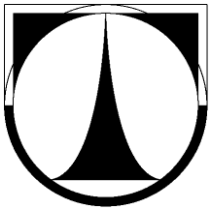
TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky
a mezioborových studií



DIPLOMOVÁ PRÁCE

Liberec 2013 Bc. Martin Jiříček

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 Elektrotechnika a informatika

Studijní obor: Informační technologie

Realizace aritmetických operací v pohyblivé řádové čárce pomocí programovatelných hradlových polí

Floating point arithmetic operations implementation using programmable gate arrays

Diplomová práce

Autor: **Bc. Martin Jiříček**

Vedoucí práce: doc. Ing. Milan Kolář, CSc.

Konzultant: Ing. Ondřej Zelinka, Ph.D.

V Liberci 2. 1. 2013

Zadání práce

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. O právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladu, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím práce a konzultantem.

Datum: 2. 1. 2013

Podpis:

Poděkování

Na tomto místě bych chtěl zejména poděkovat mému vedoucímu diplomové práce doc. Ing. Milanu Kolářovi, CSc. za jeho podporu a mnoho cenných rad. Dále bych chtěl poděkovat svým rodičům za morální i finanční podporu při studiu.

Abstrakt

Česky

Tato diplomová práce se zabývá návrhem aritmeticko-logické jednotky pro práci s čísly, používající pohyblivou řádovou čárku. Na začátku je čtenář nejprve seznámen s problematikou jednotek pro práci s pohyblivou řádovou čárkou. Dále je představena norma IEEE 754. Následně je ukázán jazyk, ve kterém bude problém řešen.

V další části je zobrazeno samotné řešení zadaného úkolu. Je zde ukázáno jádro programu a jednotlivých operací. Následně je provedena simulace funkčnosti programu a jeho částí. V poslední kapitole je poté poměřeno vyvinuté řešení s jinými již hotovými jednotkami.

Klíčová slova: Aritmeticko-logická jednotka, Čísla s pohyblivou řádovou čárkou, Jazyk VHDL, Standard IEEE 754

Anglicky

This thesis describes the design of an arithmetic logic unit working with numbers, using floating point. At the beginning, the reader is acquainted with the first issue of units which works with floating point. IEEE 754 standard is then presented. Subsequently is shown the language in which the problem will be solved.

The next section shows the actual solution of the given task. There is shown the core of the program and individual operations. In the next section was performed simulation of core and all operations. In last chapter is our unit compare against other already developed solutions.

Keywords: Arithmetic logic unit, Floating point numbers, VHDL language, IEEE 754 Standard

Obsah

Seznam zkratek

ABS absolute value - absolutní hodnota
ALU arithmetic logic unit - aritmeticko logická jednotka
FLOPS floating-point operations per second - operace v plovoucí čárce za sekundu
FPGA field-programmable gate array - programovatelné hradlové pole
FPU floating point unit - matematický koprocesor
HDL hardware description language - jazyk pro popis hardwaru
IEEE institute of electrical and electronics engineers - institut pro elektrotechnické a elektronické inženýrství
MOD modulus - modulo
NaN not a number - nejedná se o číslo
qNaN quiet not a number - tichá verze NaN
REM remainder - zbytek po dělení
ROL rotate left logical - logická rotace vlevo
ROR rotare right logical - logická rotace vpravo
SLA shift left arithmetic - aritmetický posun vlevo
SLL shift left logical - logický posun vlevo
sNaN signaling not a number - signalizující verze NaN
SRA shft right arithmetic - aritmetický posun vpravo
SRL shift right logical - logický posun vpravo
STD standard multivalue logic system for VHDL - standardní vícehodnotová logika pro VHDL
VHDL VHSIC hardware description language - VHSIC jazyk pro popis hardwaru
VHSIC very-high-speed integrated circuits - velice rychlé integrované obvody

Úvod

V dnešní době se snažíme veškeré prováděné operace či činnosti zautomatizovat tj. převést kontrolu těchto činností na nějakou samočinnou jednotku. Tato jednotka je nejčastěji počítač. Dříve počítač znamenal skříň, ke které byl připojen monitor, pokud půjdeme ještě dále, můžeme nahradit skříň místností či halou. Avšak v dnešní době už nelze počítač takto paušalizovat. Počítače na sebe vzaly mnoho podob a tak je můžeme například nalézt v mobilních telefonech, fotoaparátech, pračkách a třeba v dětských hračkách. To, co ale mají všechny tyto počítače společného, je jejich hlavní výkonná součástka. Této součástce se říká procesor.

Základním kamenem každého procesoru je aritmeticko-logická jednotka (dále jen zkráceně ALU - arithmetic logic unit). V této jednotce probíhají veškeré výpočty a není tedy možné, aby bez ní jakýkoliv počítač mohl fungovat. Dalším nutným prvkem je řadič. Tyto dvě jednotky jsou tedy nutností každého procesoru. V procesoru dále nalezneme mnoho jiných součástí. Jejich složení závisí především na použití daného procesoru.

Obsahem této práce bude návrh ALU. Konkrétně takové, jenž umožňuje práci s pohyblivou řádovou čárkou. Půjde především o to vytvořit jednotku, která bude schopna řešit základní matematické operace. Vytvoření ALU bude provedeno za pomocí programovacího jazyka VHDL. Výsledný program bude možno použít na programovatelných hradlových obvodech (FPGA).

V úvodu této práce si nejprve řekneme, co je to aritmeticko-logická jednotka, princip její činnosti a k čemu slouží. Také si řekneme rozdíly mezi jednotlivými typy. Dále se podíváme na problematiku pohyblivé řádové čárky. Popíšeme si, jaké máme normy, k čemu slouží a proč je vůbec výhodné ji používat. Následně si představíme samotný programovací jazyk, který je základním prostředkem pro uskutečnění této práce. V další kapitole přistoupíme již k samotnému řešení daného úkolu. Nejprve bude popsáno samotné jádro ALU a poté jednotlivé operace, které naše jednotka bude podporovat. Poté provedeme simulaci funkčnosti v programu Altera Quartus II. Zde zjistíme, jakým způsobem by mohla být vytvořená jednotka fyzicky fungovat. V poslední kapitole pak provedeme shrnutí získaných poznatků a porovnání vytvořené ALU s jiným již hotovými řešeními.

Cílem práce je navrhnout aritmeticko logickou jednotku sloužící pro práci s pohyblivou řádovou čárkou. Tato jednotka by měla být plně funkční a obsahovat základní operace. Jednotka by mohla být použít jak samostatně, tak i v rámci většího systému, kde by pracovala s čísly v pohyblivé řádové čárce. Také by zde měla být možnost dalšího rozšíření.

1 Aritmeticko-logická jednotka

1.1 Arithmetic Logic Unit – ALU

Aritmeticko-logická jednotka je základní výkonná jednotka každého procesoru. Zde probíhají veškeré aritmetické a logické operace. Dokonce i nejjednodušší mikroprocesory mají ALU, která slouží pro jediný účel, jako například udržování časovače. Základními aritmetickými operacemi jsou například sčítání, odčítaní, násobení atd. Mezi logické operace pak řadíme například logický součin, negaci a jiné. Samozřejmě lze i sestavit ALU pro výpočet složitějších operací jako je například odmocnina. V ten okamžik se ALU však stává mnohem komplikovanějším jak na výrobu, tak i na samotný provoz. Proto se v tomto ohledu snaží vývojáři najít rozumný kompromis. Jako první navrhl ALU americký matematik John von Neumann v roce 1945.

Základní princip je pak takový, že ALU načte data ze vstupního registru. Externí řídicí jednotka řekne, co má s těmito daty provést. Výsledky poté posílá ALU výstupného registru. Řídicí jednotka (řadič) je tedy zodpovědný za přemisťování dat mezi registry, ALU a externí pamětí.

V dnešní době připadá na jeden procesor i několik jednotlivých ALU, které pak mají každá svůj vlastní účel a jsou pro něj i speciálně konstruovány. Procesor také může mít více stejných ALU, které pracují na sobě téměř nezávisle. Standardní ALU pracuje s pevnou řádovou čárkou. Dále pak existují ALU, které pracují s pohyblivou řádovou čárkou a těm se poté říká FPU (Floating Point Unit – jednotka pro práci s pohyblivou řádovou čárkou). Právě takovouto jednotku budeme konstruovat.

1.2 Floating Point Unit – FPU

Jednotka pro práci s čísly s pohyblivou řádovou čárkou, někdy též označována jako matematický koprocesor. Hlavním rozdílem oproti klasické ALU je to, že pracuje s čísly, která jsou ve formátu pohyblivé řádové čárky. Dříve se FPU nacházela mimo hlavní procesor, avšak díky větší integraci jednotlivých součástek, se dnešní matematické koprocesory nacházejí již v hlavním procesoru vedle ostatních ALU. Mezi nejčastější operace FPU patří sčítání, odčítání, násobení, dělení a druhá odmocnina. Některé, zejména starší, mohou mít i exponenciální nebo trigonometrické výpočty. Většina moderních FPU to však řeší za pomocí softwarových knihoven.

Hlavním rozdílem mezi FPU a ALU je tedy v práci s čísly, konkrétně ALU standardně pracuje s dvojkovým kódem a FPU s čísly pohyblivou řádovou čárkou. FPU pak bývají v tomto ohledu mnohem složitější než klasické ALU.

