VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY
DEPARTMENT OF MICROELECTRONICS

PROSTŘEDÍ PRO VERIFIKACI DIGITÁLNÍCH FILTRŮ
SOFTWARE FOR DIGITAL FILTER VERIFICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jan Tesařík**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Marián Pristach**
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Mikroelektronika**
Ústav mikroelektroniky

Student: Bc. Jan Tesařík ***ID:*** 146978
Ročník: 2 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Prostředí pro verifikaci digitálních filtrů

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s funkcí digitálních filtrů a možnostmi jejich návrhu v programu Matlab. Navrhněte a vytvořte programové vybavení pro verifikaci digitálních filtrů typu CIC, FIR a IIR a systémů založených na digitálních filtrech. Vytvořené prostředí bude umožňovat analýzu systémů, generování verifikačního prostředí v jazyce SystemVerilog a s využitým programu Matlab ověření funkce jednotlivých digitálních filtrů. Na tvorbu programu použijte programovací jazyk C++.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání: 8.2.2016 ***Termín odevzdání:*** 26.5.2016

Vedoucí práce: Ing. Marián Pristach
Konzultant diplomové práce:

doc. Ing. Lukáš Fujcik, Ph.D., *předseda oborové rady*

ABSTRAKT

Diplomová práce se zabývá návrhem verifikačního prostředí pro analýzu systémů s digitálními filtry. Verifikační prostředí je napsáno v jazyce SystemVerilog a je generováno programem, který také obstarává generování vstupních dat pro systém filtrů. Pro získání referenčních dat je využito programového prostředí Matlab. Simulace navrženého zapojení s digitálními filtry probíhá v programu ModelSim. HIavním sledovaným parametrem je funkční pokrytí, které udává jak velká část HDL popisu byla otestována.

KLÍČOVÁ SLOVA

filtr, FIR, CIC, IIR, verifikace, SystemVerilog

ABSTRACT

Diploma thesis deals with design of verification environment for analyzing systems with digital filters. Verification environment is written in SystemVerilog language and it is generated by program, which is also providing generation of input data for system of filters. Matlab environment is used for gaining the reference data. The simulation of the designed involvement with digital filters is performed by program ModelSim. The most watched parameter is functional coverage which indicates how big part of the HDL description has been tested.

KEYWORDS

filter, FIR, CIC, IIR, verification, SystemVerilog

TESAŘÍK, J. *Prostředí pro verifikaci digitálních filtrů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 59 s. Vedoucí diplomové práce Ing. Marián Pristach.

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Technicka 12, CZ-61600 Brno
Czech Republic

http://www.six.feec.vutbr.cz

Experimentální část této diplomové práce byla realizována na výzkumné infrastruktuře vybudované v rámci projektu CZ.1.05/2.1.00/03.0072 **Centrum senzorických, informačních a komunikačních systémů (SIX)** operačního programu Výzkum a vývoj pro inovace.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Prostředí pro verifikaci digitálních filtrů“ jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové 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)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Mariánu Pristachovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....
(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Tato práce se věnuje problematice verifikace systémů s digitálními filtry, které stále častěji nahrazují analogové filtry. Rozšíření použití digitálních filtrů souvisí se zvyšováním hustoty integrace v integrovaných obvodech. Se zvyšováním složitosti systémů se zvyšuje potřeba vývoje prostředí, která by umožňovala efektivní návrh a verifikaci těchto systémů. Při návrhu zákaznických integrovaných obvodů (ASIC Application Specific Integrated Circuit – Zákaznický integrovaný obvod) je také potřeba přihlížet na požadavky kladené na vývoj systémů. Mezi hlavní požadavky většinou patří nízká výrobní cena, která je přímo úměrná ploše čipu, tj. kvalitě vygenerovaného HDL popisu celého systému.

V dnešní době je před samotnou výrobou obvodu nepostradatelnou a nejvíce časově náročnou fází procesu vývoje testování navržených systémů. Nedílnou součástí testovací fáze je verifikace. Důvodem zavedení verifikace je nalezení chyb a jejich odstranění před výrobou obvodu.

U vydaných programů lze chyby opravit aktualizací, avšak u digitálních filtrů a obecně návrhu digitálních obvodů by pak vynechání verifikace mohlo mít za následek vyrobení vadných obvodů.

Hlavním tématem práce je vytvoření verifikačního prostředí, které by umožňovalo simulaci systémů s digitálními filtry a sledování pokrytí, které určuje jak velká část systému byla otestována.

1 DIGITÁLNÍ FILTRY

Filtr v elektrotechnice obecně slouží pro změnu spektra výstupního signálu. U digitálních filtrů se jedná o změnu spektra diskrétního signálu, tzn. okamžitá hodnota se nemění spojitě v čase. Digitální filtry mohou v určitých případech nahrazovat pasivní a aktivní analogové filtry.
Porovnání digitálních a analogových filtrů:

Vlastnosti digitálních filtrů:

- vysoká přesnost,
- nemají drift,
- mohou mít lineární fázi (FIR),
- možnost adaptivní filtrace (tzv. samoučící filtrace),
- snadná simulace a návrh,
- výpočet musí proběhnout během periody vzorkování,
- filtrace nízkých frekvencí,
- nevhodné pro vf signály (omezeno výpočetní technikou).

Vlastnosti analogových filtrů:

- menší přesnost,
- drift vlivem změn součástek,
- nelineární fáze,
- nelze použít adaptivní filtraci,
- obtížná simulace a návrh,
- vhodné pro vysoké frekvence.

Pokud nahrazujeme analogový filtr digitálním (číslicovým), je třeba použít A/D převodníky a poté případně D/A převodníky. Při převodu analogového signálu na digitální musíme signál vhodně navzorkovat (signál se stává diskrétním).

Vzorkování musí splňovat Shannonův–Kotělníkův teorém o dvojnásobné vzorkovací frekvenci:

$$f_s \geq 2 \cdot f_{MAX} \tag{1.1}$$

kde f_s je vzorkovací frekvence a f_MAX je maximální frekvence obsažená ve vzorkovaném signálu.

Pokud není splněn vzorkovací teorém, při zpětné rekonstrukci diskrétního signálu na analogový může dojít k aliasingu (periodizovaný signál se překrývá).

Číslicové filtry jsou zpravidla popisovány matematickými rovnicemi. Diferenční rovnice vyjadřuje závislost mezi posloupnostmi a jejich diferencemi:

$$b_s y(n+s)+b_{s-1}y(n+s-1)+\ldots+b_0 y(n) = a_r x(n+r)+a_{r-1}x(n+r-1)+\ldots+a_0 x(n) \tag{1.2}$$

kde $b_0, b_1, \ldots, b_s$ a $a_0, a_1, \ldots, a_r$ jsou koeficienty diferenční rovnice, $x(n)$ je známá vstupní posloupnost a $y(n)$ je hledané řešení diferenční rovnice.

Provedeme–li $\mathcal{Z}$ transformaci diferenční rovnice, dostaneme přenosovou funkci:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_0 + a_1 z + a_1 z^2 + \ldots + a_r z^r}{b_0 + b_1 z + b_1 z^2 + \ldots + b_s z^s} \tag{1.3}$$

kde $Y(z)$ je obraz výstupního signálu v $\mathcal{Z}$ transformaci, $X(z)$ je obraz vstupního signálu v $\mathcal{Z}$ transformaci, $a_0, a_1, \ldots a_r$ a $b_0, b_1, \ldots b_s$ jsou koeficienty diferenční rovnice a z^r a z^s jsou operátory vyjadřující zpoždění systému.

Z přenosové funkce lze zjistit:

- nulové body a póly,
- impulsní charakteristiku,
- stabilitu filtru,
- řád filtru.

Nulové body a póly

Polynomy v čitateli $Y(z)$ a jmenovateli $X(z)$ ze vzorce 1.3 můžeme rozložit na součin kořenových činitelů:

$$H(z) = \frac{Y(z)}{X(z)} = K\frac{(z-n_1)(z-n_2)+\ldots+(z-n_M)}{(z-p_1)(z-p_2)+\ldots+(z-p_N)} \tag{1.4}$$

kde $n_{1,2,\ldots,M}$ jsou nulové body a $p_{1,2,\ldots,N}$ jsou póly přenosové funkce.

Nulové body jsou tedy řešením rovnice $Y(z) = 0$ a póly jsou řešením $X(z) = 0$. Pro grafické znázornění je možné nulové body a póly zakreslit do komplexní roviny. Podle počtu pólů se určuje řád filtru.

Impulsní charakteristika

Impulsní charakteristika je vynucená odezva systému na jednotkový impuls a vychází ze součinu obrazů diskrétní konvoluce posloupností:

$$y(n) = \sum_{m=0}^{n} x(m)h(n-m) \tag{1.5}$$

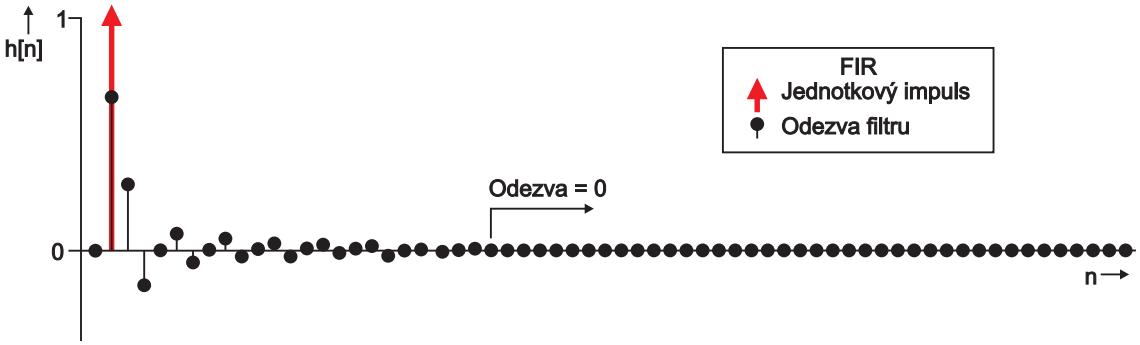
kde $y(n)$ je výstupní signál, $x(n)$ je vstupní signál a $h(n)$ je impulsní odezva (charakteristika) digitálního filtru.

Přenosová funkce $H(z)$ je obrazem impulsní charakteristiky digitálního filtru $h(n)$. Impulsní charakteristika $h(n)$ je dána jako zpětná $\mathcal{Z}$ transformace k přenosové funkci $H(z)$.

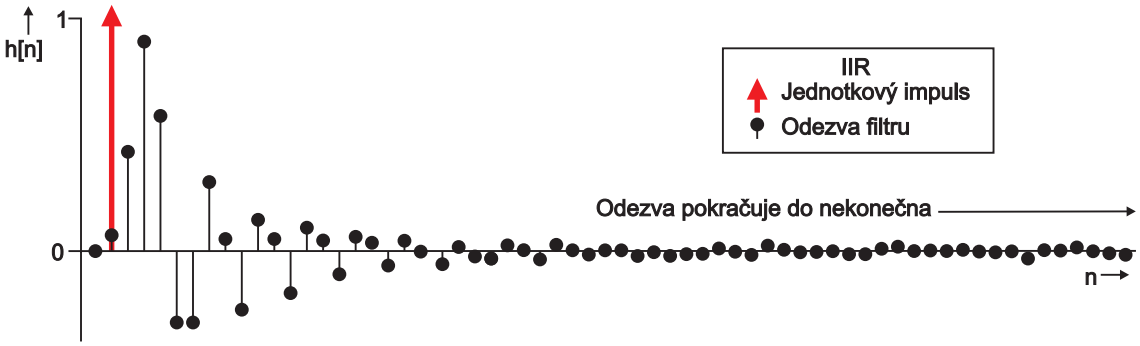
Podle délky odezvy impulsní charakteristiky rozdělujeme filtry na:

- diskrétní systémy s konečnou impulsní charakteristikou FIR (Finite Impulse Response),
- diskrétní systémy s nekonečnou impulsní charakteristikou IIR (Infinite Impulse Response).

Na obr. č. 1.1 a 1.2 jsou uvedeny charakteristiky výše zmíněných filtrů:



Obr. 1.1: Impulsní odezva FIR filtru na jednotkový impuls [1]



Obr. 1.2: Impulsní odezva IIR filtru na jednotkový impuls [1]

Stabilita filtru

Digitální filtr je stabilní, pokud všechny jeho póly leží uvnitř jednotkové kružnice v komplexní rovině $\mathcal{Z}$.

1.1 FIR filtr

Filtry s konečnou impulsní charakteristikou (FIR – Finite Impuls Response) jsou nerekurzivními filtry (nemají zpětné vazby), tzn. výstupní signál záleží pouze na hodnotě vstupního signálu.

Diferenční rovnice FIR filtru

$$y[n] = h_0 x[n] + h_1 x[n-1] + \ldots + h_{N-1} x[n-N-1] = \sum_{k=0}^{N-1} h_k x[n-k] \qquad (1.6)$$

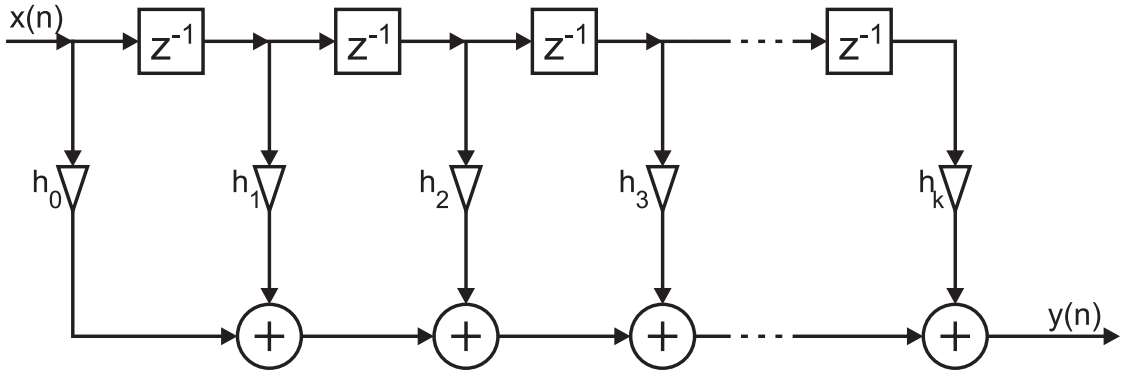
kde $y[n]$ je výstupní signál, $h_0, h_1, \ldots, h_{N-1}$ jsou hodnotami impulsní odezvy, $x[n]$ je vstupní signál, N–1 je řád filtru.

