České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra teorie obvodů

Diplomová práce

Aritmetické operátory v pohyblivé řádové čárce

Bc. Jan Kuklík

Vedoucí práce: Ing. Jakub Šťastný, Ph.D.

Studijní program: Otevřená informatika (magisterský)

Obor: Počítačové inženýrství

11. května 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Jan Kuklík**

Studijní program: Otevřená informatika
Obor: Počítačové inženýrství

Název tématu: **Aritmetické operátory v pohyblivé řádové čárce**

Pokyny pro vypracování:

1. Srovnejte vlastnosti obvodů pro aritmetické operace v pevné a pohyblivé řádové čárce. Zvažte možnosti implementace (kombinační obvod vs. realizace s pipeline).
2. Navrhněte a ověřte správnou funkci základních znovupoužitelných bloků pro aritmetické operace (násobení, sčítání, dělení, konverze na a z fixed point čísla). Bloky implementujte v jazyce VHDL na RTL úrovni na platformě FPGA.
3. Bloky integrujte s procesorem PLASMA a demonstrujte jejich použití pomocí jednoduché aplikace.

Seznam odborné literatury:

[1] ŠŤASTNÝ Jakub. FPGA Prakticky, BEN Praha 2010
[2] PALAŠČÁK Ján. Implementace FFT jádra, diplomová práce, FEL ČVUT v Praze, 2012
[3] GOLDBERG David. What Every Computer Scientist Should Know About Floating Point Arithmetic, ACM Computig Surveys, 1991
[4] MALIK A. Design tradeoff analysis of floating-point adders in FPGAs, Canadian Journal of Electrical and Computer Engineering, vol. 33, no. 3/4, pp. 169-175

Vedoucí: Ing. Jakub Šťastný, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 4. 2. 2014

Poděkování

Na tomto místě bych rád poděkoval vedoucímu diplomové práce Ing. Jakubovi Šťastnému, Ph.D. za vedení této práce a za odborné rady. Děkuji také rodině, přítelkyni a kamarádům za podporu v průběhu celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 5. 5. 2014 .....

Abstract

This thesis is dealing with a design of conversion from and to fixed point number and with a design of arithmetical operators in floating point (multiplication, addition, division) on RTL level at the platform of FGPA. The work is analyzing existing solutions of implementation of operators. On the base of those implementations and description of the operation's algorithm the author deducts his own design of those operators. The design for the conversion is arising from the description of the algorithm of the transfer's operation. All those circuits have through generic constants adjustable amount of processed numbers. Through verification is verified the right function of designed logic circuits. The thesis is describing the integration with the processor Plasma and is demonstrating the function of individual operators and logic circuits for conversion on the examples of simple applications.

Abstrakt

Diplomová práce se zabývá návrhem konverze z a na fixed point číslo a aritmetických operátorů v pohyblivé řádové čárce (násobení, sčítání, dělení) na RTL úrovni na platformě FPGA. Analyzuje existující řešení implementace operátorů. Na základě těchto implementací a popisu algoritmu operace je vyvozen vlastní návrh těchto operátorů. Návrh pro konverzi vzniká z popisu algoritmu operace převodu. Všechny tyto obvody mají pomocí generických konstant volitelnou šířku zpracovávaných čísel. Pomocí verifikace je ověřena správná funkce navrhovaných bloků. Práce popisuje integraci s procesorem Plasma a demonstruje funkci jednotlivých operátorů a bloků pro konverzi na jednoduchých aplikacích.