Operace FPU – jelikož práce s pohyblivou řádovou čárkou je na rozdíl od práce s pevnou řádovou čárkou komplikovanější, FPU obsahují nejčastěji tyto operace.

- Sčítání
- Odčítaní
- Násobení

Tyto tři operace jsou těmi nejzákladnějšími, které musí mít každá FPU. Další operace, které může obsahovat, jsou pak tyto.

- Dělení
- Druhá odmocnina
- Další speciální operace

Základním problémem jednotky pro práci s pohyblivou řádovou čárkou je její zápis, respektive zápis čísel v tomto formátu a následná práce s tímto formátem. Pro sjednocení práce byl vyvinut standard IEEE 754 (více samostatná kapitola 3).

2 Pohyblivá řádová čárka

Reprezentace čísla, respektive to jak je znázorněno, je důležité pro to, aby následně mohlo být zakódováno jako řetězec určitých číslic (hodnot). Aritmetika je pak udávána jako množina operací nad množinou čísel. To znamená, že máme čísla, mezi kterými můžeme provádět matematické či jiné operace (sčítat, odčítat, atp.). Čísla lze reprezentovat několika různými způsoby. Jelikož budeme konstruovat jednotku pro práci s pohyblivou řádovou čárkou, budou nás zajímat pouze necelá čísla.

Standardně se používá libovolně dlouhý formát čísla s použitím čárky na přesně stanovené místo. Pokud čárka není poznamenána, znamená to, že je až na konci za posledním číslem. Tudíž nemá význam ji zapisovat. Dalším způsobem je pevná pohyblivá řádová čárka. To znamená, že systém má dopředu určeno, kde se bude nacházet řádová čárka. To má výhodu především v tom, že není potřeba uchovávat informace o lokaci řádové čárky a tím tedy ušetřit místo při práci s danými čísly. Například pokud budeme mít číslo 00123456 a systém bude mít pevnou desetinou čárku mezi 4. a 5. pozicí číslovek (počítáno zleva), pak lze číslo zapsat v klasickém zápisu jako 12,3456. [1][4]

Čísla pro práci s pohyblivou řádovou čárkou byla zavedena kvůli nedostatečné velikosti čísel s pevnou řádovou čárkou, respektive formátu, ve kterém jsou ukládány. Čísla, která jsou příliš malá nebo naopak příliš veliká, využívají právě onu pohyblivou řádovou čárku. Ta ovšem s sebou kromě již zmíněné výhody přináší i několik obtíží. Například při práci s hodně malými či velkými čísly. U těchto čísel nezbývá prostor pro zapsání těch nejmenších hodnot. Proto je nutné tyto hodnoty zaokrouhlovat. Tímto tedy přicházíme o přesnost. Další nevýhodou, s kterou se budeme potýkat, je vytvoření vlastní metodiky pro práci s těmito čísly. A to především u jednotlivých operací, které se tímto stávají obtížnější. [1]

Rychlost, jakou dokáže počítač zpracovat jednu operaci v pohyblivé řádové čárce, je velmi důležitá a používá se jako indikátor pro měření výkonu počítačů. Udává se v takzvaných FLOPS (floating-point operations per second) – operace s plovoucí řádovou čárkou za sekundu.

2.1 Zápis

Pro představu, jak funguje pohyblivá čárka, slouží následující příklad. Číslo 123,456789 je až na polohu desetinné čárky stejné jako číslo 123456,789. První se dá tedy zapsat jako $1{,}23456789 * 10^2$ a druhé pak $1{,}23456789 * 10^5$. Exponent nám tedy pohybuje s řádovou čárkou, odtud pohyblivá řádová čárka. [2]

Nejobecnější zápis čísel s pohyblivou řádovou čárkou je ve tvaru:

$$X = mantisa * základ^{exponent}$$

X – výsledné číslo,
mantisa – platná číslice (v našem přecházejícím příkladu to byla 1,23456789),
základ – zaleží na soustavě, ve které se počítá 2, 10, 16 (v našem příkladu 10),
exponent – číslo určené pro exponent. [2]

2.2 Typy čísel

Jedná se o základní typy čísel, se kterými se můžeme setkat při používání čísel s pohyblivou řádovou čárkou.

- Normalizovaná čísla – tato čísla jsou reprezentována jakoukoliv hodnotou uloženou v mantise a jakoukoliv hodnotou v exponentu s výjimkou nuly a maximální hodnoty. Mantisa vždy začíná jedničkou, za kterou je poté řádová čárka, z tohoto důvodu není nutné ji zapisovat.
- Denormalizovaná čísla – jsou reprezentována nulovým exponentem a nenulovou hodnotou v mantise. Tato čísla slouží nejčastěji pro ukládání velmi malých hodnot, které nelze uložit jako normalizovaná.
- Nekonečno – má maximální exponent tj. samé jedničky a naopak nulovou mantisu
- Nula – jak exponent, tak i mantisa jsou nulová.
- Nečíselná hodnota (NaN – Not a Number) – maximální exponent a nenulová mantisa. Norma IEEE 754 je dále dělí na sNaN, kde s znamená „signaling“ což znamená, že takové číslo vyhodí výjimku. Druhé možnost je poté qNaN, kde q znamená „quiet“ tedy bez výjimky. [2]

3 Standard IEEE 754

IEEE 754 je nejrozšířenější standard pro aritmetiku v pohyblivé řádově čárce. Standard specifikuje aritmetiku v pohyblivé řádové čárce pro čísla v dvojkové a desítkové soustavě.

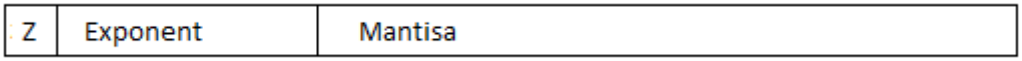
Základním účelem standardu IEEE 754 je poskytnou metody pro výpočty při práci s čísly s pohyblivou řádovou čárkou, které poskytnou stejný výsledek nehledě na to, zdali jejich zpracování umožnil hardware, software či kombinace obou. Výsledek takového výpočtu bude totožný, nezávislý na implementaci za předpokladu stejného vstupu. Chyby a chybové stavy budou správně oznámeny nehledě na implementaci.

Standard definuje následující:

- Formátování binárních a decimálních dat pro jejich výpočet a výměnu jejich výsledků
- Operace sčítání, odčítání, násobení, dělení, druhou odmocninu, porovnání a další operace
- Převod mezi formátem celých čísel a čísel plovoucí řádové čárky
- Převod mezi různými formáty čísel plovoucí řádové čárky
- Převod mezi čísly plovoucí řádové čárky a externí reprezentací
- Obsloužení výjimek

3.1 Formátování

Určuje, jakým způsobem bude číslo reprezentováno. Číslo je rozděleno na tři části (viz Obrázek 1). První část má velikost jednoho bitu, tento bit určuje znaménko celého čísla (0 – kladné, 1 – záporné). Druhá část je určena pro exponent, velikost tohoto bloku je dána typem přesností. Poslední část složí pro uložení hodnot mantisy, i zde opět záleží na typu přesnosti. Existuje pět základních typů formátování ve standardu IEEE 754. Tři pro počítání s binárními čísly o velikosti 32, 64 a 128 bitů (viz Tabulka 1). Další dva pak slouží pro práci s decimálními čísly, ty mají velikost 32 a 64 bitů (viz Tabulka 2). Tyto základní formáty potom rozšiřují formáty další. [1]



Obrázek 1 – Formátování

Tabulka 1 – Binární formátování

Velikost	znaménko	exponent	mantisa
32	1	8	23
64 bitů	1	11	52
128 bitů	1	15	112

Tabulka 2 – Decimální formátování

Velikost	znaménko	exponent	mantisa
64 bitů	1	11	52
128 bitů	1	15	112

Reprezentace čísla v pohyblivé řádové čárce se skládá z následujících věcí:

- Trojice (znaménko, exponent, mantisa)
  $(-1)^{znaménko} * b^{exponent} * mantisa$; b=soustava (2 nebo 10)
- +nekonečno, -nekonečno
- qNaN, sNaN

3.2 Atributy

Atributy jsou logicky přiřazeny k určité části programu. Slouží pro úpravu numerické či výjimkové sémantiky daného kódu. Zjednodušeně se jedné o hodnoty určených parametrů, na jejich základě se vykoná příslušná část programu.

Změnou atributů můžeme ovlivnit:

- způsob zaokrouhlování
- způsoby zpracování výjimek
- šířku slov
- a jiné

3.3 Zaokrouhlování

Slouží k normalizaci vypočteného čísla. Jelikož vypočtená čísla mohou překročit přesnost (počet bitů určených pro vyjádření daného čísla), která jim byla určena. Takovéto číslo je potřeba zaokrouhlit podle vybrané metody. Až takovýto výsledek je normovaný. Je také potřeba následně dát vědět, že číslo bylo zaokrouhleno a není tedy naprosto přesné.