Přenosová funkce FIR filtru

$$H(z) = \sum_{k=0}^{N-1} h_k \cdot z^{-k} \qquad (1.7)$$

kde h_k jsou koeficienty filtru. Filtr má jeden N násobný pól $z = 0$, v počátku a systém je tedy vždy stabilní.

Přímá struktura FIR filtru je uvedena na obr. č. 1.3:



Obr. 1.3: Přímá struktura FIR filtru [2]

Výhody:

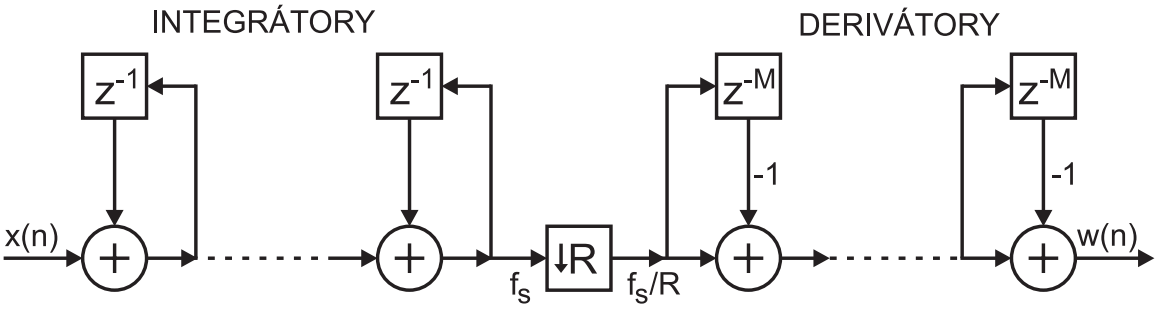
- vždy stabilní,
- matematicky jednoduše popsatelné,
- mohou mít lineární fázovou a kmitočtovou charakteristiku (vhodné pro signály s velkou šířkou pásma).

Nevýhody:

- nemají ekvivalent v analogových filtrech,
- pro dosažení velké strmosti nutno volit vysoký řád filtru,
- nevýhodou je zpoždění filtru při zpracování vstupního vzorku (velký počet zpožďovacích členů, násobiček),
- optimální iterační metody jsou výpočetně náročné.

1.2 CIC filtr

Speciálním typem FIR filtru je filtr CIC (Cascaded Integrating Comb – Hřebenový integrační filtr). Využívá decimace (snížení vzorkovací frekvence) a interpolace (zvýšení vzorkovací frekvence). Častěji se využívá decimace jako na obr. č. 1.4. Před samotným procesem decimace musí být splněn vzorkovací teorém (vzorec 1.1). Poté je snížen vzorkovací kmitočet decimačním faktorem. Decimační faktor označujeme R, z vypočítaných hodnot se vybere každý R–tý vzorek. Ostatní vzorky se neberou v úvahu. [3]



Obr. 1.4: Struktura filtru CIC – decimátor [4]

Signál x(n) vstupuje navzorkovaný frekvencí f_s, výstupní signál w(n) má vzorkovací frekvenci f_s/R.

Příklad použití CIC filtru je veden na obr. č. 1.5:

Obr. 1.5: Blokové schéma filtru s použitím CIC

CIC filtr zpracovává signály s vysokými vzorkovacími frekvencemi a snižuje je. FIR filtr lépe pracuje právě na nižších kmitočtech (má strmější kmitočtovou charakteristiku). Dále kompenzuje útlum CIC filtru.

1.3 IIR filtr

Filtry s nekonečnou impulsní charakteristikou (IIR – Infinite Impulse Response) jsou rekurzivní filtry (mají zpětné vazby). IIR filtry mají obecně větší možnosti zapojení než FIR právě kvůli rekurzivní části.

Diferenční rovnice IIR filtru:

$$y[n] = \sum_{m=0}^{M} b_m x[n-m] - \sum_{l=1}^{L} a_l y[n-l] \tag{1.8}$$

kde $y[n]$ je výstupní signál, a_l jsou koeficienty zpětných vazeb, b_m jsou koeficienty dopředné zpětné vazby, M je počet zpoždění v nerekurzivní části, L je počet zpoždění v rekurzivní části a udává řád filtru, $x[n-m]$ a $y[n-l]$ jsou vzorky vstupního a výstupního signálu zpožděné o m nebo l.

Přenosová funkce IIR filru:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + \ldots + a_L z^{-L}} \tag{1.9}$$

IIR filtry mají alespoň jeden pól mimo počátek souřadnic roviny $\mathcal{Z}$, proto může dojít k nestabilitě.

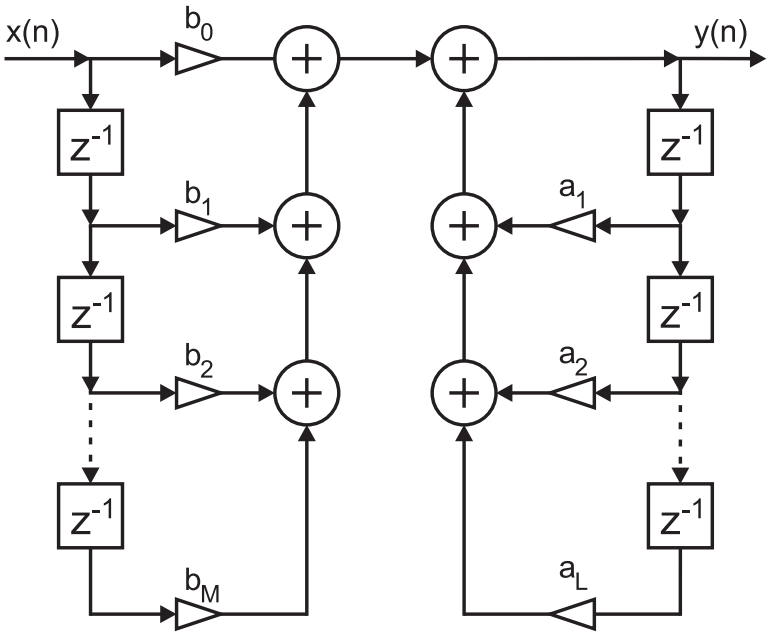
Výhody:

- malý řád přenosové funkce,
- malé zpoždění při zpracování vstupního vzorku (čím nižší řád filtru, tím méně zpožďovacích členů je nutné použít),
- mají ekvivalent v analogových filtrech.

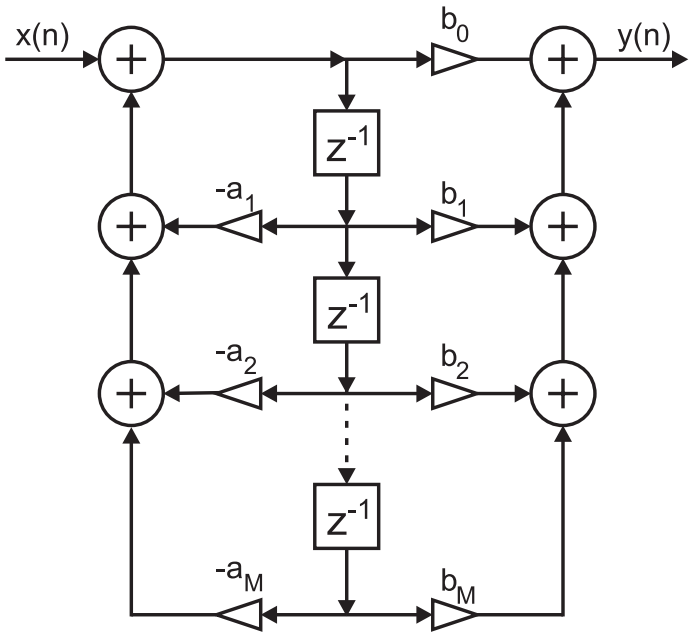
Nevýhody:

- nejsou vždy stabilní,
- nemohou mít lineární fázovou a kmitočtovou charakteristiku,
- velká citlivost na kvantování.

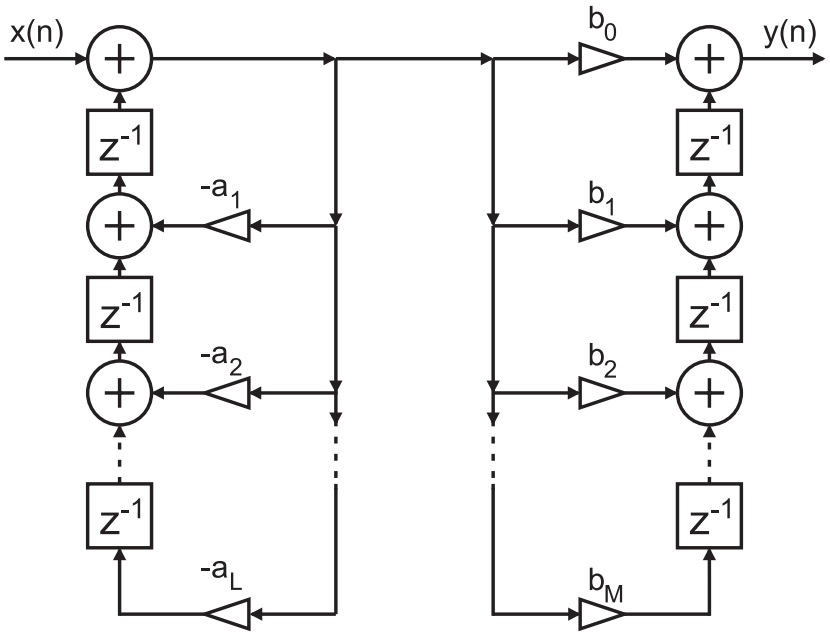
Přímé struktury IIR filtrů jsou uvedeny na obr. č. 1.6 a 1.8. Od nich odvozené (kanonické) struktury jsou uvedeny na obr. č. 1.7 a 1.9.



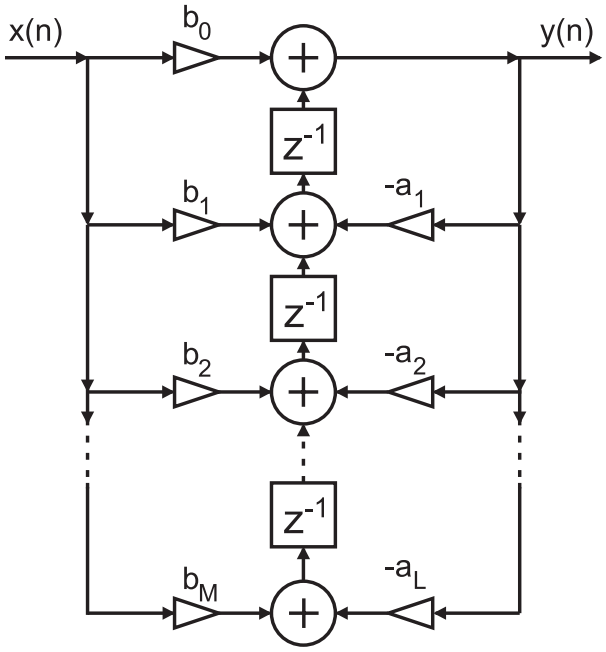
Obr. 1.6: Přímá struktura IIR filtru (Direct Form I) [2]



Obr. 1.7: Kanonická struktura IIR filtru (Direct Form II) [5]



Obr. 1.8: Přímá transponovaná struktura IIR filtru (Transposed Direct Form I) [5]

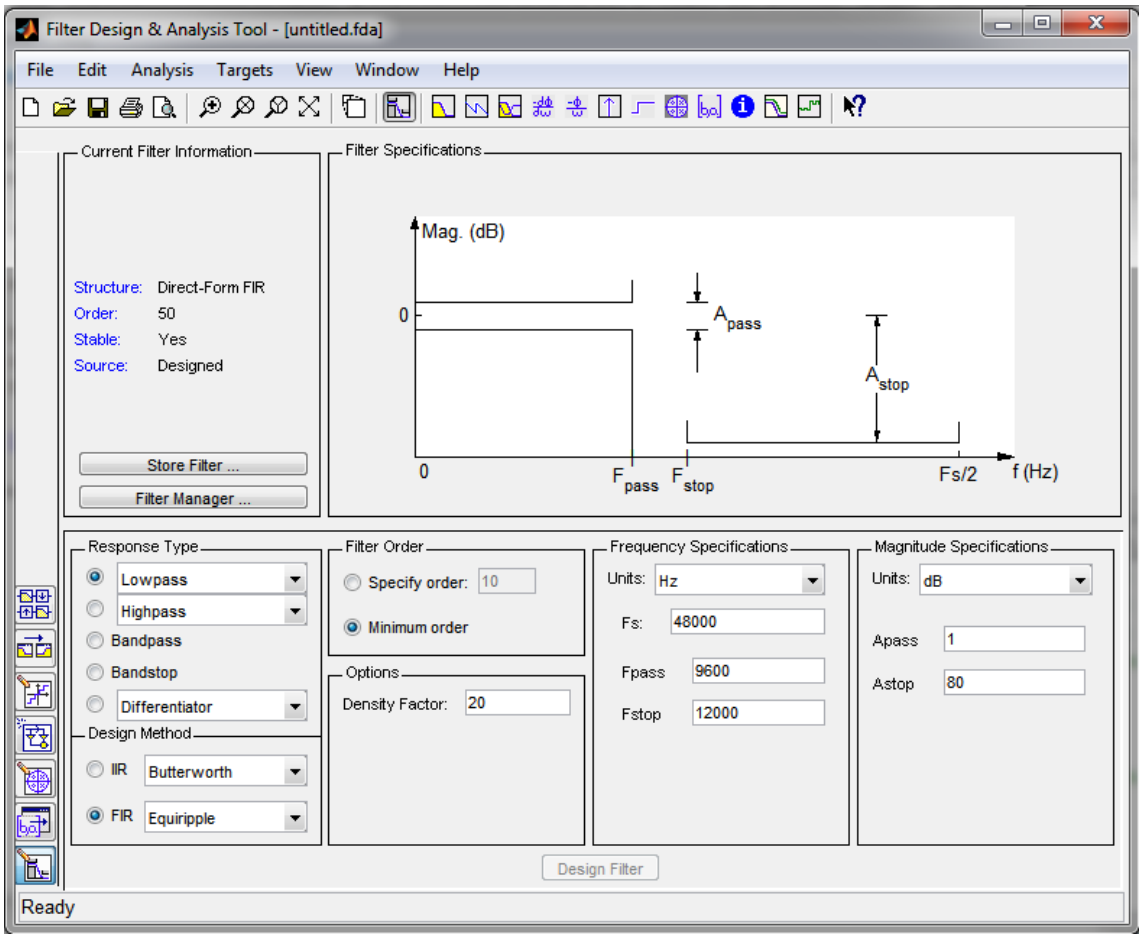


Obr. 1.9: Kanonická transponovaná struktura IIR filtru (Transposed Direct Form II) [5]

Problémy s citlivostí kvantování koeficientů a stability IIR řeší filtry SOS (Second-Order Sections – Filtry druhých řádů). Filtry vyšších řádů jsou nahrazovány kaskádou filtrů druhých řádů jedné z uvedených struktur.