Obsah

1 Úvod **1**

2 Cíle práce **3**

3 Reprezentace numerických hodnot **5**

3.1 Reprezentace čísel ve formátu pevné řádové čárky . . . 5

3.1.1 Aritmetické operace . . . 6

3.1.1.1 Součet . . . 6

3.1.1.2 Rozdíl . . . 7

3.1.1.3 Součin . . . 7

3.1.1.4 Podíl . . . 8

3.1.2 Přednosti a zápory formátu pevné řádové čárky . . . 8

3.2 Reprezentace čísel ve formátu pohyblivé řádové čárky . . . 8

3.2.1 Základní formáty podle normy IEEE 754 . . . 10

3.2.1.1 Formát single . . . 10

3.2.1.2 Formát double . . . 10

3.2.1.3 Formát extended . . . 11

3.2.1.4 Formát binary16 . . . 11

3.2.2 Ostatní formáty . . . 11

3.2.3 Výjimky . . . 11

3.2.4 Zaokrouhlování . . . 12

3.2.5 Aritmetické operace . . . 12

3.2.5.1 Součet a rozdíl . . . 12

3.2.5.2 Součin . . . 13

3.2.5.3 Podíl . . . 14

3.2.6 Přednosti a zápory formátu plovoucí řádové čárky . . . 14

3.2.7 Porovnání reprezentací . . . 15

4 Systémový návrh **17**

4.1 Sčítačka . . . 17

4.1.1 Analýza existujících řešení . . . 17

4.1.1.1 Popis řešení autorů Preethi Sudha Gollamudi a M. Kamaraju 17

4.1.1.2 Popis řešení autorů G. Govindu, L. Zhuo, S. Choi a V. Prasanna 18

4.1.1.3 Porovnání implementací sčítaček . . . 18

4.1.2 Algoritmus sčítaní . . . 19

4.1.3 Specifikace obvodu sčítačky . . . 20
4.2 Násobička . . . 21
4.2.1 Analýza existujících řešení . . . 21
4.2.1.1 Popis řešení autorů L. Rajesh, Prashant. V. Joshi a Dr.S.S. Manvi . . . 21
4.2.1.2 Popis řešení autorů G. Govindu, L. Zhuo, S. Choi a V. Prasanna 22
4.2.1.3 Porovnání implementací násobiček . . . 23
4.2.2 Algoritmus násobení . . . 24
4.2.3 Specifikace obvodu násobičky . . . 25
4.3 Dělička . . . 25
4.3.1 Analýza existujících řešení . . . 25
4.3.1.1 Popis řešení autorů Anuja J. Thakkar a Abdel Ejnioui . . . 25
4.3.1.2 Popis řešení autorů Xin Fang a Miriam Leeser . . . 26
4.3.1.3 Porovnání implementací děliček . . . 26
4.3.2 Algoritmus dělení . . . 27
4.3.3 Specifikace obvodu děličky . . . 28
4.4 Konverze z fixed point čísla . . . 28
4.4.1 Algoritmus konverze . . . 28
4.4.2 Specifikace obvodu pro konverzi z fixed point čísla . . . 29
4.5 Konverze z floating point čísla . . . 30
4.5.1 Algoritmus konverze . . . 30
4.5.2 Specifikace obvodu pro konverzi z floating point čísla . . . 30
4.6 Integrace s procesorem Plasma . . . 31
4.6.1 Procesor Plasma . . . 31
4.6.2 Kompilace programu . . . 32

5 RTL návrh **33**
5.1 Sčítačka . . . 33
5.1.1 Popis bloků . . . 33
5.1.2 Integrace s procesorem Plasma . . . 35
5.2 Násobička . . . 38
5.2.1 Popis bloků . . . 38
5.2.2 Integrace s procesorem Plasma . . . 40
5.3 Dělička . . . 40
5.3.1 Popis bloků . . . 40
5.3.2 Integrace s procesorem Plasma . . . 45
5.4 Konverze z fixed point čísla . . . 45
5.4.1 Popis bloku . . . 45
5.4.2 Integrace s procesorem Plasma . . . 45
5.5 Konverze z floating point čísla . . . 46
5.5.1 Popis bloku . . . 46
5.5.2 Integrace s procesorem Plasma . . . 46

6 Verifikace **47**
6.1 Verifikační plán . . . 47
6.1.1 Verifikační plán pro sčítačku . . . 47
6.1.2 Verifikační plán pro násobičku . . . 48
6.1.3 Verifikační plán pro děličku . . . 48
6.1.4 Verifikační plán pro konverzi z FX čísla . . . 49
6.1.5 Verifikační plán pro konverzi z FP čísla . . . 50
6.2 Verifikace na RTL úrovni . . . 50
6.2.1 Verifikace sčítačky na RTL úrovni . . . 50
6.2.2 Verifikace násobičky na RTL úrovni . . . 52
6.2.3 Verifikace děličky na RTL úrovni . . . 53
6.2.4 Verifikace konverze z FX čísla na RTL úrovni . . . 54
6.2.5 Verifikace konverze z FP čísla na RTL úrovni . . . 54
6.3 Verifikace na GATE LEVEL úrovni . . . 55
6.3.1 Verifikace sčítačky na GATE LEVEL úrovni . . . 55
6.3.2 Verifikace násobičky na GATE LEVEL úrovni . . . 56
6.3.3 Verifikace děličky na GATE LEVEL úrovni . . . 56
6.3.4 Verifikace konverze z FX čísla na GATE LEVEL úrovni . . . 56
6.3.5 Verifikace konverze z FP čísla na GATE LEVEL úrovni . . . 57

7 Implementace **59**
7.1 Popis implementace sčítačky . . . 59
7.1.1 Příklad softwaru pro výpočet na procesoru Plasma . . . 61
7.2 Popis implementace násobičky . . . 62
7.2.1 Příklad softwaru pro výpočet na procesoru Plasma . . . 63
7.3 Popis implementace děličky . . . 64
7.3.1 Příklad softwaru pro výpočet na procesoru Plasma . . . 65
7.4 Popis implementace konverze na a z fixed point čísla . . . 66
7.4.1 Příklad softwaru pro výpočet na procesoru Plasma . . . 67