Způsoby zaokrouhlování se dělí na pět druhů. První dva způsoby jsou tzv. zaokrouhlování k nejbližší hodnotě. Další tři způsoby jsou směrová zaokrouhlení.[1]

1. Zaokrouhlování k nejbližší hodnotě – nulový způsob – zaokrouhlí k nejbližší hodnotě, pokud se číslo nachází uprostřed, zaokrouhlí to k číslu, které má nejméně významný bit rovný nule.
2. Zaokrouhlování k nejbližší hodnotě – nenulový způsob – zaokrouhlí k nejbližší hodnotě, pokud se číslo nachází uprostřed, zaokrouhlí to k číslu, které je nejbližší vyšší hodnota (pro kladná čísla) nebo k nejbližší nižší hodnotě (pro záporná čísla).
3. Zaokrouhleni směrové – zaokrouhlení na nulu
4. Zaokrouhleni směrové – zaokrouhlení na + nekonečno
5. Zaokrouhleni směrové – zaokrouhlení na - nekonečno

3.4 Operace

Nejdůležitější částí kódu jsou samozřejmě operace. IEEE 754 na ně samozřejmě myslí a určuje daný postup, jak při nich postupovat. Nejdůležitější aritmetické operace, které IEEE 754 obsahuje, jsou:

- Sčítání – přijímá dva zdroje (a,b), jeden výsledek (x = a + b)
- Odčítání – přijímá dva zdroje(a,b), jeden výsledek (x = a - b)
- Násobení - přijímá dva zdroje(a,b), jeden výsledek (x = a * b)
- Dělení - přijímá dva zdroje(a,b), jeden výsledek (x = a / b)
- Druhá odmocnina – přijímá jeden zdroj (a), jeden výsledek (x = $\sqrt{a}$)

Ostatní operace:

- Konverze mezi formáty
- Porovnávání
- Klasifikace čísel (např.: zdali je číslo NaN)
- Třídění
- A jiné

3.5 Speciální hodnoty

Norma IEEE 754 obsahuje několik speciálních hodnot neboli speciálních označení pro zvláštní hodnoty. Z několika jsme se již seznámili, zde se pro pořádek seznámíme se všemi.

Nekonečno (∞) – jelikož aritmetika reálných čísel do jisté míry omezuje právě při používání čísel s pohyblivou řádovou čárkou, může se stát, že i právě tato čísla při nesprávném používání mohou dosáhnout nekonečných hodnot. Pro takové případy existuje hodnota nekonečno. Plusové nekonečno pro kladné nekonečné číslo a minusové nekonečno pro záporné číslo nekonečné hodnoty. Takovéto hodnoty nejčastěji vyvolají výjimkový stav, avšak existují i situace, kdy nikoliv. [1]

Případy, kdy výpočty proběhnou v pořádku i s nekonečnými hodnotami:

1. sčítání (x, ∞), (∞, x), odčítání (x, ∞), (∞, x), pro x = konečné číslo
2. násobení (∞, x), (∞, x), pro x ≠ 0
3. dělení (∞, x), (∞, x), pro x = konečné číslo
4. druhá odmocnina (+∞)
5. a jiné

Případy, kdy výpočty nahlásí chybu:

1. ∞ je neplatný operand
2. ∞ vznikne výpočtem dvou konečných čísel (přetečením nebo dělením nulou)

NaN (Not a Number) – tento výraz odpovídá hodnotě, jež nelze brát jako číslo, proto název výraz není číslo. Nejčastějším zástupcem je pak například zmiňované nekonečno. Vzniká například při dělení nulou. NaN jsou speciální permutací čísel, které systém vyhodnotí, právě jako nečíslo. Hodnota daného NaNu může obsahovat informace o vzniku tohoto nečísla. V normě IEEE 754 se dále děli na qNaN a sNaN.

qNaN (quiet NaN) – při jejich zpracování nedochází k vyvolání žádných dalších výjimek

sNaN (signal NaN) – při jejich zpracování dochází ke spuštění výjimek, nejčastěji například pokud došlo k přetečení či podtečení daného čísla [1]

3.6 Výjimky

Slouží jako prostředek při nečekané změně nebo při vzniku nestandardního problému. Výjimky oznamují, že se stalo něco, co vyžaduje buď nápravu původního zadání, nebo že musíme počítat s nestandardním výsledkem. Existuje pět základních výjimek v normě IEEE 754, nicméně je možné si vytvořit další potřebné výjimky. [1]

- Neplatná operace – takováto výjimka je signalizována pouze, pokud by daný výsledek nebyl definovatelný (například dělení nuly nulou atp.). Výsledek takovéto operace je pak qNaN.
- Dělení nulou – pokud je konečné číslo dělené nulou. Výsledek je nekonečno
- Přetečení – k přetečení dochází, pokud výsledek operace je větší než umožňuje zapsat daný formát čísla s pohyblivou řádovou čárkou. Výsledek je závislý na zvoleném typu zaokrouhlování.
- Podtečení – k podtečení dochází, pokud je výsledek operace natolik malý, že by jeho zapsání bylo nepřesné. Výsledek je nenormalizované číslo.
- Nepřesnost – pokud číslo musí být zaokrouhleno. Výsledek je závislý na zaokrouhlování.

4 VHDL

Jazyk VHDL slouží pro návrh a simulace číslicových obvodů a systémů. Samotný název je složen z HDL (Hardware Description Language – jazyk pro popis hardwaru) a V je zkrácení VHSIC (Very High Speed Integrated Circuits – velmi rychlé integrované obvody). [3]

Tento jazyk je vhodný jak pro simulaci programovatelných hradlových polích jako je například FPGA, tak je zároveň vhodný i pro jejich popis. Výhodou jazyka VHDL je, že není závislý na komponentě, na které bude použit. VHDL je jedním z nejvíce používaných jazyků pro popis hardwarových struktur hradlových polí. Umožnuje návrhy jak kombinačních, tak i sekvenčních struktur. Mezi jeho hlavní výhody patří především univerzálnost jeho použití. Samotný program lze naprogramovat v jazyce VHDL a takovýto kód lze pak použít na téměř jakémkoliv hradlovém poli (Altera, Xilinx, apod). Jediné, co je důležité, je použít správný kompilační program pro převod na formu srozumitelnou danému výrobci. V našem případě se jedná o Altera Quartus II.

Další výhodou toho jazyka je možnost dělit program do menších logických částí, které na sebe navazují. Tímto je docílena lepší přehlednost kódu a také jeho možná modularita. Tyto jednotlivé bloky jsou pospojovány pomocí signálů, což je určitá forma klasických elektrických signálů. Z toho si tak může i samotný návrhář programů udělat obraz o tom, jak to bude pravděpodobně vypadat i fyzicky.

Základní konstrukce jazyka

- **Entita** – definuje rozhraní dané části programu, určují se zde pouze vstupy a výstupy (porty). Každá entita musí mít alespoň jednu architekturu.
- **Architektura** – zde je popsáno samotné chování entity. Určuje, co probíhá mezi jednotlivými porty dané entity. Dále je ještě dělena na deklarační část a samotnou výkonnou část.

4.1 Entita

Jedná se o jednu ze základních komponent každého bloku programu, který je napsán v jazyce VHDL. Zde se definují vstupní a výstupní signály popisovaného bloku, typy definovaných signálu a případně další parametry. Tuto část si lze představit jako rozhraní daného modulu. Pokud je to popis hierarchicky nejvýše postavené komponenty, pak to bude skutečné fyzické rozhraní, na které budou připojeny fyzicky signály. Naopak pokud je to nějaký vnitřní blok, slouží toto pouze pro připojení k jeho nadřazenému modulu. [3]

Porty

Porty jsou části komponenty Entita, slouží pro popis vnějších signálů entity a umožňují tak komunikaci s dalšími bloky programu. Jsou základním kamenem rozhraní entity. Skládají se ze dvou částí. První je jméno daného portu, toto jméno může mít libovolný název (skládající se z písmen, čísel a podtržítek). Samozřejmostí je zákaz používání takových názvů, které jsou vyčleněny pro klíčová slova jazyk VHDL. Druhá část je mód daného portu. Zde se jedná o určení toku dat v daném portu. Například pokud se jedná o vstup, mód bude IN. Další módy jsou OUT, BUFFER, INOUT a LINKAGE. Nejpoužívanějším módem je datový typ definovaný standardem IEEE 1164. Jedná se o multihodnotovou logiku std_logic.

Generic

Generic je méně používaná položka entity, která je obdobou portů. Tato položka ale nepředstavuje žádný signál a používá se pouze jako parametr. Slouží především pro lepší modulárnost daného bloku.

4.2 Architektura

Architektura je druhá povinná komponenta popisující chování entity. V této části se nachází algoritmus, jenž je vykonáván při zavolání daného modulu. Skládá se ze dvou částí. První je deklarační část, ve které jsou deklarovány pomocné proměnné, signály a konstanty. Druhá část je příkazová a je v ní napsána výkonná část kódu. [3]

Operátory

Jedním ze základů každého jazyka jsou jeho operátory. V jazyce VHDL nalezeme sedm druhů operátorů. Jsou to binární, slučující, relační, operátory posuvu, unární znaménkové operátory, operátory násobení a smíšené operátory. Díky těmto operátorům můžeme popisovat vztahy a funkce jednotlivých signálů či proměnných.