1.4 Návrh filtrů

Programové prostředí Matlab nabízí mimo jiné i program Filter Design & Analysis Toolbox (obr. č. 1.10), který slouží pro vytváření digitálních filtrů. V programu Matlab se spouští příkazem „fdatool“.



Obr. 1.10: Filter Design & Analysis Toolbox

Možnosti nastavení:

- typ modulové frekvenční charakteristiky (dolní propust, horní propust, pásmová propust, pásmová zádrž, apod.),
- druh filtru z hlediska použité metody – FIR nebo IIR a k nim příslušné aproximace (např. Butterworth, Chebyshev),
- frekvence – Fs – vzorkovací frekvence, Fpass – mezní frekvence, Fstop – frekvence potlačení,
- útlum – Apass – zvlnění v propustném pásmu, Astop – zvlnění v závěrném pásmu,
- řád filtru – možnost nastavit manuálně nebo nejmenší možný řád filtru,
- možnost nastavení kaskády filtrů,

- kvantovací parametry:
  - výstup koeficientů – fixed point (nastavení pevné desetinné čárky), double–precision floating point, single–precision floating point (formáty s plovoucí desetinnou čárkou),
  - šířka a rozsah koeficientů, vstupních a výstupních dat,
  - možnost nastavení kaskády filtrů,
  - kvantování vstupů, výstupu a koeficientů (může vést k nestabilitě IIR filtrů).

Datový typ s pevnou desetinnou čárkou (fixed point) umožňuje explicitní určení polohy desetinné čárky, tzn. nastavení přesnosti prováděných výpočtů.

Program má mnoho užitečných funkcí:

- zobrazení modulové i fázové frekvenční charakteristiky, skupinového a fázového zpoždění, impulsní a skokové odezvy,
- zobrazení nulových bodů, pólů a koeficientů filtru,
- možnost přidávat nulové body a póly,
- transformaci filtru na jiný typ (např. dolní propust $\rightarrow$ horní propust),
- možnost realizace filtru jako bloku do programu Simulink (další z programů Matlabu),
- export filtru do dalších podpůrných programů Matlab,
- generování filtru ve VHDL (Very High Speed Integrated Circuits Hardware Description Language – Jazyk pro popis velmi rychlých integrovaných obvodů) nebo v hlavičkovém souboru v jazyce C,
- výpočet koeficientů filtrů a jejich uložení v desítkové, binární i hexadecimální soustavě,
- umožňuje faktorizovat IIR filtry vysokých řádů na kaskádu IIR filtrů druhého řádu. Důvodem je nestabilita IIR vysokého řádu.

Vytvořený filtr může být vyexportován do souboru s příponou „*.fcf“. V souboru jsou vypočteny koeficienty přenosové funkce ve zvolené číselné soustavě. Dále také informace o typu filtru, bitové šířce koeficientů, rozsahu vstupních a výstupních dat, stabilitě filtru, atd.

2 VERIFIKACE

Slovem verifikace označujeme v oblasti techniky proces ověření správnosti určitých dat. Správností dat je myšlena správná forma i platnost dat. Např. verifikace telefonního čísla ověří, zda–li má číslo správný počet cifer, funkčnost čísla nebo také zda číslo patří danému člověku.

Proces vývoje programu může být rozdělen do těchto fází:

1. specifikace,
2. návrh,
3. implementace,
4. testování systému,
5. provoz a údržba.

Do fáze testování obvodu můžeme spolu s verifikací zahrnout i proces validace a testování. Validace nám dává odpověď na otázku, zda obvod odpovídá reálným požadavkům zákazníka. Testování je ověřování obvodu pomocí konečné sady příkladů. Verifikace zjišťuje, zda vytvořený obvod odpovídá výchozí specifikaci, tzn. srovnává specifikaci a implementaci. Specifikace je obvykle ve formě dokumentace, kde je popsáno veškeré chování programu jako jsou rozhraní (vstupní, výstupní, sběrnice), funkce obvodu na vyšší úrovni abstrakce (aritmetické operace, zpracování instrukcí), formát vstupů a výstupů (formát paketu, paměťové operace), časování, frekvence obvodu a požadavky na paměť.

Důvodem zavedení verifikace je nalezení chyb a jejich odstranění před samotnou výrobou obvodu. Zavedení verifikace do procesu vývoje obvodu se může někdy zdát jako neekonomický krok, avšak v případě odhalení chyby v obvodu po zahájení výroby může mít za následek výměnu všech chybných obvodů. Díky verifikaci také roste důvěra zákazníků a prestiž vývojáře, protože je obvod bezchybný a bezpečný.

Pokud daný test nenalezne žádné chyby, nemůžeme říct, že je obvod v pořádku. Pouze, že test neodhalil chybu.

Jelikož je proces verifikace časově náročný, provádí se verifikace automatizovaná s podporou simulačních nástrojů. Výhodou je zrychlení procesu, částečná eliminace lidského faktoru, snadněji opakovatelná procedura a možnost analyzovat komplexnější systémy.

Pro dosažení spolehlivých obvodů je nutné dodržovat programátorské techniky jako jsou např. ošetření všech výjimek a všech stavů a stejnorodost zdrojových kódů.

2.1 Druhy verifikací

Verifikace může být členěna na tyto skupiny:

- verifikace simulací (program ModelSim, QuestaSim) – verifikace v simulačním prostředí (testbench), který vytváří stimuly a kontroluje odezvy,
- formální verifikace (Formality, Conformal) – formální důkaz funkčních vlastností obvodu,
- statická časová analýza (STA) (PrimeTime) – automatická kontrola časových vlastností obvodu.

Dalším přístupem je rozdělení verifikace na:

- fyzickou,
- formální,
- funkční.

Fyzická verifikace je ověření správného návrhu na úrovní layoutu. Formální verifikace využívá matematických metod k formálnímu popisu systému a jeho vlastností a ověřuje, zda je funkčnost systému v souladu se specifikací. Nevyžaduje generování stimulů. Funkční verifikace ověřuje, zda model obvodu plní specifikaci sledováním jeho vstupů a výstupů v simulaci. Verifikace probíhá v simulačním prostředí, které si může samo vytvářet vstupy (stimuly) a kontroluje výstupy (odezvy). Funkční verifikace se snaží simulovat skutečnou činnost obvodu, proto jsou vstupní data pouze takové kombinace, které mohou reálně nastat.

2.2 Pokrytí

Jedním ze sledovaných parametrů je pokrytí (coverage), které udává, jak velká část obvodu byla testována. Pro testovaný obvod se zavádí pojem DUT (Design/device Under Test – Testovaný systém/zařízení) a pojem „bod pokrytí“ (coverpoint). Každý bod pokrytí musí být otestován alespoň jednou. Ukončení verifikace může být dáno pokrytím, kdy je určena hranice dostatečného pokrytí. Jakmile je tato hranice překročena, je možné považovat systém za verifikovaný.

K dosažení pokrytí je nutné generovat vstupní data pro DUT. Na základě různých přístupů generování vstupních dat rozdělujeme testy na:

- přímé testy (Direct stimuli test),
- testy s náhodnými daty (Constraint–random stimuli test),
- verifikace řízená pokrytím (Coverage–driven verification, Metric–driven Verification).

U přímého testu jsou testovací data vytvořena verifikačním inženýrem. Výhodou je rychlé vytvoření testu a možnost určení, jaká funkce bude otestována. Nevýhodou je, že se testuje chování obvodu, které verifikační inženýr očekává. Verifikační plán je třeba upravit při každé změně specifikace.

Testy s náhodnými daty využívají náhodného generování vstupních dat. Výhodou je pokrytí i nepředpokládaných chyb a rychlejšího dosažení 100 % pokrytí. Nevýhodou je nutnost věnovat delší čas vývoji verifikačního prostředí.

Verifikace řízená pokrytím měří pokrytí během celého procesu verifikace a bere v úvahu, které vlastnosti systému byly během verifikace řádně prověřeny, a na základě toho řídí generování vhodných vstupních dat. V dnešní době je tento typ verifikace nejvíce používaný.

Dále rozlišujeme tyto typy pokrytí:

- Funkční pokrytí (Functional coverage) – pokrytí funkcí a chování systému. V případě funkčního pokrytí je sledováno, zda byly otestovány body pokrytí, které jsou navrženy ve verifikačním plánu.
- Pokrytí kódu (Code coverage) – provedení všech různých kódových konstrukcí,
  - Pokrytí výrazů (Statement coverage) – každý výraz musí být proveden alespoň jednou.
  - Pokrytí větví (Branch coverage) – podmínka každého větvení musí být alespoň jednou pravdivá a jednou nepravdivá.
  - Pokrytí podmínek (Condition coverage) – zkontroluje všechny možné způsoby, za kterých je podmínka pravdivá či nepravdivá.
  - Pokrytí přechodů (Toggle Coverage) – každý bod pokrytí musí projít z log. 0 do log. 1 a naopak (případně navíc kombinace stavů „Z“ a „X“).
  - Pokrytí cest (Path coverage) – provedené cesty v systému, vyžaduje provedení všech různých cest. V praxi je neproveditelné.
  - FSM pokrytí (FSM coverage) – navštívené stavy stavových automatů (FSM – Finite State Machine – Konečný statový automat) a kombinace všech přechodů mezi stavy (často nelze pokrýt všechny přechody z podstaty stavového automatu).

2.3 Verifikační plán

Verifikační plán vychází ze specifikací na začátku vývojového procesu. Obsahuje informace o tom, co (body pokrytí) a jakým způsobem (přímé testy, verifikační prostředí) bude systém verifikován. Ideální verifikační plán počítá s otestováním všech možných kombinací. Verifikace u větších systémů by byla však velice časově náročná, proto se používá verifikační plán, který otestuje pouze významné (hraniční) hodnoty pro každý parametr.

2.4 SystemVerilog

SystemVerilog je objektově orientovaný jazyk určený speciálně pro tvorbu verifikačních prostředí (HVL – Hardware Verification Language – Jazyk pro verifikaci digitalních obvodů) i implementaci hardwarových systémů. Je založen na prvcích jazyka Verilog, ale zahrnuje v sobě výhody různých programovacích jazyků s důrazem na simulaci a verifikaci. Vznikl spojením jazyka pro popis digitálních obvodů (Hardware Description Language, HDL – Verilog, VHDL) a jazyka pro verifikaci digitálních obvodů (HVL – Vera). Standardy jazyků Verilog a SystemVerilog byly v roce 2009 sloučeny do jednoho společného standardu IEEE Std. 1800–2009.

Samotný jazyk SystemVerilog je možné rozdělit na dvě podmnožiny:

- RTL (Register-Transfer Level – Metodika návrhu digitálních obvodů),
- verifikace.

Část jazyka určená pro RTL vychází z Verilogu ve verzi 2005. Navíc přináší mnoho nových datových typů (dvoustavové i čtyřstavové), nové typy procedurálních bloků, rozšířené operátory, nové výrazy a mnoho dalších rozšíření.

Verifikační podmnožina jazyka je založená na principech objektově orientovaného programování. SystemVerilog přináší především objektové třídy, podporu pro funkční pokrytí a princip generování rozsahem omezených náhodných čísel, vektorů a řad (constrained random generation). Ten se uplatní při získávání testovacích hodnot. Omezení hodnot je nutné, aby byly generovány pouze přípustné hodnoty. Při správném nastavení tak lze automaticky ověřit všechny přípustné kombinace vstupních hodnot. Dále jazyk umožňuje volat funkce, které jsou napsané v jiných programovacích jazycích, např. C/C++.

Zápis funkcí je velmi blízký jazyku C. Rozlišujeme dva typy:

- *task* – může obsahovat časové řízené příkazy a volat další *task* a *function*,
- *function* – se provede za nulový čas (tzn. nemůže být použito příkazů pro časové prodlevy nebo čekání). [9]

Každá komponenta verifikačního prostředí pracující s časem musí mít nastaveno rozlišení a jednotku času, se kterými pracuje.

