: Portable High Resolution Mixed Signal Oscilloscopes* Online. Dostupné z: https://digilent.com/shop/adp-3000-series/. [cit. 2024-02-19].

[21] WIKIPEDIA contributors *Semiconductor detector* Online. Dostupné z: https://en.wikipedia.org/wiki/Semiconductor_detector. [cit. 2024-03-01].

Seznam symbolů a zkratek

A/D	Analogově-digitální převodník
CIC	Káskádový filtr s integrátorem - Cascaded Integrator Comb
D/A	Digitálně-analogový převodník
DSP	Číslicové zpracování signálu - Digital Signal Processing
EBR	Vestavná paměť typu RAM - Embedded block RAM
FFT	Rychlá Fourierova transformace - Fast Fourier Transform
FIFO	První dovnitř, první ven - First in, first out
FIR	Filtr s konečnou impulzní odezvou - Finite Impulse response
FPGA	Programovatelná hradlová pole - Field Programmable Gate Array
HDL	Jazyk pro popis hardware - Hardware Description Language
LCD	Displej z tekutých krystalů - Liquid Crystal Display
LSB	Nejméně významný bit - Least significant bit
LUT	Pravdivostní tabulka - Lookup Table
MSB	Nejvýznamnější bit - Most significant bit
RAM	Paměť s náhodným přístupem - Random Acccess Memory
SPI	Sériové periferní rozhraní - Serial peripheral interface
UART	Univerzální asynchronní rozhraní - Universal asynchronous receiver-transmitter
VHDL	VHSIC jazyk pro popis hardware - VHSIC Hardware Description Language

A Obsah elektronické přílohy

- /
  - `readme.md` ..... Doplňující popis elektronické přílohy
  - `diplomova_prace.pdf` ..... Elektronická verze práce
  - `demo.mp4` ..... Video demonstrující praktické řešení práce
  - `PreprocessingHDL`
    - `hdl` ..... Zdrojové soubory ve `VHDL`
      - `common`
        - `ADC`
          - `ADC.vhd`
          - `TLA2518.vhd`
        - `Comm`
          - `communication_core.vhd`
          - `spi_slave.vhd`
        - `Filter`
          - `cic.vhd`
          - `decimator.vhd`
          - `filter.vhd`
          - `fir.vhd`
        - `preprocessing_module.vhd`
        - `preprocessing_module_pkg.vhd`
      - `constraints`
        - `PreprocessingHDL.lpf`
      - `preprocessing_module_MachXO2.vhd`
    - `implem` ..... Projektové soubory pro Lattice Diamond
      - `PreprocessingHDL.ldf`
      - `PreprocessingHDL.xcf`
      - `PreprocessingHDL1.sty`
    - `python-playground` ..... Soubory pro simulování filtrů v `Python`
      - `decimation.py`
      - `filter_sandbox.py`
    - `real-test` ..... Soubory pro reálné testování
      - `csv` ..... Naměřená data z A/D převodníku a FPGA
        - `noise_fir_aver_11.csv`
        - `noise-cic_3stages_7dombdelay_8decimation.csv`
        - `sine-cic_3stages_7dombdelay_1decimation.csv`
        - `sine-cic_3stages_7dombdelay_8decimation.csv`
        - `sine-noise_fir_lp_11.csv`
        - `sine-noise-cic_3stages_7dombdelay_8decimation.csv`
      - `main.py` ..... `Python` skript pro analyzování naměřených dat
      - `preprocessing_module.dwf3work` ..... Měřicí skript pro Waveforms

```
/
├── testbench ..... Zdrojové soubory pro simulaci ve VHDL
│   ├── communication_core_tb.vhd
│   ├── filter_tb.vhd
│   ├── spi_slave_tb.vhd
│   ├── tla2518_tb.vhd
│   ├── vhdl_input-signal.txt
│   └── vhdl_output-signal.txt
└── vunit_run.py ..... Spouštěcí skript pro provedení VUnit testů
```