Binární operátory: NOT, AND, NAND, OR, XOR, XNOR

Slučující operátory: +, -, &

Relační operátory: =, =/, >, <, =>, <=

Operátory posuvu: SLL, SRL (posuv vlevo a vpravo logický), SLA, SRA (posuv vlevo a vpravo aritmetický), ROL, ROR (rotace vlevo a vpravo)

Unární znaménkové operátory: +, -

Operátory násobení: *, /, mod, rem

Smíšené operátory: not, abs, **

5 Řešení FPU

Nadcházející kapitola se bude věnovat samotné implementaci kódu v jazyce VHDL. Avšak nebude se jednat o strohé popsání jednotlivých příkazů, ale o jakýsi volný překlad algoritmu do srozumitelné řeči.

5.1 Jádro programu

Nejdůležitější části celého programu je samozřejmě jeho jádro. Zde probíhají zásadní rozhodnutí o volbě operace, zavolání správného podprogramu atd. Důležitou vlastností jádra je také, že funguje jako rozhraní, slouží tedy pro komunikaci s okolním světem, ať už se jedná o vstup nebo výstup.

Popis jádra:

V úvodu popisu jádra jsou do programu importovány knihovny, konkrétně se jedná o pět knihoven. První tři jsou IEEE knihovny - tyto knihovny slouží pro práci s danými logikami a velmi ulehčují práci. O těchto knihovnách bylo více řečeno výše a jejich detailní popis není tedy na místě. Další dvě knihovny jsou už ovšem speciálně vytvořeny právě pro tuto práci. Jedná se o pomocné knihovny, ve kterých jsou nadefinována data pro práci s čísly v pohyblivé řádové čárce. Více o těchto knihovnách si povíme po popisu jádra.

Nyní přejděme k popisu rozhraní jádra. Rozhraní jádra se skládá ze vstupů a výstupů. Nejdůležitějšími vstupy jsou dva operandy, nad kterými v pozdější fázi probíhá výpočet. Jedná se tedy o vstup operandu A a operandu B. Dalším vstupem je operace - tato proměnná určuje, jaká operace se má vykonat. Tyto tři vstupy jsou naprostým základem a nelze bez nich uskutečnit žádnou operaci. Jedinou výjimku tvoří speciální operace, které pracují s jediným operandem. Jako další vstup je proměnná, která už je speciálně pro práci s pohyblivou řádovou čárkou. Jedná se o určení módu zaokrouhlování, který dá informace programu o tom, kterou metodu má zvolit, pokud dojde k zaokrouhlování. Posledním vstupem je řídicí signál, který dává na vědomí programu, aby začal pracovat. Jedná se o signál start. Základním výstupem je číslo udávající výslednou hodnotu. Speciálními výstupy, které opět slouží pro práci s pohyblivou řádovou čárkou, jsou výjimky. Tyto výjimky byly popsány výše v kapitole Standard IEEE 754. Zde je jen potřeba říci, že výjimky jsou na výstupu pouze za předpokladu, že byly během výpočtu vyvolány. Poslední výstup je opět řídící, tento výstup dává na vědomí, že výpočet je kompletní a program je připraven pro další práci. Jedná se o signál ready.

V předchozím odstavci jsme si popsali entitu jádra programu neboli rozhraní. Nyní si řekneme, jak funguje architektura našeho jádra. Nejdříve je potřeba si vytvořit

pomocné proměnné. To je jedna z hlavních náplní práce jádra. Jelikož nelze pracovat během výpočtu přímo se vstupy, je potřeba přiřadit každému vstupu pracovní proměnou. Dále je potřeba si nadefinovat proměnné nové. Jedná se především o proměnné jednotlivých výpočetních operací. Další proměnné jsou pro správnou funkci samotného jádra. Takto vytvořené proměnné se nazývají pracovní. Nyní k samotné práci jádra. V první části jsou přiřazeny vstupní hodnoty, do právě vytvořených proměnných, které jsou dále zpracovávány v podružných modulech jednotlivých operací, o nich více dále. Během celé této operace dohlíží stavový automat nad tím, zdali je počítáno (probíhá výpočet) nebo nikoliv a udává tak, kdy je program schopen dalšího výpočtu. Dalším důležitým činitelem je přepínač. Ten na základě zadané proměnné operace určí, které hodnoty půjdou do výstupních lokálních proměnných. Úplně na závěr jsou tyto proměnné přiřazeny do skutečných výstupních proměnných a tak vlastně vyslány na výstup.

Pomocné knihovny:

Jak bylo již řečeno dříve, program využívá dvě knihovny speciálně vytvořené pro tento program. Prvních z nich nese název pomocná. A tato knihovna neslouží k ničemu jinému než k tomu, že definuje proměnné, které pak jsou používány v jednotlivých komponentách. Důvod vytvoření takovéto knihovny je, aby jednotlivé kódy operací byly co nejvíce přehledné. Slouží tedy spíše pro programátora nežli pro samotný program. Druhou pomocnou knihovnou je knihovna jménem konstanty. Zde jsou nadefinovány rozměry jednotlivých proměnných. Především se jedná o informace sloužící pro práci s pohyblivou řádovou čárkou. Změnou těchto konstant lze dosáhnout, že program umožní práci s různými typy čísel v pohyblivé řádové čárce, co se do velikosti týče. V knihovně jsou poté ještě dvě speciální funkce, které slouží pro výpočet nul v daném čísle, ať už zprava nebo zleva.

Na jádro jsou poté napojeny jednotlivé operace, které provedou nad zadanými operandy danou operaci a vrátí výsledek zpátky do jádra. Více o jednotlivých operacích dále.

5.2 Sčítání a odčítání

Přesto, že jsou sčítání a odčítání dvě rozdílné operace, v programu jsou brána jako jedna. Důvod je jednoduchý, práce s čísly v pohyblivé řádové čárce umožnuje ukládat i čísla záporná. Pak tedy není rozdíl, pokud sčítám kladné číslo se záporným, nebo odečítám kladné číslo od kladného. Proto jsou obě operace sloučené do jedné a je potřeba přenášet i typ dané operace do samotného bloku operace.

Samotná operace se skládá ze tří bloků. V prvním bloku se provede úprava vstupních operandů a to tak, že se rozdělí na menší logické části, se kterými se lépe pracuje. Druhá část provede samotnou operaci a třetí část slouží k normalizování výsledku tak, aby odpovídal normě. Ve všech třech blocích jsou importovány čtyři knihovny - tři standardní knihovny pro práci s STD logikou a námi vytvořená knihovna Konstanty.

Popis algoritmu:

Před-normalizace

Jako první je definováno rozhraní tohoto bloku. Do bloku vstupují oba operandy a samozřejmě hodinový takt. Výstupem je pak upravená mantisa prvního a druhého operandu. Posledním výstupem je výsledný exponent.

V architektuře jsou nejprve nadefinovány pomocné signály, které slouží pro práci v tomto bloku. Ve funkční části architektury přiřadíme vstupní hodnoty do signálu námi právě vytvořenými. Následně určíme výstupní exponent, to provedeme poměřením exponentů jednotlivých operandů a vybereme ten větší. Na tomto základu proběhnou pomocné úpravy, které upravují redukované části mantisy. Naposled se provedou posuny na základě rozdílu exponentů. Pokud je potřeba posunou se nuly zprava. Výsledné mantisy obou operandů jsou poslány na výstup.

Sčítání/odčítání

Opět je nejprve nadefinováno rozhraní daného bloku. Vstupem je zde typ probíhané operace a upravené mantisy obou operandů z předcházejícího bloku. Dále pak zde vstupují znaménka obou operandů z hlavní části programu a samozřejmě hodinový signál. Výstupem je vypočtená mantisa a vypočtené konečné znaménko.

Prvním krokem při návrhu architektury je opět nadefinování pomocných signálů. Ve výpočetní části si přiřadíme vstupní hodnoty námi vytvořeným signálům. Dále proběhne zjištění, zdali budeme sčítat nebo odčítat. To záleží na znaménkách operandů a zvolené operaci. Následně je vypočteno výsledné znaménko výsledku. Poslední fáze výpočtu je sečtení či odečtení obou mantis. Takto vypočtená mantisa a znaménko se pošlou do posledního bloku.

Po-normalizace

Do rozhraní vstupují základní vstupní operandy, typ operace a způsob zaokrouhlování z jádra programu. Dále pak vypočtená mantise, exponent a znaménko

z předchozího bloku a samozřejmě hodinový signál. Výstupem je vypočtené číslo a informace, zdali došlo k nepřesnosti či nikoliv.

Na začátku architektury si opět připravíme podpůrné signály nezbytné pro provádění výpočtů. Následně přiřadíme vstupní hodnoty do vytvořených signálů. Poté zjistíme výsledný exponent a potřebný posun. Ten následně provedeme a, pokud je třeba, zaokrouhlíme podle zvolené metody. Následně opět provedeme posun, pokud je třeba. Naposled zjistíme, zdali při normalizaci nedošlo k nějaké výjimce. Pokud ano, provedeme její ošetření. Výslednou hodnotu spolu s informací o přesnosti předáme na výstup.

5.3 Násobení

Pro násobení jsme použili sériový způsob. Celá operace je stejně jako předchozí opět rozdělena do tří samostatných bloků. V prvním bloku dochází k rozdělení operandů na menší části a k příprava na výpočet. Ve druhé části dochází k samotnému výpočtu a ve třetí části dochází k normalizaci výsledku a jeho poslání zpět do jádra programu. Všechny tři části mají importované čtyři knihovny, tři pro STD logiku a speciální knihovnu Konstanty.