V simulačních prostředích může ověřovat verifikační prostředí napsané v jazyce SystemVerilog funkčnost souborů napsaných v jazyku jiném (VHDL, Verilog).

2.5 Verifikační metodiky

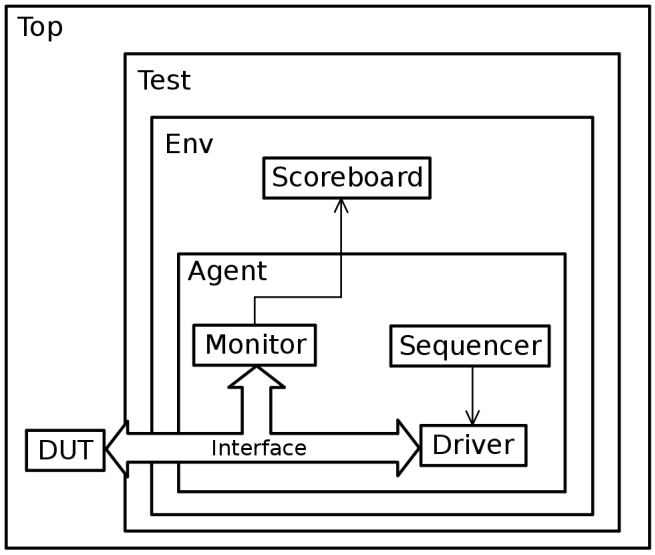
Verifikační metodiky jsou standardy, které specifikují, jak vytvářet znovupoužitelné a snadno rozšířitelné verifikační prostředí v jazyce SystemVerilog. Následující názvy komponent vycházejí z dané metodiky a pro zachování jednoznačnosti budou používány v anglickém jazyce. Umožňují použít předpřipravené a volně přístupné komponenty verifikačních prostředí definované v podobě knihoven. Používá se tzv. vrstvený testbench (layered testbench), který umožňuje snadnější úpravu při změně specifikace, snadnější opravu možných chyb a použití stejných komponent v různých projektech.

Nejznámější metodiky jsou:

- VMM – Verification Methodology Manual – Verifikační metodika od společnosti ARM a Synopsys,
- OVM – Open Verification Methodology – Verifikační metodika od společnosti Cadence a Mentor Graphics,
- UVM – Universal Verification Methodology – Verifikační metodika od společnosti Accelera. [10]

UVM je metodika, která vychází z úspěšných VMM i OVM metodik. Je zpětně kompatibilní s OVM. Jedná se o metodiku založenou na testech s náhodně generovanými daty a testech řízených pokrytím.

Na následujícím obrázku (2.1) je zobrazeno jednoduché blokové schéma podle metodiky UVM.



Obr. 2.1: Typické blokové schéma podle UVM

Agent

Agent obsahuje skupinu UVM komponent, kterými jsou *Sequencer*, *Driver* a *Monitor*. *Sequencer* a *Driver* vytváří vstupní data (stimulus) pro DUT. *Sequencer* na úrovni funkcí a příkazů, *driver* na úrovni bitové.

Kontrola výstupních dat může být pro malé obvody ruční, pro větší systémy je nutná kontrola automatická. Pro automatickou kontrolu se ve verifikačních prostředích používá tzv. *Monitor*, který čte hodnoty na pinech DUT. Přijatá data předává do *Scoreboard* pro porovnání s referenčními daty.

Interface

Někdy také nazývaná jako signálová úroveň. Data z a do DUT jsou posílána na bitové úrovni.

Environment

Instance *Environment* obsahuje blok *Scoreboard* a jeden nebo více bloků *Agent* a *Monitor* pro měření pokrytí (někdy nazývaný jako *Subscriber*). *Scoreboard* srovnává přijatá data z komponenty *Monitor* s referenčními daty, která jsou často výstupy referenčního modelu. Referenční model slouží k tomu, aby bylo možné výstupní data z DUT porovnávat s očekávanými a správnými daty. Bývá popsán v programovacím jazyku C++, SystemVerilog nebo Matlab. *Environment* slouží k propojení bloků *Scoreboard* a *Agent*.

Test

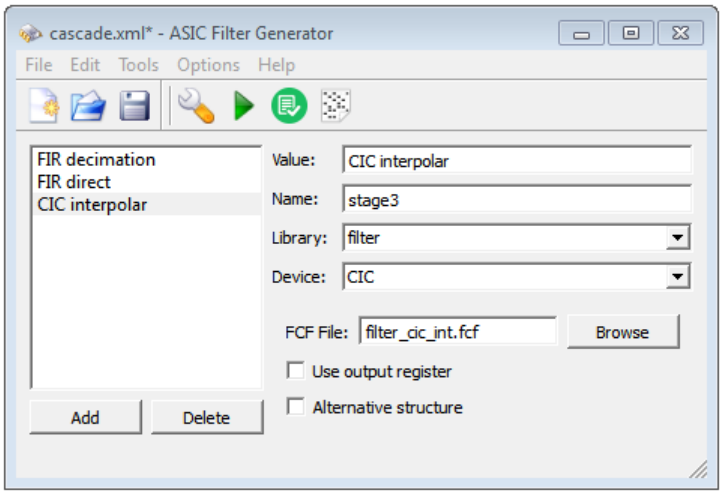
Obsahuje samotné testy (testbench). Každý *Test* obsahuje instanci *Environment* a její nastavení. Klade se důraz, aby nebylo nutné zasahovat při různých testech do instance *Environment*, ale pouze do jejího nastavení.

Top

Komponenta *Top* slouží ke spouštění jednotlivých testů a slouží jako spojení mezi DUT a testy. [11] [12]

3 FUNKCE PRO VERIFIKACI

Cílem této práce je navázat na program ASIC Filter Generator (obr. č. 3.1), který umožňuje generovat popis kaskádního zapojení filtrů v jazyce VHDL, a doplnit jej o možnost verifikovat vytvořený VHDL popis filtrů. Program je napsán v jazyce C++ v prostředí Qt4 Framework, a proto je rozšiřují funkce pro verifikaci taktéž napsána v tomto prostředí. Verifikační prostředí je napsáno v jazyce SystemVerilog a samotné prostředí je spouštěno v programu ModelSim nebo QuestaSim. Vytvořené prostředí umožňuje analýzu filtrů a sledování pokrytí testů. Dále je využito programového prostředí Matlab pro vytváření souborů s popisem filtrů a pro získání referenčních dat pro verifikaci.



Obr. 3.1: Grafická podoba programu

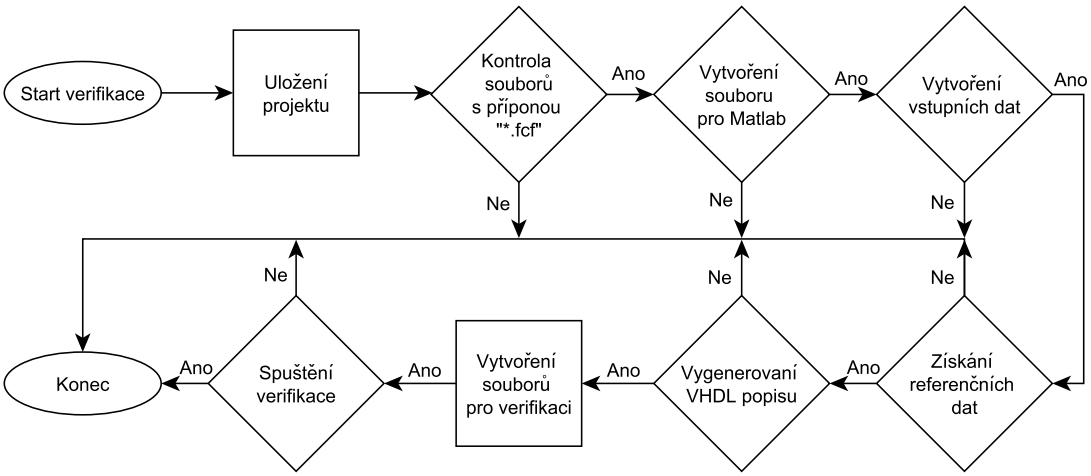
Program umožňuje:

- vkládat FIR a CIC filtry (IIR filtry nejsou podporovány),
- měnit názvy filtrů, které jsou použity ve VHDL popisu a tím pádem i ve verifikačním prostředí,
- nastavovat jednotlivým filtrům parametry (pro CIC jsou to použití výstupního registru a alternativní struktury, pro FIR filtry jsou to použití výstupních registrů nebo paměti a možnost vynechat reset datové paměti), tyto parametry mění vlastnosti filtrů a je nutné s nimi při generování verifikačního prostředí počítat.
- nastavovat typ hodinového signálu (na náběžnou nebo sestupnou hranu) a typ resetu (asynchronní/synchronní a aktivní na log. 0 nebo log. 1),
- možnost generovat popis v jazyce Verilog nebo VHDL,
- další uživatelské funkce (ukládání, načítání a další). [7]

3.1 Hlavní smyčka funkce pro verifikaci

Funkce *verify()* (na obr. č. 3.2) obstarává celý proces verifikace a její průběh je následující:

1. uložení – kontrola zda nebyla provedena změna a případné uložení designu.
2. Kontrola souborů s příponou „*.fcf“ – volání funkce *checkFcfFiles()*.
3. Vytvoření souboru pro Matlab – volání funkce *createMFile()*.
4. Vytvoření vstupních dat – volání funkce *createWave()*.
5. Získání referenčních dat – volání funkce *startMatlab()*.
6. Vygenerování VHDL popisu – volání funkce *generate()*. Právě tento vygenerovaný VHDL popis bude verifikován. Tato funkce není předmětem diplomové práce.
7. Vytvoření souborů pro verifikaci – volání funkce *createFileForVerification()*.
8. Spuštění verifikace – volání funkce *startQuesta()*.



Obr. 3.2: Vývojový diagram funkce pro verifikaci

Jak je z vývojového diagramu patrné, pokud některá z funkcí neproběhne v pořádku, může být celá funkce *verify()* ukončena. Jednotlivé funkce budou popsány v následujících kapitolách. Umístění jednotlivých funkcí ve zdrojových souborech je v tabulce č. 3.1:

3.2 Kontrola souborů popisující filtry

Kontrola souborů probíhá pomocí funkce *checkFcfFiles()*. Je kontrolováno, zda mají všechny filtry přiřazený soubor s příponou „*.fcf“. Soubory mohou být přesunuty,

Tab. 3.1: Výčet zdrojových souborů

Název souboru	Funkce
mainwindow_mfile.cpp	vytvoření souboru pro Matlab
mainwindow_verify.cpp	uložení, kontrola souborů s příponou „*.fcf“, vytvoření vstupních dat, získání referenčních dat, spuštění verifikace,
mainwindow_verify_file.cpp	vytvoření souborů pro verifikaci.

a proto se dále kontroluje, zda se vůbec soubor existuje právě na uložené cestě. Jestliže je nalezena jedna ze zmíněných chyb, program nahlásí chybu.

3.3 Vytvoření souboru pro Matlab

V kapitole 1.4 je uvedena možnost generování filtrů a následné ukládání do souboru s příponou „*.fcf“ z programu Filter Design & Analysis Toolbox (dále FDA Tool), který je součástí programového prostředí Matlab. Takto uložený soubor však nelze zpětně otevřít v programu FDA Tool a z tohoto důvodu nelze vytvořit referenční model přímo v Matlabu pouhým otevřením souboru s příponou „*.fcf“. Aby bylo možné získat referenční model a tím pádem i referenční data, je nutné převést informace ze souboru „*.fcf“ do souboru, který lze v programu Matlab spustit (s příponou „*.m“). Tento převod obstará funkce *createMFile()*. Pokud funkce nenalezne v souboru s příponou „*.fcf“ jakýkoli parametr nutný pro vytvoření souboru s příponou „*.m“, program nahlásí chybu. Příklad souboru je v příloze č. P.3.

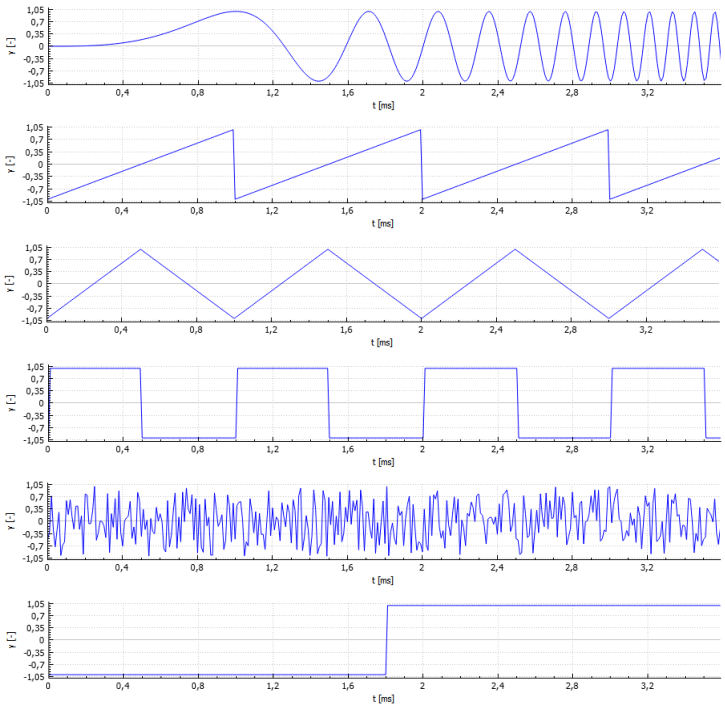
Vytvořený soubor obsahuje:

- specifikace jednotlivých filtrů – decimační nebo interpolační faktor, zpoždění, délku filtru a počet sekcí.
- Vytvoření filtru v programovém prostředí Matlab.
- Nastavení šířky vstupních a výstupních dat, akumulátoru, koeficientů filtru, režim zaokrouhlování a přetečení.
- Je–li filtrů více, jsou předchozí tři body zopakovány.
- Vytvoření finální podoby – kaskády filtrů. Jedná se tedy o referenční model.
- Načtení vstupních dat do datového typu s pevnou desetinnou čárkou (fixed–point), kterému je nastavena pevná délka desetinné i celé části. Tím je docíleno přesnosti výpočtu.