Popis algoritmu:

Před-normalizace

Rozhraní tvoří šest signálů. Do bloku vstupují dva operandy a samozřejmě hodinový takt. Výstupem je pak upravený exponent a dvě upravené mantisy jednotlivých operandů.

V architektuře dochází k nadefinování pomocných signálů pro pomoc při výpočtech. Ve funkční části architektury dojde k přiřazení hodnot exponentů a mantis do proměnných. Následně jsou normalizovány mantisy a vytvořen výsledný exponent. Obojí je posláno do další části.

Násobení

V části entity je opět nejprve potřeba nadefinovat rozhraní pro daný blok. Do bloku vstupují mantisy obou operandů, znaménka obou operandů a pro řízení hodinový signál a signál start. Na výstup je pak poslána výsledná mantisa, výsledné znaménko a signál udávající dokončení výpočtu.

Na začátku architektury dojde opět k nadefinování potřebných proměnných. Dále jsou přiřazeny hodnoty do právě vytvořených signálů. Pomocí operace xor mezi vstupujícími znaménky je vypočteno výsledné znaménko. Pro kontrolu výpočtu je vytvořen stavový automat, který dohlíží na obsazení operace násobení. Dále je sériově vypočtena mantisa. Obě vypočtené hodnoty jsou následně předány na výstup spolu se signálem udávající konec výpočtu.

Po-normalizace

Jako první krok je opět nadefinování rozhraní. Do bloku vstupují oba operandy. Dalším vstupem je vypočtený exponent z prvního bloku operace. Dále to je vypočítaná mantisa a znaménko z předcházejícího bloku. Posledním vstupem je pak režim zaokrouhlování. Výstupem je výsledné číslo a informace o přesnosti výpočtu.

V architektuře jsou vytvořeny pomocné signály. Následně jsou přeneseny hodnoty ze vstupních signálů do pomocných signálů a zjistíme výsledný exponent. Dále vypočteme posun a následně provedeme. V další fázi provedeme případné zaokrouhlování. Naposled zjistíme speciální vlastnosti výsledku, zdali se nejedná o nekonečno či jinou nestandardní hodnotu. Poté jsou ošetřeny případné výjimky a výsledné hodnoty poslány na výstup.

5.4 Dělení

Operace dělení je zde vytvořena pomocí sériového způsobu. Operace dělení je opět rozdělena do třech funkčních částí. První část slouží pro rozložení vstupních operandů na menší části, s kterými se lépe provádí výpočet. Ve druhé části poté dochází k samotnému výpočtu. Třetí část slouží k složení jednotlivých částí vypočítaného čísla a ošetření výjimek. Opět jsou zde importovány tři knihovny pro STD logiku a knihovna Konstanty. Importovány jsou do všech třech komponent.

Popis algoritmu:

Před-normalizace

Rozhraní je definováno pomoci třech vstupních a třech výstupních signálů. Vstup zde tvoří oba operandy a samozřejmě hodinový signál. Výstupem je zde exponent a mantisa dělence a dělitele.

Prvním krokem architektury je definování pomocných proměnných. Dále jsou do těchto pracovních proměnných předány vstupní hodnoty. Dalším krok je zjištění počtu nul dělitele a dělence. Následně proběhne posun vlevo a výpočet mantis na základě počtu nul. Naposled je zjištěn předběžný exponent a výsledky jsou odeslány na výstup.

Dělení

V části entity si nejprve nadefinujeme rozhraní. Do bloku vstupuje znaménko dělitele a dělence. Dále vstupují vypočtené mantisy z předchozího bloku, hodinový signál a řídicí signál start. Na výstupu je výsledná mantisa, znaménko a řídicí signál udávající možnost dalšího výpočtu. Je zde také informace o tom, zdali bylo děleno nulou.

V architektuře opět dojde k definování pomocných proměnných. Ve funkční části jsou přiděleny hodnoty do pomocných signálů. Následně je vypočteno výsledné znaménko pomoci operace xor mezi dělitelem a dělencem. Dále proběhne zjištění, zdali bylo děleno nulou či nikoliv. Stejně jako u násobení je zde stavový automat sloužící k řízení operace. Naposled je sériově vypočtena výsledná mantisa a výsledné hodnoty přiřazeny na výstup.

Po-normalizace

Jako první je vytvořeno rozhraní. Do bloku vstupují oba operandy a vypočtená mantisa s exponentem a znaménkem. Dále do ní vstupuje z jádra typ zaokrouhlování. Výstupem je pak výsledné číslo a údaj o nepřesnosti.

Na začátku architektury opět provedeme definici pomocných signálů. Dále přiřadíme hodnoty do těchto signálů. Poté zjistíme v operandech exponent a provedeme případné posuvy vpravo či vlevo. Dále provedeme úpravy u exponentů a proběhne další posun mantisy. Následně provedeme případné zaokrouhlení na základě předchozích výpočtů. Pro výstup si zjistíme, zdali některé číslo není jedno ze speciálních (například nekonečno apod.). Dále jsou ošetřeny případné výjimky. Na úplný závěr je složeno výsledné číslo z předcházejících výpočtů a posláno na výstup, tedy do jádra.

5.5 Druhá odmocnina

Operace druhá odmocnina je počítána za pomocí iterativního algoritmu, ve kterém je zapotřebí tolik opakování, kolik je přesnost požadovaného výsledku. Tento algoritmus nepotřebuje žádné dělení ani násobení, neboť je vše provedeno za pomocí posunů vlevo či vpravo. Operace se opět skládá ze tří částí. První část slouží pro rozložení vstupního operandu do menších částí. V druhé části dojde k samotnému výpočtu druhé odmocniny. Poslední část sloučí dohromady jednotlivé výsledky a na výstup dá normalizovaný výsledek.

Popis algoritmu:

Před-normalizace

V entitě dojde nejprve k definování rozhraní. Do bloku vstupuje operand, a jelikož se jedná o operaci druhá odmocnina, nevstupuje do ní žádný druhý operand jako u předchozích operací. Dále do ní vstupuje hodinový signál. Na výstupu je vypočítaný exponent a mantisa.

V části architektura nejprve nadefinujeme pomocné signály, které následně naplníme vstupními hodnotami. Dále si zjistíme, zdali je operand nenormalizovaný a zjistíme si počet nul. Následně proběhne úprava exponentu. V dalším kroku si pomocí levého posunu, přidáním nul, zjistíme výstupovou mantisu. Nakonec výsledky pošleme na výstup.

Druhá odmocnina

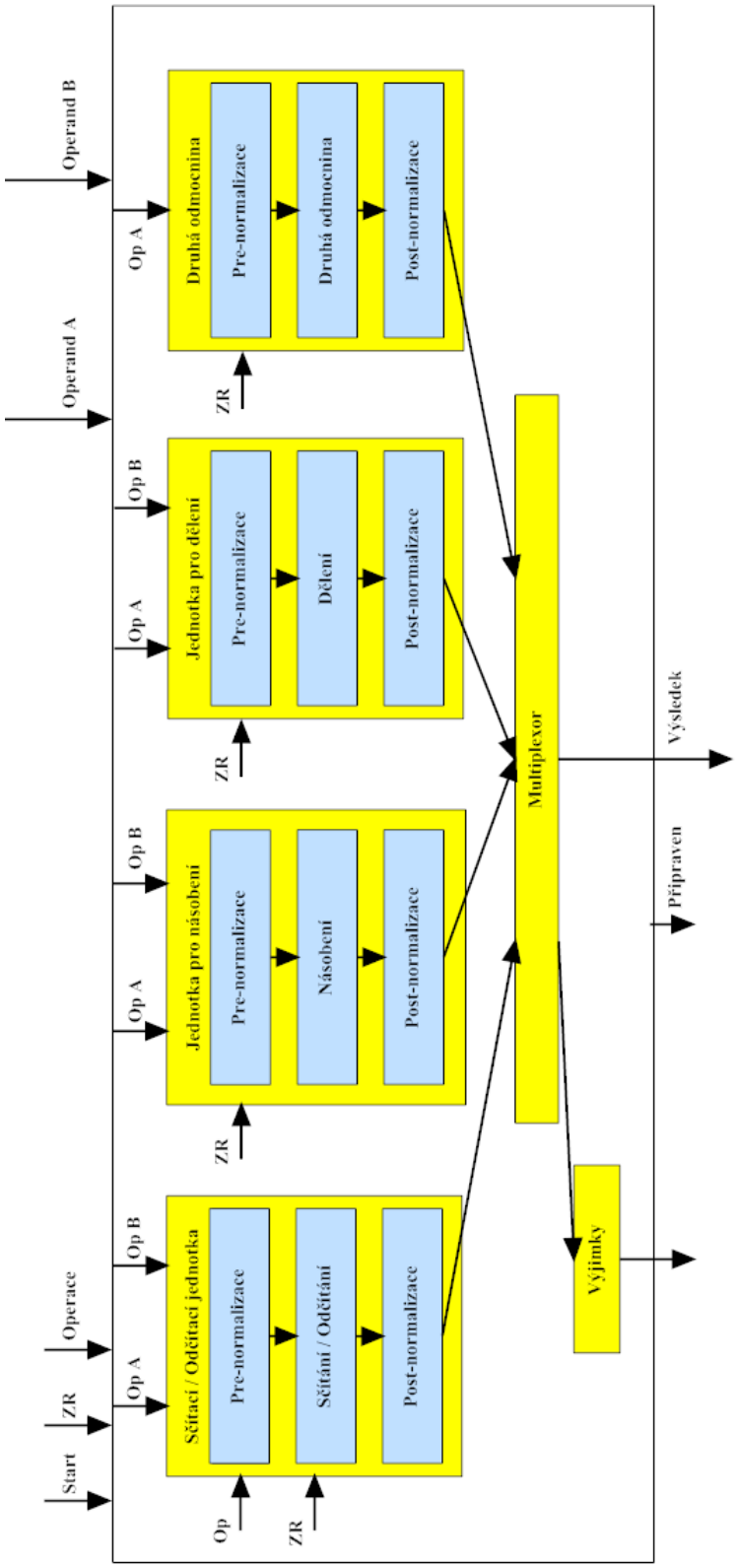
Na začátku bloku si opět nadefinujeme rozhraní. Do bloku vstupuje mantisa, hodinový signál a řídicí signál start. Na výstupu je výsledné číslo a dva informační signály – Jeden udávají informaci o možnosti dalšího počítání a druhý o přesnosti výpočtu.