- Vstupní data jsou předána do prvního filtru. Výstupní data z filtru jsou uložena do souboru v hexadecimálním tvaru a jsou předána druhému filtru atd.
- Vytvoření VHDL popisu filtrů z Matlabu. Tento zdrojový kód obsahuje velké množství sčítaček a násobiček a je nevhodný pro použití v obvodech ASIC.

3.4 Vytvoření vstupních dat

Data pro vstup filtru jsou vytvořena pomocí funkce *createWave()*. Nejprve je vytvořen adresář „verification“, do kterého bude soubor se vstupními daty uložen. Poté je vytvořen soubor „data_in“. Do souboru jsou postupně ukládány generované průběhy jako jsou rozmítaný sinusový, pilovitý, trojúhelníkový a obdélníkový průběh. Dále je generován bílý šum a jednotkový skok. Signály jsou na obrázku č. 3.3. Pokud vytvoření souboru „data_in“ selže, program nahlásí chybu.



Obr. 3.3: Generované referenční signály

Data jsou generována v záporných i kladných hodnotách a jejich rozsah je určen vstupní bitovou šířkou prvního filtru. Pokud máme například 10 bitový vstup, tak

je postup následující:

$$\begin{aligned} rozsah\ hodnot &= 2^{10} &&= 1024 \\ maxinum &= +2^{10-1} - 1 &&= +511 \\ mininum &= -2^{10-1} &&= -512 \end{aligned} \tag{3.1}$$

Výsledný rozsah vzorků je pak v intervalu <–512; 512). Vzorky průběhů jsou převedeny do hexadecimálního tvaru. Záporné vzorky jsou převedeny ve dvojkovém doplňku. Dále je kontrolován správný počet cifer hexadecimálních čísel. Každý vzorek je ukládán na nový řádek.

3.5 Získání referenčních dat

Referenční data jsou získána voláním funkce *startMatlab()*. V tabulce č. 3.2 jsou popsány nejdůležitější funkce použité v programovém prostředí Matlab nutné k vytvoření referenčního modelu.

Tab. 3.2: Použité funkce v programu Matlab

Název funkce	Příkaz
Vytvoření datového typu Fixed–Point	*fi(v, s, w, f)*
Vytvoření FIR filtru	*dfilt.dffir(num)*
Vytvoření decimačního FIR filtru	*mfilt.firdecim(decf, num)*
Vytvoření interpolačního FIR filtru	*mfilt.firinterp(intf, num)*
Vytvoření decimačního CIC filtru	*mfilt.cicdecim(decf, numsecs, diffd)*
Vytvoření interpolačního CIC filtru	*mfilt.cicinterp(intf, diffd, numsecs)*
Vytvoření kaskády filtrů	*mfilt.cascade(Hd_1)*
Přidání filtru do kaskády	*addstage(Hd, Hd_2)*
Zpracování dat filtrem	*filter(Hd.Stage(1), input_data)*

Kde *v* jsou vstupní data, *s* je nastavení znaménkovosti, *w* je bitová šířka celého čísla, *f* je bitová šířka desetinné části, *decf* je decimační faktor, *intf* je interpolační faktor, *num* jsou koeficienty filtru, *numsecs* je počet sekcí filtru, *diffd* je diferenciální zpoždění, *Hd, Hd_1, Hd_2* jsou objekty s filtry, *Hd.Stage(1)* je první filtr v kaskádě filtrů *Hd* a *input_data* je vektor vstupních dat v datovém typu fixed–point. [13]

Datový typ Fixed–Point je používán pro vkládání koeficientů filtrů a pro vstupní i výstupní data.

Spuštění programového prostředí Matlab probíhá příkazem „matlab“ s použitím dalších parametrů. Celý příkaz, který lze použít také v příkazové řádce, je zde:

```
matlab -nodisplay -nosplash -nodesktop
-r "run('C:/Example/get_data_ref.m');"
```

Kde jednotlivé parametry znamenají:

- **-nodisplay** – nejsou zobrazovány žádné příkazy,
- **-nosplash** – není zobrazována úvodní obrazovka při spuštění,
- **-nodesktop** – program Matlabu je spuštěn v příkazové řádce,
- **-r "run('C:/Example/get_data_ref.m');"** – je proveden příkaz v uvozovkách. V tomto případě spuštění souboru.

Spouštění Matlabu a výpočet referenčních dat může trvat několik minut, a proto mohou být program Matlab i program ASIC Filter generator ve stavu „Neodpovídá".

Pro každý filtr v kaskádě je uložen soubor s výstupními referenčními vzorky do adresáře „verification" s názvem „data_out_ref_X", kde „X" je pořadí filtru. Dále jsou do pracovního adresáře uloženy „*.fcf" soubory jednotlivých filtrů. Pokud není program Matlab nainstalován, je nahlášena chyba a funkce je ukončena.

3.6 Vytvoření souborů pro verifikaci

V tabulce č. 3.3 je výčet souborů, které jsou nutné k vytvoření verifikačního prostředí.

Tab. 3.3: Seznam souborů pro verifikaci

Název souboru	Funkce
all.do	nastavení a spuštění simulace
driver.sv	komponenta pro předávání vzorků do DUT
monitor.sv	komponenta pro sledování výstupních hodnot DUT
run	dávkový soubor pro nastavení adresáře a spuštění souboru all.do
scoreboard.sv	komponenta kontrolující funkční pokrytí designu
testcase.sv	řízení celé verifikace
top_tb.sv	propojení komponent s DUT (hlavní entita VHDL popisu má název „top")
wave.do	nastavení zobrazovaných signálů v okně simulace

Všechny tyto soubory jsou ukládány do pracovního adresáře do složky „verification". Vytváření souborů pro verifikaci vykonává funkce *createFileForVerification()*, která si volá jednotlivé funkce pro sestavení každého souboru. Při vytváření souborů je nutné zohledňovat názvy souborů, komponent,

signálů a jejich bitovou šířku. Dále aktivní hranu hodinového signálu a typ resetu. Jednotlivé funkce souborů a komponent jsou popsány v kapitole č. 4.2.

3.7 Spuštění verifikace

Spuštění verifikace obstarává funkce *startQuesta()*. Je vykonán následující příkaz, který lze opět použít v příkazové řádce:

```
vsim -do C:/Example/verification/run
```

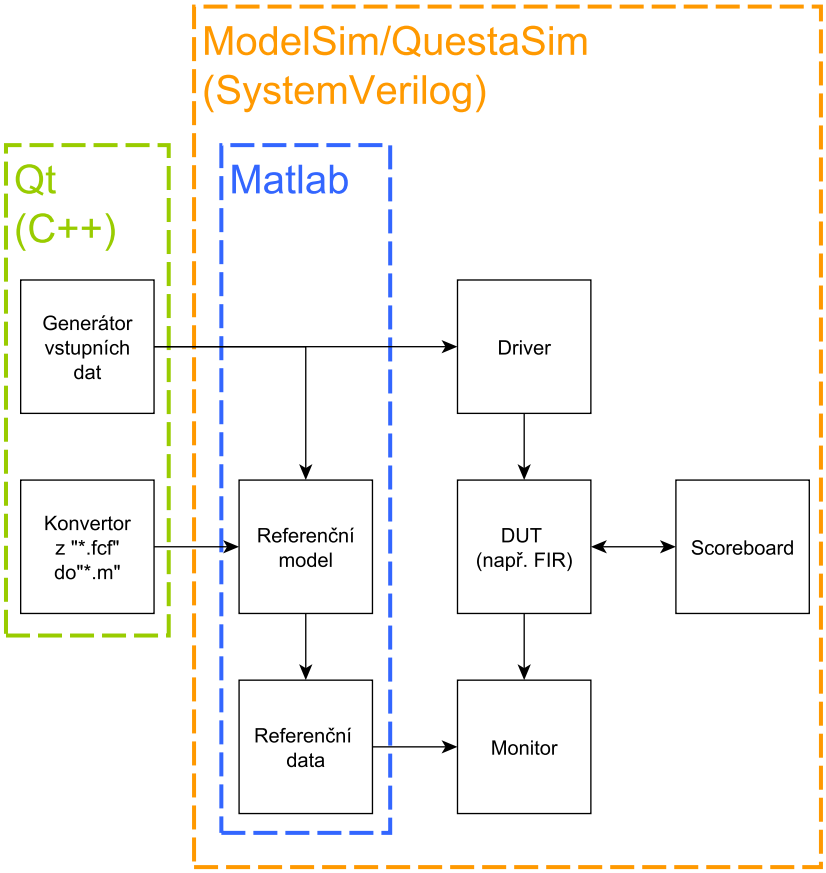
Program QuestaSim je spuštěn příkazem „vsim“ s parametrem „-do“ a cestou k souboru „run“, tzn. jsou vykonány příkazy uložené v tomto souboru. Pokud není program QuestaSim nebo ModelSim nainstalován, program nahlásí chybu.

4 VERIFIKAČNÍ PROSTŘEDÍ

Prostředí pro verifikaci filtrů vychází z verifikační metodiky UVM. Návrh blokového schématu včetně podpůrných funkcí je na obr. č. 4.1. V blokovém schématu jsou znázorněny oblasti, ve kterých programech a programovacích jazycích jednotlivé části pracují. Pro vytvoření hlavní funkce *verify()* byl použit jazyka C++ a programového prostředí Qt4 Framowork verze 4.8.4. Pro získávání referenčních dat byl použit Matlab R2010a a pro verifikaci byl použit program QuestaSim–64 10.3d. Všechny zdrojové soubory pro Matlab i QuestaSim však byly sestaveny v jazyce C++.

Jak již bylo popsáno, nejprve se vytvoří soubor pro Matlab, poté jsou vygenerována vstupní data. Následně je spuštěn Matlab a vytvořen referenční model, kterému jsou předána vstupní data. Výstupem referenčního modelu jsou tedy referenční data. Nakonec je vytvořeno verifikační prostředí.

Podrobnější popis jednotlivých komponent verifikačního prostředí je v kapitole č. 4.2. Bloky pro C++ a Matlab byly rozebrány v kapitole č. 3.



Obr. 4.1: Blokové schéma s podpůrnými funkcemi

4.1 Podpůrné soubory

Na spuštění a kompilaci verifikačního prostředí se podílí soubory „run“, „all.do“ a „wave.do“.

4.1.1 Soubor *run*

V souboru se nachází dva jednoduché příkazy, kterou jsou provedeny po spuštění programu QuestaSim. Prvním příkazem je nastavení adresáře, ze kterého bude simulace spuštěna. Tím je složka „verification“ v pracovním adresáři celého designu. Druhým příkazem je spuštění skriptu „all.do“, který je popsán v kapitole 4.1.2.

```
cd C:/Examples/cascade_CIC_dec_dec/verification
do all.do
```

4.1.2 Soubor *all.do*

Soubor slouží k nastavení a spuštění verifikačního prostředí. Příklad souboru je v příloze č. P.5. Soubor postupně vykoná tyto příkazy:

1. Pokud není QuestaSim v příkazovém režimu, je vymazáno okno „Transcript“, které slouží k zadávání příkazů a zachová historii příkazů.
2. Ukončí se předchozí simulace.
3. Vynucení ukončení simulátoru, pokud je zjištěna chyba.
4. Pokud již existuje knihovna „work“, je vymazána.
5. Vytvoření knihovny „work“.
6. Kompilace zdrojových souborů ve VHDL nebo ve Verilogu a jejich přidání do knihovny „work“. Dále jsou přidány parametry pro tzv. „code coverage“ – pokrytí zdrojového kódu.
7. Kompilace souborů verifikačního prostředí a jejich přidání do knihovny „work“.
8. Nastavení parametrů simulace – časové rozlišení je nastaveno na 1 ps a nastavení sledování pokrytí.
9. Vypnutí zobrazování varování u datového typu NumericStd.
10. Přidání souboru „wave.do“.
11. Spuštění simulace.
12. Uložení provedených příkazů z okna „Transcript“ včetně výpisu chyb při verifikaci.
13. Uložení funkčního pokrytí a pokrytí zdrojového kódu do jednotlivých souborů.
14. Ukončení simulace.

Kompilované názvy souborů v jazyce VHDL (nebo Verilog) jsou určeny podle názvů z generátoru. Pro hlavní soubor, ve kterém jsou instance jednotlivých filtru, je název odvozen od názvu „Top module“ v hlavním nastavení designu. Stejně je pojmenovaná i hlavní entita, tedy verifikovaný DUT. Pro každý filtr CIC je generován jeden soubor s názvem, jaký byl uveden v názvu filtru v programu. Pro každý filtr FIR jsou generovány tři soubory: jednotka pro násobení a akumulaci, kontrolér filtru a samotný filtr.

4.1.3 Soubor *wave.do*

V tomto souboru jsou přidány všechny signály, které chceme zobrazit v okně simulace. Ve funkci je nutné nastavit správné názvy signálů, které budou taktéž používány v dalších souborech.