Na začátku architektury dojde k nadefinování pomocných signálů. Ve funkční části dojde k přiřazení hodnot do právě vytvořených signálů. Celý proces výpočtu řídí stavový automat, který na základě hodinového cyklu a počtu iterací udává dobu výpočtu. Následně dochází k samotnému výpočtu, kdy je poměřováno, zdali mantisa operandu je větší nežli předběžný výsledek výpočtu v kvadrátech. Na tomto výsledku dojde k dalšímu výpočtu a vše je opakováno, nežli dojde k dostatečné přesnosti. Jakmile je přesnost splněna, je na výstup poslán vypočtený výsledek.

Po-normalizace

Na začátku bloku je nadefinováno rozhraní v entitě. Do bloku vstupuje operand, vypočtená mantisa a exponent. Dále vstupuje hodinový signál, signál udávající přesnost a typ zaokrouhlování. Na výstupu je výsledné číslo a údaj o přesnosti výsledku.

Další částí bloku je architektura, ve které si nejprve nadefinujeme pomocné signály, které jsou nezbytné pro výpočty v tomto bloku. Ve výkonné části architektury si nejprve přiřadíme hodnoty do právě vytvořených signálů. Následně provedeme zaokrouhlení pomocí zvolené zaokrouhlovací metody. Poté složíme jednotlivé části dohromady. Naposled zjistíme, zdali výsledek není nějaké speciální číslo a údaj o přesnosti výsledku. Zjištěné hodnoty odešleme na výstup do jádra programu.



Obrázek 2 - diagram programu

6 Simulace v Altera Quartus II

Pro ověření funkčnosti programu je důležitá nejenom jeho kompilace, ale i odsimulování funkcí. Samotná kompilace poslouží spíše k ověření správnosti napsání programu, co se týče syntaxe. Pomocí simulace lze následně otestovat správnost programu a jeho jednotlivých algoritmů. V této kapitole si tedy ověříme, zdali naprogramované jádro FPU a jeho jednotlivé operace dodávají správné výsledky. Psaní celého kódu a jeho následná kompilace probíhala v programu Altera Quartus II. Tento program má také možnost odsimulování funkčnosti napsaného kódu. Tohoto využijeme a provedeme ověření našeho programu v simulační části programu Altera Quartus II.

6.1 Altera Quartus II

Quartus je vývojový systém od firmy Altera. Umožnuje syntézu a analýzu kódu jazyka pro popis hardwaru. Umožnuje návrhářům kompilovat jejich projekty, vytvářet jejich časové analýzy a prohlížet diagramy těchto projektů. Dále lze otestovat různé vstupy do navržených bloků a jejich reakce. To vše na předem určených zařízeních, které si programátor sám vybere.

Altera Quartus II Web edice

Firma Altera umožnuje používání jejího softwaru zcela zdarma. Konkrétně se jedná o Altera Quartus II Web edice. Tu lze zdarma stáhnout z jejich webových stránek a používat následně bez omezení. Právě s touto verzí pracujeme. Konkrétně se jedná o verzi 8.11.

Použité nástroje:

- Nástroj pro implementaci VHDL
- Simulace implementovaného návrhu

Nástroj pro implementaci VHDL – zde byl napsán kompletní kód všech komponentů našeho programu. Program umožnuje rozdělení celého kódu do jednotlivých částí a jejich následné propojení pomocí rozhraní jednotlivých bloků. Dále zde byla provedena kompilace a tedy ověření správnosti napsání kódu v jazyce VHDL.

Simulace implementovaného návrhu – zde proběhlo ověření funkčnosti jádra programu a také jednotlivých operací.

6.2 Simulace operací

Postup při simulování jednotlivých operací

V programu Quartus si otevřeme projekt s již hotovým programem a založíme nový soubor pro simulaci implementovaného návrhu. Následně je potřeba přidat vstupní a výstupní signálu programu. To spočívá v přidání vstupních proměnných, jako jsou operandy, nad kterými proběhne výpočet, typ operace, mód zaokrouhlování, hodinový takt a řídicí signál udávající začátek výpočtu. Dále musíme přidat výstupní proměnné. Zde se jedná o výsledek dané operace, řídicí signál udávající možnost dalšího výpočtu a speciální výstupy ošetřující výjimky a podobně. Jelikož program počítá vždy jen jednu operaci v daný moment, je vhodné pro každou operaci provést samotnou simulaci. V každé této simulaci si nastavíme na vstupy různé hodnoty.

Nastavované hodnoty na vstupu:

- Hodinový takt (stejný pro všechny operace)
- Operand A
- Operand B
- Výpočetní operace
- Režim zaokrouhlování
- Signál start

Po nastavení jednotlivých hodnot v programu se spustí samotná simulace. Následně se zobrazí zjištěné hodnoty ve zprávě o simulaci. Zde se dozvíme, jaké hodnoty dostáváme na vybraných výstupech v průběhu času. Z toho lze odečíst výslednou hodnotu a případné výjimky.

Legenda k simulaci výpočtů

clk_i	hodinový signál
start_i	řídicí signál start
opa_i	operand A
opb_I	operand B
fpu_op_i	operace
rmode_i	režim zaokrouhlování
output_o	výsledek
ready_o	řídicí signál připraven

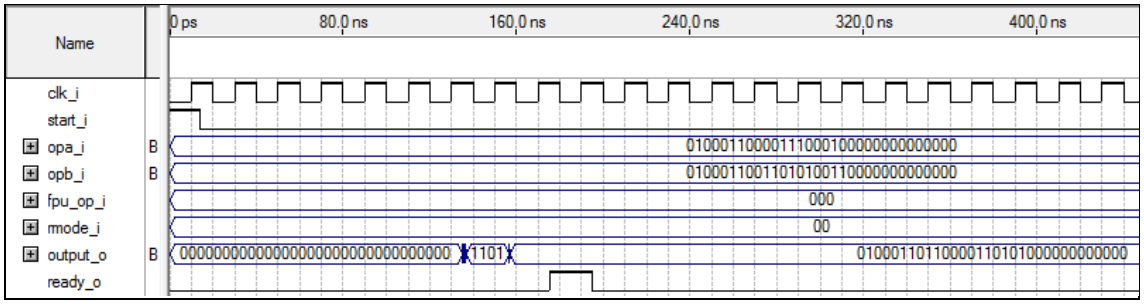
Sčítání

Při operaci sčítání sečteme dva vstupující operandy a bude zkoumat výsledné hodnoty, které se zobrazí na výstupu. Porovnáme tedy výsledek s očekávaným výsledkem (správným).

Zadané hodnoty:

	Decimální zobrazení	Binární zobrazení	Jiné
Operand A	10 000	01000110 00011100 01000000 00000000	
Operand B	15 000	01000110 01101010 01100000 00000000	
Operace	0	000	sčítání
Očekávaný výsledek	25 000	01000110 11000011 01010000 00000000	

Simulace:



Obrázek 3 - simulace sčítání

Výsledné hodnoty:

	Decimální zobrazení	Binární zobrazení
Výsledek	25000	01000110 11000011 01010000 00000000

Závěr:

Pomocí simulace sčítání byla sečtena dvě čísla. Výsledná hodnota odpovídala předpokládanému výsledku. Ze zjištěných hodnot lze vyvodit, že operace sčítání pracuje správně.

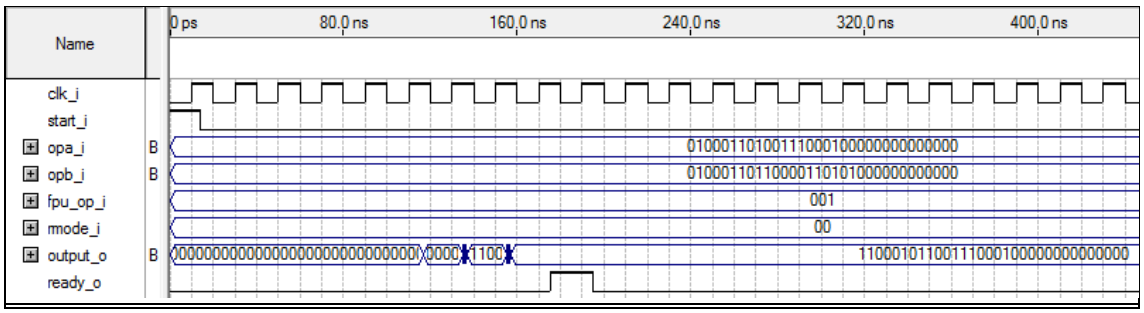
Odčítání

V operaci odčítání odečteme od operandu A operand B. Následně ověříme hodnoty na výstupu s předpokládanými hodnotami.