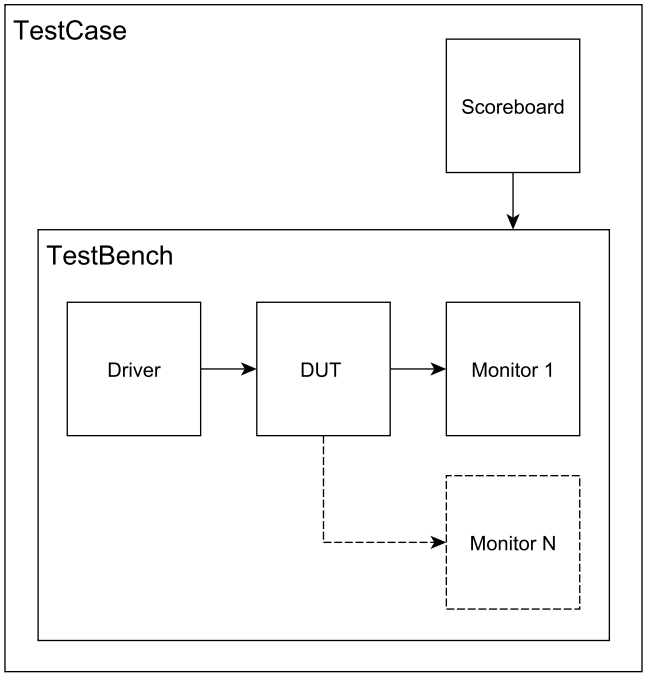
Krátký příklad souboru je zde:

```
add wave -noupdate -format Logic /clk
add wave -noupdate -divider {stage1}
add wave -noupdate -format Literal -radix hexadecimal /tb/input_data
add wave -r /tb/*
```

První řádek je přidán signál „clk“ datového typu Logic. Druhým řádkem je přidán oddělovač v okně simulace s názvem „stage1“. Třetím řádek je přidán signál „input_data“ v hexadecimálním tvaru. Signál se nachází v komponentě s názvem „tb“. Čtvrtým řádek jsou přidány všechny signály v celém designu. Příklad kompletního souboru je v příloze č. P.6. [8]

4.2 Komponenty verifikačního prostředí

V této kapitole jsou psány jednotlivé komponenty verifikačního prostředí. Na obr. č. 4.2 je blokové schéma samotného verifikačního prostředí bez okolních podpůrných komponent.



Obr. 4.2: Blokové schéma verifikačního prostředí

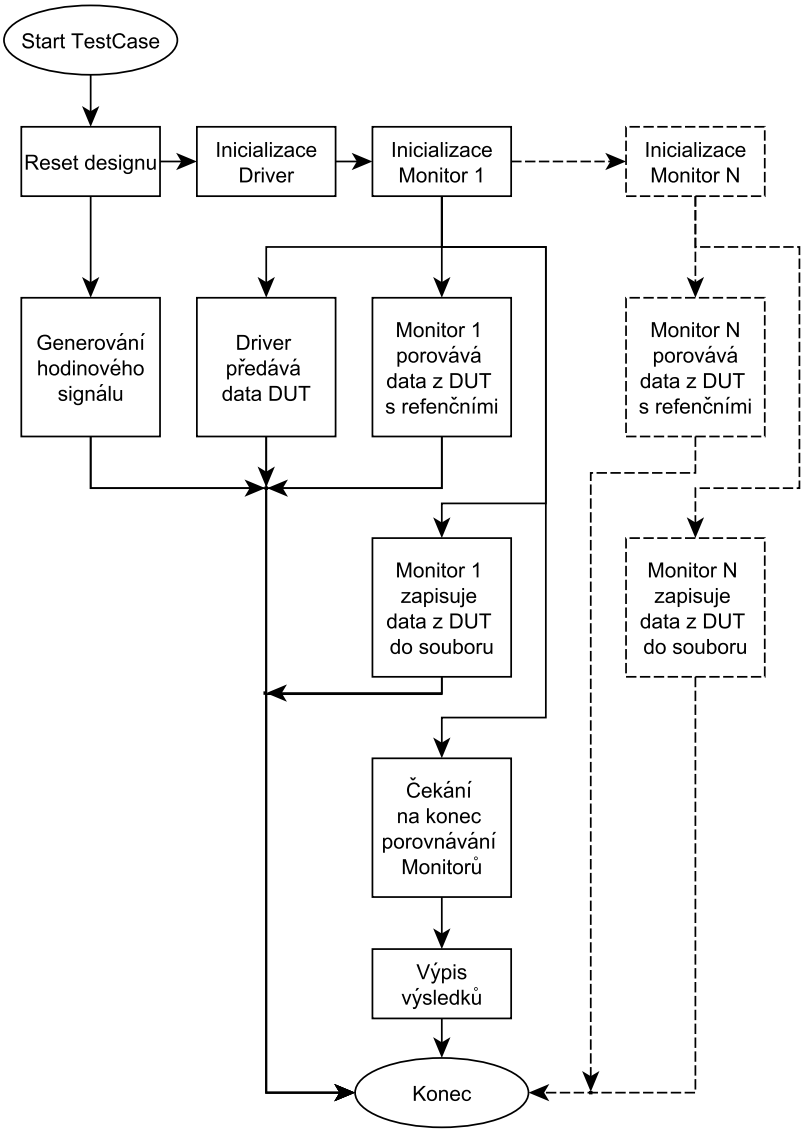
U všech komponent je nastaveno časové rozlišení i přesnost simulace na 1 ns. Dále je používána perioda hodinového signálu 1 μs.

4.2.1 Komponenta *TestCase*

Komponenta *Test Case* řídí prakticky celou verifikaci a volá funkce jednotlivých komponent. Na začátku je vytvořena proměnná třídy „scoreboard“ (více v kapitole č. 4.2.5). Na obr. č. 4.3 je průběh funkce v *TestCase*. Nejprve je celý design resetován. Následně je zahájeno generování hodinového signálu, jsou inicializovány moduly *Driver* a všechny moduly *Monitor*. Poté začíná *Driver* číst vstupní data a předává je *DUT*. Komponenty *Monitor* porovnávají referenční data s výstupními daty z *DUT* a zapisují je do souborů.

Jakmile všechny komponenty *Monitor* porovnají všechna referenční data, jsou vypsány výsledky (počty chyb) z každého *Monitor*. Poté je simulace ukončena.

Funkce hodinového signálu, načítání vstupních dat v *Driver* a porovnávání a zápis výstupních dat z *DUT* běží souběžně.



Obr. 4.3: Vývojový diagram průběhu testu

4.2.2 Komponenta *Test Bench*

V komponentě *Test Bench* jsou vloženy a propojeny komponenty *Driver*, *DUT* a *Monitor*. Dále komponenta obsahuje dvě funkce typu „task“. První funkcí je resetování designu s názvem *reset_design()*, druhou je generování hodinového signálu v nekonečné smyčce *while(1)* s názvem *clock_generation()*.

4.2.3 Komponenta *Driver*

Komponenta *Driver* obsahuje dvě funkce. Funkce *init()* slouží na inicializaci. V této funkci je otevřen soubor „data_in“. Pokud je soubor prázdný, simulace končí. Dále

je zde nastaven povolovací signál „enable“ pro *DUT*. Pro decimační CIC filtr je signál nastaven pouze do log. 1. Pro ostatní filtry je vytvořena funkce *enable_init()*, která generuje signál jako obdélníkový průběh o definované střídě. Nastavení střídy je nutné, aby filtry měly dostatek času na výpočet hodnot. Střída je v poměru $1 : n$, kde n je určováno podle vlastností filtru. V tabulce č. 4.1 je přehled, jak se parametr n určuje.

Tab. 4.1: Určení střídy 1:n signálu „enable“

Typ filtru	Parametr n
interpolační CIC	n = interpolační faktor – diferenční zpoždění
FIR	n = délka filtru – 1
decimační FIR s posuvným registrem	n = délka subfiltru – 1
decimační FIR s pamětí pro vstupní data	n = délka subfiltru
interpolační FIR	n = (interpolační faktor * počet sekcí) – 1

Pro zvýšení pokrytí zdrojového kódu pro FSM u FIR filtrů je v průběhu simulace měněna střída signálu „enable“ v intervalu <1:n; 1:n+5>.

Navíc pokud u FIR filtrů použijeme místo posuvného registru paměť RAM pro vstupní data (Random Access Memory – Paměť s přímým přístupem) a zároveň nevynecháme reset při zapisování dat, tak je nutné začít signál „enable“ generovat s určitou prodlevou. Tato prodleva je opět dána vlastnostmi filtru a je v tabulce č. 4.2.

Tab. 4.2: Nutné prodlevy před spuštění signálu „enable“

Typ filtru	Prodleva d
FIR	d = délka filtru
decimační FIR	d = decimační faktor * počet sekcí filtru
interpolační FIR	d = interpolační faktor * počet sekcí filtru

Druhá funkce *read_input_data()* obstarává postupné čtení dat z textového souboru a jejich posílání do DUT. Dále posílá i povolovací signál „enable“. Pokud je signál „enable“ v log. 1, tak je načten jeden vzorek vstupních dat. Poté funkce čeká jednu periodu hodinového signálu. V případě decimačního CIC filtru jsou tedy vstupní data posílána do DUT každou periodu hodinového signálu. Celá funkce opět běží v nekonečné smyčce *while(1)*.

4.2.4 Komponenta *Monitor*

Komponenta *Monitor* obsahuje tři funkce. První je stejně jako u komponenty *Driver* funkce *init()*, která otevře soubor s referenčními daty pro čtení a soubor pro zapisování výstupních dat z DUT. Další funkcí je samotné zapisování dat do souboru. V této funkci jsou také načítány vzorky referenčních dat. Pokud *Monitor* načte poslední vzorek, je nastavena proměnná „end_of_compare“ do log. 1.

Třetí funkce porovnává výstupní data s referenčními a má název *test()*. Porovnávání probíhá pokud je signál „input_en“ (tedy výstupní signál „output_en“ z DUT.) v log. 1 a zároveň je proměnná „end_of_compare“ v log. 0 (tedy nebyly porovnány všechny referenční vzorky). Pokud nastane chyba, tak je okamžitě vypsána do okna „Transcript“ a je přičtena do celkového počtu chyb.

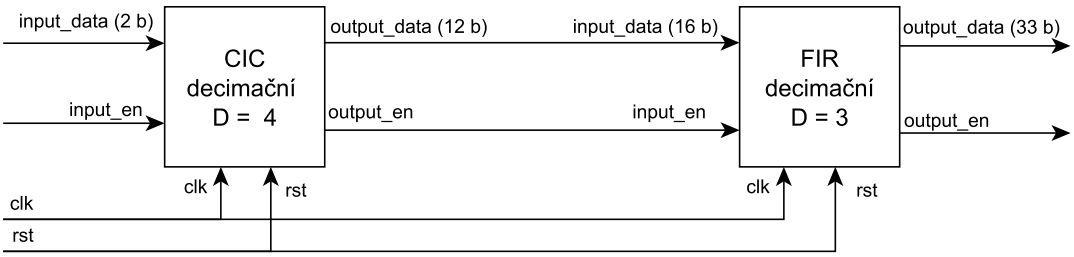
Pro každý filtr v kaskádě je nutné použít jeden *Monitor*.

4.2.5 Třída *Scoreboard*

Třída *Scoreboard* v případě tohoto verifikačního prostředí slouží k zjištění funkčního pokrytí, tzn. zda jsou využity všechny kombinace vstupů nebo výstupů. Pro každý filtr je vytvořena skupina bodů pokrytí – covergroup. V každé skupině jsou pak dva body pokrytí – coverpoints. Jeden pro vstup a jeden pro výstup filtru. Body pokrytí mají také nastaveno rozdělení intervalu všech kombinací. [14] [15]

4.3 Výsledky verifikace a pokrytí

Pro vyhodnocení verifikace bude použito příkladu zapojení s decimačním CIC a s decimačním FIR filtrem (obr. č. 4.4).



Obr. 4.4: Příklad zapojení

Verifikace proběhla bez chyb a všechna výstupní data se shodují s referenčními. V okně „Trancript“ (obr. č. 4.5) jsou vypsány výsledky z jednotlivých komponent

Monitor. Kde „stage1“ je první filtr a „stage2“ druhý filtr. Verifikace trvala 1798 μs a byla ukončena po porovnání všech referenčních dat z bloku *Test Case*.

```
# Test(stage1) OK -           0 error.
# ------------------------------------------------
# Test(stage2) OK -           0 error.
# ** Note: $stop    : testcase.sv(36)
#    Time: 1798 us  Iteration: 0  Instance: /testcase
# Break in Task end_of_sim at testcase.sv line 36
```

Obr. 4.5: Výsledky verifikace

Na obr. č 4.6 jsou pak výsledky funkčního pokrytí přímo z programu QuestaSim. Výsledky jsou v záložce „Covergroups“. Jsou zde vidět dvě skupiny bodů pokrytí a v každé skupině bod pokrytí pro vstupní a výstupní data. Výsledky jsou rovněž ukládány do souboru „coverage_functional.txt“.

Name	Coverage	Goal	% of Goal	Status	Included
/testcase_sv_unit/scoreboard					
TYPE cg_stage1	100.0%	100	100.0%		
CVP cg_stage1::cp_stage1_input_data	100.0%	100	100.0%		
CVP cg_stage1::cp_stage1_output_data	100.0%	100	100.0%		
TYPE cg_stage2	50.0%	100	50.0%		
CVP cg_stage2::cp_stage2_input_data	20.0%	100	20.0%		
CVP cg_stage2::cp_stage2_output_data	80.0%	100	80.0%		

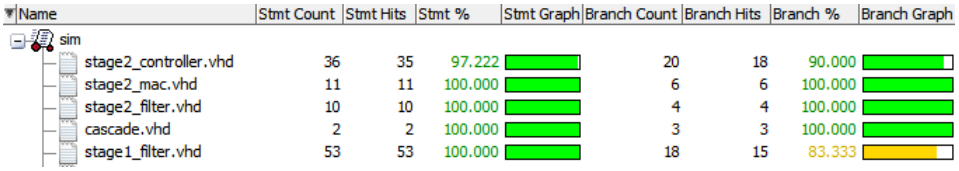
Obr. 4.6: Funkční pokrytí

Z výsledků funkčního pokrytí je vidět, že vstupní generovaná data pro první filtry pokryla interval vstupních i výstupních hodnot prvního filtru dostatečně. Pro druhý filtr, který má o 4 bity větší bitovou šířku vstupu než je výstup prvního filtru, je patrné, že vstupní data nenabývala všech možných hodnot. Výstup druhého filtru byl pokryt na 80 %. Výsledné funkční pokrytí je 75.0 %

Na obr. č. 4.7, 4.8 a 4.9 jsou výsledky pokrytí zdrojového kódu. Výsledky jsou v programu QuestaSim v záložce „Files“ nebo „sim“. Podrobné výsledky pokrytí jsou v souboru „coverage_code.txt“. Lze v něm nalézt, které části zdrojového kódu byly pokryty a které nikoli.

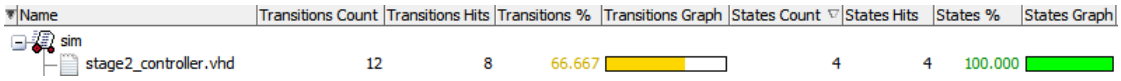
Z obrázků lze vyčíst maximální počty pokrývaných míst ve zdrojovém kódu a reálně pokryté části. Na obr. č. 4.7 je pokrytí téměř u všech částí zapojení filtrů na 100 %. U prvního filtru není pokryta jedna větev a u kontroléru druhého filtru není pokryto jedno přiřazení a dvě větve.

Na obr. č. 4.8 je zobrazeno pokrytí stavových automatů použitých ve zdrojových souborech pro FIR filtr. „Transition“ znamená pokrytí přechodů mezi stavy. Z výsledků je patrné, že nebyly pokryty všechny kombinace přechodů



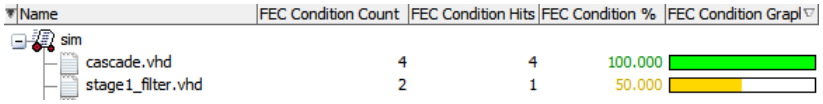
Name	Stmt Count	Stmt Hits	Stmt %	Stmt Graph	Branch Count	Branch Hits	Branch %	Branch Graph
sim								
stage2_controller.vhd	36	35	97.222		20	18	90.000	
stage2_mac.vhd	11	11	100.000		6	6	100.000	
stage2_filter.vhd	10	10	100.000		4	4	100.000	
cascade.vhd	2	2	100.000		3	3	100.000	
stage1_filter.vhd	53	53	100.000		18	15	83.333	

Obr. 4.7: Pokrytí zdrojového kódu – přiřazení a větve



Name	Transitions Count	Transitions Hits	Transitions %	Transitions Graph	States Count	States Hits	States %	States Graph
sim								
stage2_controller.vhd	12	8	66.667		4	4	100.000	

Obr. 4.8: Pokrytí zdrojového kódu – stavové automaty



Name	FEC Condition Count	FEC Condition Hits	FEC Condition %	FEC Condition Graph
sim				
cascade.vhd	4	4	100.000	
stage1_filter.vhd	2	1	50.000	

Obr. 4.9: Pokrytí zdrojového kódu – podmínky

stavů. Toto pokrytí je však dáno způsobem funkce stavového automatu. Je možné, že automat nepovoluje přechody mezi některými stavy.

Na obr.č 4.8 je pokrytí podmínek. U prvního filtru byla pokryta jedna podmínka ze dvou.

Výsledné pokrytí zdrojového kódu je 88,9 %.

Po skončení simulace pracovní adresář obsahuje soubory uvedené v příloze č. P.1. Soubor použitý pro vytvoření referenčního modelu a získání referenčních dat je v příloze č. P.3. Dále jsou v přílohách soubory pro sestavení verifikačního prostředí (příloha č. P.5) a soubor pro nastavení zobrazování signálů v simulačním okně (příloha č. P.6). Celý tento příklad je obsahem přiloženého CD.

4.4 Ukládané soubory

Po skončení verifikace jsou uloženy tyto soubory:

- *coverage_code.txt* – výsledky pokrytí zdrojového kódu,
- *coverage_functional.txt* – výsledky funkčního pokrytí,
- *data_out_X* – výstupní data filtru (X je pořadí filtru v kaskádě),
- *transcript* – provedené příkazy během verifikace a výsledky testů,
- *vsim* – záznam okna verifikace.

4.5 Výsledky verifikace pro konkrétní zapojení

V následujících dvou kapitolách jsou v tabulkách výsledky verifikace pro filtry s různými parametry. Hodinový signál i reset jsou u jednotlivých filtrů nastaveny tak, aby byly vyzkoušeny všechny kombinace. Všechny testované filtry jsou součástí přiloženého CD. V tabulce č. 4.3 a č. 4.4 je pod pojmem „faktor“ myšleno decimační nebo interpolační faktor a to podle typu filtru na příslušném řádku tabulky.

Tab. 4.3: Výsledky pro CIC filtry

Typ filtru	Faktor	Počet sekcí	Počet chyb [-]	Funkční pokrytí [%]	Pokrytí zdrojového kódu [%]
decimační CIC	4	4	0	100	77,8
decimační CIC	32	3	0	100	77,8
decimační CIC	32	5	0	100	77,8
decimační CIC	32	10	0	100	77,8
interpolační CIC	4	5	0	90	98,9
interpolační CIC	12	5	0	80	98,7
interpolační CIC	32	10	0	100	98,9
průměr	–	–	0	96	86,8

V tabulce č. 4.3 jsou výsledky pro filtry CIC. Průměrná hodnota funkčního pokrytí dosahuje 96 % a pokrytí zdrojového kódu dosahuje 86,8 %.

Tab. 4.4: Výsledky pro FIR filtry

Typ filtru	Faktor	Délka filtru	Počet chyb	Funkční pokrytí [%]	Pokrytí zdrojového kódu [%]
decimační FIR	3	10	0	90	94,5
decimační FIR	3	31	0	90	94,5
FIR	–	11	0	90	94,5
FIR	–	46	0	90	95,8
interpolační FIR	3	10	0	80	95,8
interpolační FIR	2	11	0	95	95,8
interpolační FIR	3	51	0	90	95,8
průměr	–	–	0	89	95,2

V tabulce č. 4.4 jsou výsledky pro filtry FIR. Průměrná hodnota funkčního pokrytí dosahuje 89 % a pokrytí zdrojového kódu dosahuje 95,2 %.

Tab. 4.5: Výsledky pro dvoustupňový design

První filtr	Bitová šířka vstupu	Bitová šířka výstupu	Druhý filtr	Bitová šířka vstupu	Bitová šířka výstupu	Počet chyb [-]	Funkční pokrytí [%]	Pokrytí zdrojového kódu [%]
CIC_dec (4,6)	s2,0	s14,0	CIC_dec (4,6)	s2,0	s14,0	0	90,0	93,9
CIC_dec (4,5)	s2,0	s12,0	FIR_dec (3,6)	s16,15	s33,31	0	75,0	89,0
CIC_dec (32,4)	s2,11	s11,0	FIR_dec (2,22)	s10,0	s10,0	0	67,5	79,8
CIC_dec (4,5)	s2,0	s12,0	FIR (4)	s16,15	s33,31	0	75,0	89,4
FIR (4)	s2,0	s18,15	CIC_int (2,2)	s19,18	s20,18	0	87,5	96,0
FIR (4)	s2,0	s18,15	FIR_int (2,4)	s19,17	s35,32	0	77,5	83,9
FIR (4)	s2,0	s18,15	FIR (4)	s19,17	s34,32	0	77,5	82,4
průměr	–	–	–	–	–	0	78,6	87,8

V tabulce č. 4.5 znamená:

- CIC_dec (4,6) – CIC_decimační (decimační faktor, počet sekcí),
- FIR_dec (3,6) – FIR_decimační (decimační faktor, délka filtru),
- FIR (4) – FIR (délka filtru),
- CIC_int (2,2) – CIC_interpolační (interpolační faktor, počet sekcí),
- FIR_int (2,4) – FIR_interpolační (interpolační faktor, délka filtru),
- s2,0 – bitová šířka celého čísla je 2, bitová šířka desetinné části je 0.

V tabulce č. 4.5 jsou výsledky pro kaskádu dvou filtrů. Průměrná hodnota funkčního pokrytí dosahuje 78,6 % a pokrytí zdrojového kódu dosahuje 87,8 %. Nižší hodnota funkčního pokrytí je dána obvykle nižším pokrytím druhého filtru, kdy výstupní data prvního filtru neobsáhnou celý interval vstupního rozsahu druhého filtru.

Tab. 4.6: Výsledky pro trojstupňový design

Typ filtru	Bitová šířka vstupu	Bitová šířka výstupu	Počet chyb [-]	Funkční pokrytí [%]	Pokrytí zdrojového kódu [%]
FIR_dec (2,5)	s2,0	s11,10	0	85,0	94,6
FIR (4)	s13,12	s29,27			
CIC_int (2,2)	s30,28	s31,28			

V tab. č. 4.6 jsou zobrazeny výsledky simulace pro trojstupňový design. Filtry jsou zapojeny za sebou v pořadí, v jakém jsou uvedeny v tabulce. Funkční pokrytí dosahuje 85.0 % pokrytí zdrojového kódu 94,6 %.

5 UŽIVATELSKÝ MANUÁL

V následují kapitole je rozebráno ovládání programu na konkrétním příkladu zapojení.

5.1 Vytvoření souborů s popisem digitálních filtrů

Vytvoření digitálních filtrů je možné v programovém prostředí Matlab v programu Filter Design & Analysis Toolbox, který lze spustit v Matlabu příkazem „fdatool“. Program je velice intuitivní, avšak je nutné nastavit „Filter Arithmetic“ na Fixed–Point. Pro uložení filtru do souboru s příponou „*.fcf“ je následující postup File → Export. Poté nastavit export do „Coefficient File (ASCII)“, formát „Binary“.

5.2 První spuštění programu

Při prvním spuštění programu (na obr. č. 3.1) je potřeba nastavit cestu ke generátoru VHDL. Cestu ke generátoru je možno změnit v menu File → Main Settings. V tomto menu je dále možné měnit parametry pro generátor jako je např. parametr „-v 2“. Jedná se o detail reportu generátoru. Čím vyšší číslo, tím detailnější report je k dispozici. Popis dalších parametrů je po spuštění v okně Report.

S ohledem na standardy VHDL by se v celém programu neměla používat diakritika a mezery v názvech.

5.3 Vytvoření projektu v programu

Po úspěšném zadání cesty ke generátoru je možné přidat filtry, nastavit typ (FIR nebo CIC), jméno a cestu k souboru s popisem filtru. Je nutné ke všem vloženým filtrům vložit cestu k souboru s popisem filtru. V opačném případě program vyzve k opravě tohoto nedostatku.

V menu Options → Settings lze měnit název hlavní entity filtrů, nastavovat aktivní hranu hodinového signálu a typ resetu. Také je možnost vygenerovat VHDL popis a tedy i soubory verifikačního prostředí do odlišeného adresáře oproti tomu, ze kterého je spouštěn celý projekt. Dále lze vložit krátký popis vytvořeného zapojení.

Po stisknutí funkce „Generate HDL“ nebo „Verify“ program vyzve k uložení designu. Jakmile je zapojení uloženo, začne probíhat požadovaná funkce. V případě verifikace proběhne funkce, jak bylo popsáno v kapitole č. 3. Program Matlab je po výpočtech, které mohou trvat několik minut, ukončen automaticky. Následuje

spuštění programu QuestaSim nebo ModelSim a spuštění verifikace. Ukončení simulace je mimo jiné zobrazeno v okně „Transcript“ jako je na obr. č. 4.5. Program po ukončení simulace zůstává otevřený a dovoluje tak uživateli, aby mohl analyzovat zobrazované průběhy, počet chyb nebo pokrytí testu. Pro korektní ukončení programu je nutné nejprve ukončit simulaci. Buď příkazem „quit -sim“ a nebo pomocí menu Simulate → End Simulation.

ZÁVĚR

Cílem této práce bylo seznámit se s problematikou digitálních filtrů a jejich návrhem v programu MATLAB. Dále se seznámit s verifikací a s verifikačními metodikami. Hlavním úkolem byl návrh verifikačního prostředí pro ověření funkce systémů s digitálními filtry.

Diplomová práce navazuje na program ASIC Filter Generator umožňující generovat VHDL popis systémů s FIR a CIC filtry. Výsledek diplomové práce je možnost verifikace těchto filtrů v rámci zmíněného programu, který je napsán v jazyce C++ v prostředí Qt4 Framework. Generování verifikačního prostředí je taktéž napsána v tomto multiplatformním prostředí.

Verifikační prostředí je napsáno v jazyce SystemVerilog a je spouštěno v programu ModelSim nebo QuestaSim. Soubory verifikačního prostředí jsou generovány funkcemi v jazyce C++ podle vstupních parametrů jednotlivých filtrů (bitové šířky vstupů a výstupů) i nastavení celého systému (hodinový signál, reset).

Jako stimuly pro verifikovaný systém slouží průběhy signálů (sinusový, obdélníkový, pilovitý, trojúhelníkový apod.), které jsou generovány podle parametrů filtrů.

Verifikační prostředí nepracuje přímo s referenčním modelem. Ten je vytvořen v programovém prostředí Matlab před samotnou verifikací a do procesu verifikace tak vstupují referenční data získaná z programu Matlab.

Hlavními sledovanými parametry verifikačního prostředí je počet chyb výstupů jednotlivých filtrů v systému oproti referenčním hodnotám, funkční pokrytí a pokrytí zdrojového kódu.

Funkční pokrytí u otestovaných systémů dosahuje minimálně 78,6 % a pokrytí zdrojového kódu minimálně 86,8 %.

LITERATURA

[1] Digital Filtering. *WaveMetric* [online]. [cit. 2015-12-06]. Dostupné z: http://www.wavemetrics.com/products/igorpro/dataanalysis/signalprocessing/digitalfilters.htm.

[2] Realizace filtrů FIR a IIR v programovacím jazyce C#. *Programujte.com* [online]. 2010 [cit. 2015-12-06]. Dostupné z: http://programujte.com/clanek/2010050400-realizace-filtru-fir-a-iir-v-programovacim-jazyce-c/.