Zadané hodnoty:

	Decimální zobrazení	Binární zobrazení	Jiné
Operand A	20 000	01000110 10011100 01000000 00000000	
Operand B	25 000	01000110 11000011 01010000 00000000	
Operace	1	001	odčítání
Očekávaný výsledek	-5000	11000101 10011100 01000000 00000000	

Simulace:



Obrázek 4 - simulace odčítání

Výsledné hodnoty:

	Decimální zobrazení	Binární zobrazení
Výsledek	-5000	11000101 10011100 01000000 00000000

Závěr:

Simulace odčítání proběhla v pořádku a výsledné hodnoty na výstupu odpovídají předpokládanému výsledku. Operace odčítání pracuje správně.

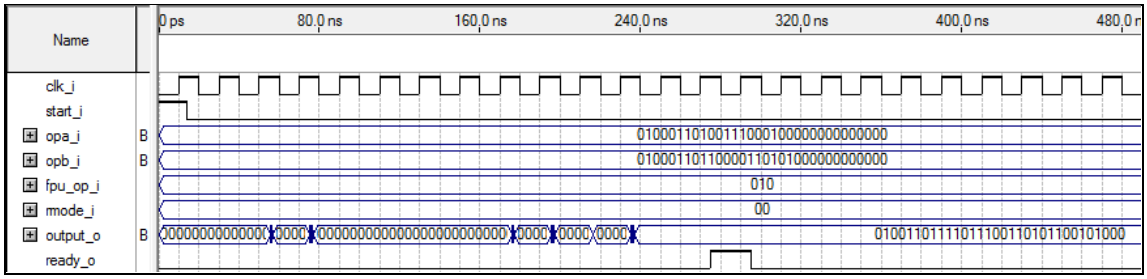
Násobení

V operaci násobení mezi sebou vynásobíme dva vstupující operandy. Po odsimulování operace zkontrolujeme výsledné hodnoty s předpokládaným výsledkem.

Zadané hodnoty:

	Decimální zobrazení	Binární zobrazení	Jiné
Operand A	20 000	01000110 10011100 01000000 00000000	
Operand B	25 000	01000110 11000011 01010000 00000000	
Operace	2	010	násobení
Očekávaný výsledek	500 000 000	01001101 11101110 01101011 00101000	

Simulace:



Obrázek 5 - simulace násobení

Výsledné hodnoty:

	Decimální zobrazení	Binární zobrazení
Výsledek	500 000 000	01001101 11101110 01101011 00101000

Závěr:

Výsledné hodnoty simulace násobení odpovídají očekávanému výsledku. Lze tedy označit operaci násobení za fungující správně.

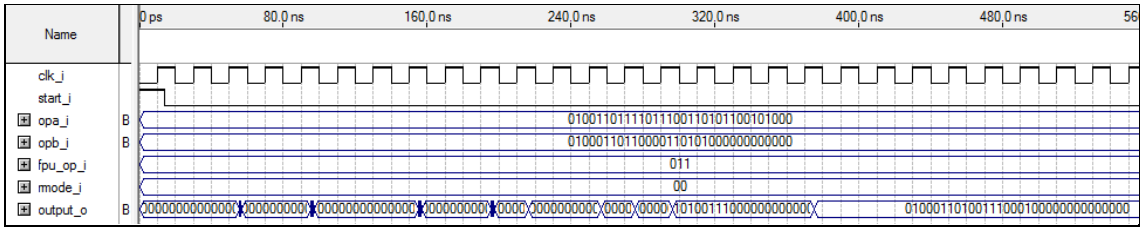
Dělení

Při simulaci operace dělení budeme dělit operand A operandem B. Výslednou hodnotu porovnáme s předpokládaným výsledkem.

Zadané hodnoty:

	Decimální zobrazení	Binární zobrazení	Jiné
Operand A	500 000 000	01001101 11101110 01101011 00101000	
Operand B	25 000	01000110 11000011 01010000 00000000	
Operace	3	011	dělení
Očekávaný výsledek	20 000	01000110 10011100 01000000 00000000	

Simulace:



Obrázek 6 - simulace dělení

Výsledné hodnoty:

	Decimální zobrazení	Binární zobrazení
Výsledek	20 000	01000110 10011100 01000000 00000000

Závěr:

Výstupní hodnota se rovná očekávanému výsledku. Z toho lze odvodit, že operace dělení pracuje správně.

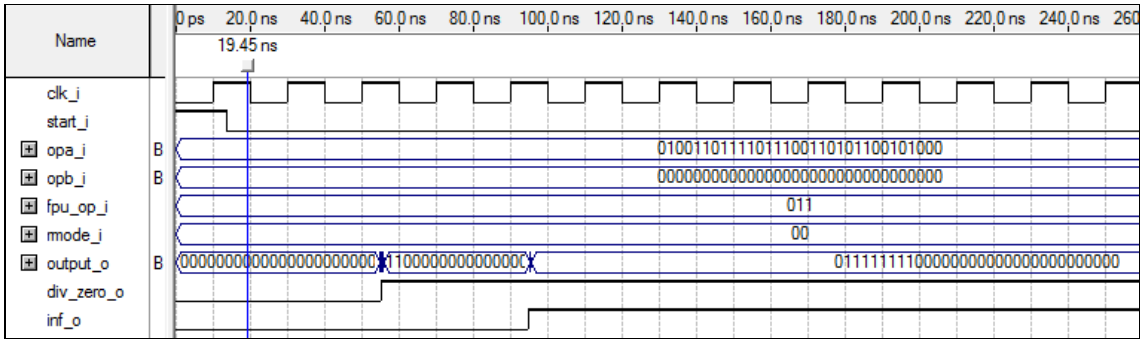
Dělení nulou

Nejčastější chybou při operaci dělení je dělení nulou. Na následující simulaci je zobrazena práce programu právě při dělení nulovým číslem.

Zadané hodnoty:

	Decimální zobrazení	Binární zobrazení	Jiné
Operand A	500 000 000	01001101 11101110 01101011 00101000	
Operand B	0	00000000 00000000 01010000 00000000	
Operace	3	011	dělení
Očekávaný výsledek	∞	01111111 10000000 00000000 00000000	nekonečno

Simulace:



Obrázek 7 – simulace dělení nulou

Výsledné hodnoty:

	Decimální zobrazení	Binární zobrazení
Výsledek	∞	01111111 10000000 00000000 00000000

Závěr:

Výstupní hodnota odpovídá předpokládanému výsledku, operace tedy pracuje správně. Ze simulace také lze vyčíst, že na výstupu dostaneme informaci o tom, že bylo děleno nulou a výsledek je nekonečno.

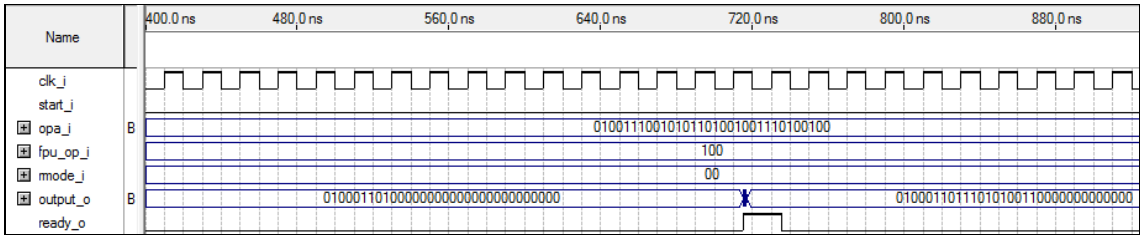
Druhá odmocnina

V této operaci použijeme pouze jeden operand. Na tento operand aplikujeme operaci druhá odmocnina. Výslednou hodnotu porovnáme s předpokládaným výsledkem.

Zadané hodnoty:

	Decimální zobrazení	Binární zobrazení	Jiné
Operand A	900 000 000	01001110 01010110 10010011 10100100	
Operace	4	100	odmocnina
Očekávaný výsledek	30 000	01000110 11101010 01100000 00000000	

Simulace:



Obrázek 8 - simulace druhé odmocniny

Výsledné hodnoty:

	Decimální zobrazení	Binární zobrazení
Výsledek	30 000	01000110 11101010 01100000 00000000

Závěr:

Výsledná hodnota na výstupu odpovídá předpokládanému (správnému) výsledku. Operace druhá odmocnina tedy pracuje správně.

7 Porovnání výsledků

V této kapitole se pokusíme porovnat výsledky vytvořené jednotky s již hotovými jednotkami pro práci s pohyblivou řádovou čárkou. Jelikož ale je velmi obtížné získat informace o specifických vlastnostech cizích jednotek, zaměříme se především na porovnání dostupných údajů. Pokusíme se tedy získat informace o jiných funkčních jednotkách, které mají stejné nebo alespoň obdobné parametry. Tyto informace následně porovnáme s údaji naší jednotky a následně vyhodnotíme. Na konci bychom měli přibližně určit výhody a nevýhody vytvořené jednotky.