[3] ALTERA. *Understanding CIC Compensation Filters* [online]. 2007 [cit. 2015-11-15]. Dostupné z: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/an/an455.pdf.

[4] E. B. Hogenauer, An economical class of digital filters for decimation and interpolation *IEEE Transactions on Acoustic, Speech and Signal Processing*. 1981 [cit. 2015-12-07].

[5] SMITH, Julius O. Introduction to digital filters. *Center for Computer Research in Music and Acoustics (CCRMA), Stanford University* [online]. [cit. 2015-12-07]. Dostupné z: https://ccrma.stanford.edu/~jos/fp/.

[6] TESAŘÍK, J. *Grafické uživatelské rozhraní pro návrh filtrů.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2014 [cit. 2015-12-03]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=88400.

[7] PRISTACH, M. *Návrh optimalizovaných architektur digitálních filtrů pro nízkopříkonové integrované obvody.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015 [cit. 2016-05-10]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=111185.

[8] ModelSim User's Manual. *Mentor Graphics* [online]. 2007 [cit. 2016-04-03]. Dostupné z: http://www.microsemi.com/document-portal/doc_view/131619-modelsim-user.

[9] TestBench. *UVM Tutorial* [online]. 2007 [cit. 2015-12-03]. Dostupné z: http://www.testbench.in/UT_00_INDEX.html.

[10] Mentor Graphics. *Verification Academy* [online]. 2010 [cit. 2015-12-03]. Dostupné z: https://verificationacademy.com/.

[11] Universal Verification Methodology (UVM) 1.2 Class Reference. In: *Accellera* [online]. 2014 [cit. 2015-12-03]. Dostupné z: http://www.accellera.

org/images/downloads/standards/uvm/UVM_Class_Reference_Manual_1.2.pdf.

[12] Universal Verification Methodology (UVM) 1.2 User's Guide. In: *Accellera* [online]. 2015 [cit. 2015-12-03]. Dostupné z: http://www.accellera.org/images//downloads/standards/uvm/uvm_users_guide_1.2.pdf.

[13] MathWorks. *Documentation* [online]. 2016 [cit. 2016-05-10]. Dostupné z: http://www.mathworks.com/help/matlab/.

[14] Gopikrishna, Maddipati. *Test Bench* [online]. 2007 [cit. 2016-05-10]. Dostupné z: http://www.testbench.in/.

[15] Deepak Kumar Tala. *ASIC WORLD* [online]. 2014 [cit. 2016-05-10]. Dostupné z: http://www.asic-world.com/.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ASIC Application Specific Integrated Circuit – Zákaznický integrovaný obvod

CIC Cascaded Integrating Comb – Hřebenový integrační filtr

DUT Design/device Under Test – Testovaný systém/zařízení

FIR Finite Impulse Response – Filtr s konečnou impulzní odezvou

FSM Finite State Machine – Konečný statový automat

HVL Hardware Verification Language – Jazyk pro verifikaci digitalních obvodů

IEEE Institute of Electrical and Electronics Engineers – Institut pro elektrotechnické a elektronické inženýrství

IIR Infinite Impulse Response – Filtr s nekonečnou impulzní odezvou

OVM Open Verification Methodology – Verifikační metodika od společnosti Cadence a Mentor Graphics

RAM Random Access Memory – Paměť s přímým přístupem

RTL Register-Transfer Level – Metodika návrhu digitálních obvodů

SOS Second-Order Sections – Filtry druhých řádů

VHDL Very High Speed Integrated Circuits Hardware Description Language – Jazyk pro popis velmi rychlých integrovaných obvodů

VMM Verification Methodology Manual – Verifikační metodika od společnosti ARM a Synopsys

UVM Universal Verification Methodology – Verifikační metodika od společnosti Accelera

SEZNAM PŘÍLOH

P.1 Příklad struktury pracovního adresáře

- složka *cascade_CIC_dec_FIR_dec*
  - složka *hdlsrc* - VDHL popis generovaný z Matlabu
  - složka *src* - VHDL popis z generátoru ASIC Filter generator
    - složka *design* - zdrojové soubory
      - soubor *cascade.vhd*
      - soubor *stage1_filter.vhd*
      - soubor *stage2_controller.vhd*
      - soubor *stage2_filter.vhd*
      - soubor *stage2_mac.vhd*
    - složka *tb* - „TestBench“ vytvářený programem ASIC Filter Generator
      - soubor *top_tb.vhd*
  - složka *verification* - složka se soubory pro verifikaci
    - složka *work* - složka s vytvořenou knihovnou během verifikace
    - soubor *all.do* - nastavení a spuštění verifikačního prostředí
    - soubor *cascade_tb.sv* - „TestBench“ - propojuje komponenty verifikačního prostředí s DUT
    - soubor *coverage_code.txt* - výsledky pokrytí zdrojového kódu
    - soubor *coverage_functional.txt* - výsledky funkční verifikace
    - soubor *data_in* - vstupní data pro první filtr
    - soubor *data_out_1* - výstupní data prvního filtru
    - soubor *data_out_2* - výstupní data druhého filtru
    - soubor *data_out_ref_1* - výstupní referenční data prvního filtru
    - soubor *data_out_ref_2* - výstupní referenční data druhého filtru
    - soubor *driver.sv* - komponenta Driver
    - soubor *monitor_1.sv* - komponenta Monitor - pro výstup prvního filtru
    - soubor *monitor_2.sv* - komponenta Monitor - pro výstup druhého filtru
    - soubor *run* - nastavení adresáře a spuštění souboru all.do
    - soubor *scoreboard.sv* - komponenta kontrolují funkční pokrytí testu
    - soubor *testcase.sv* - „TestCase“ - řídí celou verifikaci
    - soubor *transcript* - provedené příkazy během verifikace
    - soubor *vsim* - záznam okna verifikace
    - soubor *wave.do* - zobrazené signály v okně verifikace.
  - soubor *cascade.xml* - uložený design
  - soubor *filter_cic.fcf* - popis prvního filtru
  - soubor *filter_fir.fcf* - popis druhého filtru
  - soubor *get_data_ref.m* - soubor pro Matlab k získání referenčních dat.

P.2 XML soubor designu (cascade.xml)

```
<?xml version="1.0" encoding="utf-8" ?>
<filter_generator format="1.0"/>

<settings>
    <description text="CIC_dec -&gt; FIR_dec"/>
    <directory dir=""/>
    <general clk="rising_edge" reset="asynch_low" hdl="vhdl" hdl_tb="vhdl-08"/>
</settings>

<design name="cascade" library="work">
    <stage name="stage1" value="Stage 1" library="filter" device="cic">
        <param file="filter_cic.fcf" output_reg="no" alternative="no"/>
    </stage>

    <stage name="stage2" value="Stage 2" library="filter" device="fir">
        <param file="filter_fir.fcf" memory="regs" reset="yes"/>
    </stage>

</design>

<design name="top_tb" library="testbench"/>
```

P.3 Soubor pro spuštění v Matlabu (get_data_ref.m)

```
function Hd = filt
%-------------------------------------------------------------------------
% stage1 - CascadedIntegrator-CombDecimator
%-------------------------------------------------------------------------
% Filter Specification
factor  = 4;  % Decimation Factor
diffd   = 1;  % Differential Delay
numsecs = 5;  % Number of Sections

% CIC filter design
Hd_1 = mfilt.cicdecim(factor, diffd, numsecs);

% Set the arithmetic property
set(Hd_1, 'InputWordLength',2, ...
    'inputFracLength',0, ...
    'FilterInternals', 'MinWordLengths', ...
    'OutputWordLength',12, ...
    'OutputFracLength',0);

% info(hcic)
fcfwrite(Hd_1, 'filter_1.fcf', 'bin');

%-------------------------------------------------------------------------
% Set directory for HDL
dir = pwd;
hdl_dir  = strcat(dir, '/hdlsrc');

% Generating HDL code with Testbench
generatehdl(Hd_1, ...
'TargetLanguage', 'VHDL', ...
'TargetDirectory', hdl_dir, ...
'AddInputRegister', 'on', ...
'AddOutputRegister', 'off', ...
'GenerateHDLTestbench', 'on', ...
'TestBenchStimulus',  {'impulse', 'step', 'ramp', 'chirp', 'noise'});
```

```
%------------------------------------------------------------------------
% stage2 - Direct-FormFIRPolyphaseDecimator
%------------------------------------------------------------------------
% Filter Specification
decf = 3; % Decimation Factor
N = 6; % order of filter

% Numerator coefficient vector
Numerator =  fi(0,1,16,16);
Numerator.bin = '1110111010111001 1001000101111111 1110001101110 10
1110001101110 1001000101111111 1110111010111001 ';

Hd_2 = dfilt.dffir(Numerator);
num  = get(Hd_2, 'Numerator'); % Get the numerator from the curren filter
Hd_2  = mfilt.firdecim(decf, num);

% Set the arithmetic property
set(Hd_2, 'Arithmetic', 'fixed', ...
    'CoeffWordLength', 16, ...
    'CoeffAutoScale', false, ...
'NumFracLength', 16, ...
    'Signed', true, ...
    'InputWordLength',16, ...
    'inputFracLength',15, ...
    'FilterInternals', 'FullPrecision');

%  info(hcic)
fcfwrite(Hd_2, 'filter_2.fcf', 'bin');

%------------------------------------------------------------------------
% Set directory for HDL
dir = pwd;
hdl_dir  = strcat(dir, '/hdlsrc');

% Generating HDL code with Testbench
generatehdl(Hd_2, ...
'TargetLanguage', 'VHDL', ...
'TargetDirectory', hdl_dir, ...
'AddInputRegister', 'on', ...
'AddOutputRegister', 'off', ...
'GenerateHDLTestbench', 'on', ...
'TestBenchStimulus',  {'impulse', 'step', 'ramp', 'chirp', 'noise'});
```

```
%------------------------------------------------------------------------
% Create final filter
Hd = mfilt.cascade(Hd_1);
addstage(Hd,Hd_2);

%------------------------------------------------------------------------
% Get reference data
%  set fi object parameters with respect to Hd filter input
in_len  = Hd.Stage(1).InputWordLength;
in_frac = Hd.Stage(1).InputFracLength;
input_data = fi(0, 1, in_len, in_frac);

%  read and convert input data  to fixed-point
input_data.hex = char(textread('verification/data_in', '%s'));

%  filter data
output_data_1= filter(Hd.Stage(1), input_data);
output_data_2 = filter(Hd.Stage(2), output_data_1);

%  write output data
dlmwrite('verification/data_out_ref_1', upper(output_data_1.hex), 'delimiter', '');
dlmwrite('verification/data_out_ref_2', upper(output_data_2.hex), 'delimiter', '');

dlmwrite('wait_matlab',1);

exit % [EOF]
```

P.4 Soubor pro nastavení verifikace (all.do)

```
# clear transcript window
if { ![batch_mode] } {
.main clear
}

# quit any previous simulation
quit -sim

onbreak {resume}

# force quit in batch mode
if { [batch_mode] } {
onerror {quit -f}
}

# create the library
if [file exists work] {
vdel -all
}
vlib work

# compile design of project 'cascade'
vcom -work work -cover bcs "../src/design/stage1_filter.vhd"
vcom -work work -cover bfcs "../src/design/stage2_mac.vhd"
vcom -work work -cover bfcs "../src/design/stage2_controller.vhd"
vcom -work work -cover bfcs "../src/design/stage2_filter.vhd"
vcom -work work -cover bcs "../src/design/cascade.vhd"
```

```
# compile verification environment
vlog -work work driver.sv
vlog -work work monitor_1.sv
vlog -work work monitor_2.sv
vlog -work work cascade_tb.sv
vlog -work work scoreboard.sv
vlog -work work testcase.sv

# simulate design
vsim -novopt -t 1ps -coverage work.testcase work.cascade_tb

# disable NumericStd warnings
quietly set NumericStdNoWarnings 1

# definition of signals in a wave window
if { ![batch_mode] } {
source "wave.do"
}

# start of the simulation
run -all

# write transcript window
write transcript

# write coverage report - code coverage and functional coverage
coverage report -file coverage_code.txt -byfile -detail -code {s b c f}
coverage report -file coverage_functional.txt -byfile -detail -noannotate -option -cvg

# quit Questasim in batch mode
if { [batch_mode] } {
quit -f
}
```

P.5 Soubor pro přidání signálů do simulačního okna (wave.do)

```
# add waves
# general inputs
add wave -noupdate -format Logic /cascade_tb/clk
add wave -noupdate -format Logic /cascade_tb/rst
add wave -noupdate -format Literal -radix hexadecimal /cascade_tb/DUT/input_data

# stage1
add wave -noupdate -divider {stage1}
add wave -noupdate -format Logic /cascade_tb/mon_1/input_en
add wave -noupdate -format Literal -radix hexadecimal /cascade_tb/DUT/i_stage1/output_data
add wave -noupdate -format Literal -radix hexadecimal /cascade_tb/mon_1/output_data_ref

# stage2
add wave -noupdate -divider {stage2}
add wave -noupdate -format Logic /cascade_tb/mon_2/input_en
add wave -noupdate -format Literal -radix hexadecimal /cascade_tb/DUT/i_stage2/output_data
add wave -noupdate -format Literal -radix hexadecimal /cascade_tb/mon_2/output_data_ref

# add all signals in design
add wave -noupdate -divider {All signals in design}
add wave -r /cascade_tb/*

update
```

P.6 Obsah přiloženého CD

- **adresář ASIC Filter Generator**
  - **soubor ASIC_filter_generator.exe** – zkompilovaný program ASIC Filter Generator
  - **soubor generator.exe** – generátor VHDL popisu
  - nezbytné knihovny (*.dll)
- **adresář examples** – příklady vytvořených designů
- **adresář sources** – zdrojové soubory k vytvořené funkci
- **soubor DP_xtesar27** – elektronická verze diplomové práce