Parametry naší jednotky

	Počet logických elementů
Jádro	326
Sčítání a odčítání	684
Násobení	1530
Dělení	928
Druhá odmocnina	919
Celkem	4387

Tabulka 1 – počet logických elementů

Počet registrů	1402
Počet pinů	112
Maximální frekvence	92,64 MHz

Tabulka 2 – parametry

Operace	Počet cyklů
Sčítání	7
Odčítání	7
Násobení	12
Dělení	35
Druhá odmocnina	35

Tabulka 3 – počet cyklů

Parametry jiných FPU

	GRFPU	ARM VFP 11	ARM FP9-S	MEIKO
Počet log. elementů	8500	8765	9250	2800
Max. frekvence (MHz)	65	80	35	35
Sčítání (cykly)	3	3	4	8
Odčítání	3	3	4	8
Násobení	5	10	5	10
Dělení	16	33	31	50
Druhá odmocnina	24	38	X	X

Porovnání

Na závěr provedeme porovnání mezi vytvořenou jednotkou a již hotovými jednotkami, které měly dostupné informace o jejich výkonu. Na začátek je potřeba říci že se jedná o profesionální řešení jednotek pro práci s pohyblivou řádovou čárkou neboli koprocesory.

Co se týká počtu logických elementů je námi vytvořené řešení na velmi malém množství a tudíž oproti ostatním jednotkám (kromě MEIKO) zabere velice málo místa. Maximální frekvence je oproti ostatním jednotkám o něco málo vyšší, ale výhoda jako taková to může být pouze v některých případech. U operace sčítání a odčítání naše jednotka dosahuje až dvojnásobného počtu cyklů pro získání výsledku. V tomto ohledu předčí pouze řešení MEIKO. Operace násobení se pohybuje lehce nad ostatními řešeními, co se týká počtů cyklů. Delší operace dělení využívá průměrného počtu cyklů na výpočet výsledku. Operace druhá odmocnina se poté umístila mezi dvěma řešeními. Zbylé dvě jednotky tuto operaci neimplementovaly.

Výsledek porovnání

Naše jednotka má především výhodu v nízkém počtu logický elementů a vyšší maximální frekvenci. Nevýhodou je pak větší počet cyklů pro získání výsledků u některých operací. Jednotka pro práci s pohyblivou řádovou čárkou, kterou jsme vytvořili, nalezne tedy především uplatnění tam, kde je velmi důležité ušetřit co nejvíce fyzického místa výměnou za pomalejší výpočet. Velikost se také kladně projeví na nákladech za pořízení jednotky.

Závěr

Na začátku práce jsme se seznámili s problematikou aritmeticko logických jednotek a především jejich odnože, kterou jsou jednotky pracující s čísly s pohyblivou řádovou čárkou. Dále jsme popsali, co to jsou čísla s pohyblivou řádovou čárkou a jak probíhá jejich zápis. V další kapitole jsme si řekli o nejpoužívanějším standardu pro práci s čísly s pohyblivou řádovou čárkou, o jeho možnostech použití a jednotlivých typech. Tímto jsme splnili první bod ze zadání.

V další kapitole jsme se seznámili s jazykem pro popis hardwaru. Námi vybraný jazyk VHDL jsme poté následně použili pro návrh jednotky pro práci s čísly v pohyblivé řádové čárce. Zde jsme navrhli jednotku, která disponuje aritmetickými operacemi pro sčítání, odčítání, násobení a dělení. Jako operaci navíc jsme navrhli jednu z nejvíce používaných operací (mimo základní operace) a to operaci druhá odmocnina. Touto kapitolou jsme splnili požadavky na druhý bod zadání.

Hotový program jsme následně otestovali v simulátoru, jenž je součástí návrhového systému Altera Quartus II. Zde jsme otestovali jednotlivé operace a zjistili, zdali pracují správně, tedy jestli na výstup dávají správné výsledky a případně upozorňují na nastalé výjimky. Všechny navržené operace fungují správně a jsou schopné provozu. Tímto testováním jsme splnili třetí bod zadání.

Na závěr jsme zjistili parametry naší jednotky a porovnali jsme je s jinými již vytvořenými jednotkami pro práci s čísly v pohyblivé řádové čárce, které mají stejné či obdobné vlastnosti jako naše jednotka. Tímto jsme splnili i poslední bod z našeho zadání.

Touto prací vznikla jednotka sloužící pro práci s čísly v pohyblivé řádové čárce, která implementuje základní aritmetické operace. Jádro programu bylo napsáno tak, aby jednotlivé operace byly od něj odděleny. Dále bylo dbáno na to, aby vše bylo roztříděno do jednotlivých bloků, které jsou na sebe logicky navázány. Při psaní bylo dbáno na maximální dodržení standardu IEEE 754. Těmito kroky se vznikl program, jenž je variabilní (jednoduše lze změnit velikost přijímaných operandů) a přehledný (části kódu v blocích). Program je také otevřen dalšímu rozšíření ať už ze strany dalších operací s čísly nebo i další funkcí samotného programu jako přidání dalších zaokrouhlovacích režimů atd. Program může být použit samostatně pouze pro získání výpočtu nebo může být zapojen do většího systému. Zde by sloužil jako blok, který bude pracovat s čísly v pohyblivé řádové čárce.

Seznam použité literatury

[1] IEEE COMPUTER SOCIETY, Microprocessor Standards Committee, INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS a IEEE-SA STANDARDS BOARD. *IEEE standard for floating-point arithmetic*. New York, NY: Institute of Electrical and Electronics Engineers, 2008. ISBN 978-073-8157-535.

[2] MULLER, J. et al. *Handbook of floating-point arithmetic*. Boston: Birkhäuser, c2010, ISBN 08-176-4705-8. Dostupné z: http://www.google.cz/books?id=baFvrIOPvncC&printsec=frontcover&hl=cs#v=onepage&q&f=false

[3] PINKER, Jiří a Martin POUPA. *Číslicové systémy a jazyk VHDL*. 1. vyd. Praha: BEN - technická literatura, 2006, ISBN 80-730-0198-5.

[4] DESCHAMPS, Jean-Pierre, Géry Jean Antoine BIOUL a Gustavo D. SUTTER. *Synthesis of arithmetic circuits*: *FPGA, ASIC, and embedded systems*. Hoboken: John Wiley, 2006, ISBN 0-471-68783-9.

[5] ASHENDEN, Peter J, Géry Jean Antoine BIOUL a Gustavo D SUTTER. *Designer´s guide to VHDL*: *FPGA, ASIC, and embedded systems*. 2nd ed. San Francisco: Morgan Kaufmann, 2002. ISBN 1-55860-674-2. Dostupné z: http://www.google.cz/books?id=BS0npWWfCIcC&printsec=frontcover&hl=cs&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[6] ROTH, Charles H. *Digital systems design using VHDL*. Boston: PWS Publishing Company, c1998, ISBN 0-534-95099-X.

[7] LIPSETT, Roger, Carl F. SCHAEFER a Cary USSERY. *VHDL, hardware description and design*. Boston: Kluwer Academic Publishers, c1989, ISBN 0-7923-9030-X. Dostupné z: http://books.google.cz/books?id=YAAGQx3uSRsC&printsec=frontcover&hl=cs&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[8] OVERTON, Michael L. *Numerical computing with IEEE floating point arithmetic including one theorem, one rule of thumb, and one hundred and one exercises*. Philadelphia: SIAM, 2001. ISBN 0-89871-571-7.

[9] DESCHAMPS, Jean-Pierre, Gustavo D SUTTER a Enrique CANTO. *Guide to FPGA implementation of arithmetic functions*. New York: Springer, 2012. ISBN 978-94-007-2986-5. Dostupné z: http://books.google.cz/books?id=IhDMoFtigvAC&printsec=frontcover&hl=cs&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[10] RUSHTON, Andrew. *VHDL for Logic Synthesis*. 3. ed. Chichester: Wiley, 2011. ISBN 978-0-470-68847-2. Dostupné z: http://books.google.cz/books?id=lC8HKr0e2nwC&printsec=frontcover&hl=cs&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[11] PEDRONI, Volnei A. *Circuit design and simulation with VHDL*. 2nd ed. Cambridge: MIT Press, c2010, ISBN 978-0-262-01433-5.

[12] PELLERIN, David a Douglas TAYLOR. *VHDL made easy!*. Upper Saddle River: Prentice Hall, 1997, ISBN 0-13-650763-8. Dostupné z: http://www.scribd.com/doc/7218135/Vhdl-Made-Easy

[13] ČAMBOR, Michal. *Elementární procesor v aritmetice pevné a pohyblivé řádové čárky*: *Fixed and Floating Point Arithmetic Elementar Processor*. Brno: Vysoké učení technické, Fakulta informačních technologií, 2009. Bakalářská práce. Vysoké učení technické.

[14] CAVANAGH, Joseph J. *Digital design and Verilog HDL fundamentals*. Boca Raton: CRC Press, 2008, ISBN 978-1-4200-7415-4.

[15] CAVANAGH, Joseph J. *VeriLog HDL*: *digital design and modeling*. Boca Raton: CRC Press, c2007, ISBN 1-4200-5154-7.