8 Závěr **69**

Literatura **71**

A Seznam použitých zkratek **73**

B Použitý software, verze aplikací **75**

C Obsah přiloženého CD **77**

D Zdrojové kódy ukázkových programů **79**
D.1 Příklad kódu na využití sčítačky s procesorem Plasma . . . 79
D.2 Příklad kódu na využití násobičky s procesorem Plasma . . . 81
D.3 Příklad kódu na využití děličky s procesorem Plasma . . . 82
D.4 Příklad kódu na využití konverze s procesorem Plasma . . . 84

Seznam obrázků

3.1 16 bitové číslo (N = 16) ve formátu Q3.12 5
3.2 Ukázka sčítání v pevné řádové čárce bez znaménka ve formátu Q5.3 [8] 6
3.3 Ukázka násobení v pevné řádové čárce ve formátu Q6.2 [8] 7
3.4 Formát single [6] 10
3.5 Formát double [6] 11

4.1 Blokový diagram sčítačky pro čísla v pohyblivé řádové čárce [9] 18
4.2 Blokový diagram sčítačky/odčítačky [10] 19
4.3 Blokový diagram násobičky [12] 22
4.4 Násobička čísel ve formátu plovoucí řádové čárky s pipeline [12] 23
4.5 Blokový diagram násobičky [10] 23
4.6 Dělička s přesností double ve formátu plovoucí řádové čárky [13] 26
4.7 Propojení komponent děličky [14] 27
4.8 Blokové schéma připojení obvodu aritmetické operace k procesoru Plasma . . 31
4.9 Blokové schéma procesoru Plasma [18] 32

5.1 RTL schéma sčítačky 36
5.2 RTL schéma násobičky 41
5.3 RTL schéma děličky 44
5.4 Blokové schéma konverze z fixed point čísla 45
5.5 Blokové schéma konverze z floating point čísla 46

6.1 Postup verifikace pro sčítačku. Srovnání modelem [11] 49
6.2 Postup verifikace pro děličku. Srovnání modelem [11] 50
6.3 Postup verifikace pro konverzi z FX čísla. Srovnání modelem [11] 51
6.4 Postup verifikace pro konverzi z FP čísla. Srovnání modelem [11] 52
6.5 Ukázka simulace sčítačky na RTL úrovni 52
6.6 Ukázka simulace násobičky na RTL úrovni 53
6.7 Ukázka simulace děličky na RTL úrovni 54
6.8 Ukázka simulace konverze z FX čísla na RTL úrovni 54
6.9 Ukázka simulace konverze z FP čísla na RTL úrovni 55
6.10 Ukázka simulace sčítačky na GATE LEVEL úrovni 55
6.11 Ukázka simulace násobičky na GATE LEVEL úrovni 56
6.12 Ukázka simulace děličky na GATE LEVEL úrovni 57
6.13 Ukázka simulace konverze z FX čísla na GATE LEVEL úrovni 57
6.14 Ukázka simulace konverze z FP čísla na GATE LEVEL úrovni 57

Seznam tabulek

3.1 Typ čísla v závislosti na hraničních hodnotách (formát single)[6] 9
3.2 Ukázka kódování čísel v pohyblivé řádové čárce (formát single)[6] 9
3.3 Základní formáty podle normy IEEE 754 [7] 10
3.4 Výsledky součtu nebo rozdílu se speciálními hodnotami [6] 13
3.5 Výsledky součinu se speciálními hodnotami [6] 14
3.6 Porovnání reprezentací hodnot v FP a FX formátech 15

4.1 Porovnání 32 bitových sčítaček v pohyblivé řádové čárce 19
4.2 Rozhraní sčítačky 21
4.3 Porovnání 32 bitových násobiček v pohyblivé řádové čárce 24
4.4 Rozhraní násobičky 25
4.5 Porovnání 64 bitových děliček v pohyblivé řádové čárce 27
4.6 Rozhraní děličky 29
4.7 Rozhraní konverze z fixed point čísla 30
4.8 Rozhraní konverze z floating point čísla 31

5.1 Závislost výstupů na vstupech v bloku n_case sčítačky (podle[6]) 34
5.2 Výstupní znaménko a znaménko operace podle vstupních znamének v bloku block_adder 35
5.3 Mapovaní adresového prostoru procesoru Plasma 37
5.4 Rozhraní sčítačky s procesorem Plasma 38
5.5 Registr pro řízení sčítání s procesorem Plasma 38
5.6 Závislost výstupů na vstupech v bloku n_case násobičky (podle[6]) 39
5.7 Závislost výstupů na vstupech v bloku n_case děličky 42
5.8 Řídící signály a reakce čítače děličky 43
5.9 Registr pro řízení konverze z FX čísla s procesorem Plasma 46

7.1 Zvolené FPGA pro imlementaci obvodů 59
7.2 Implementační parametry obvodu sčítačky 61
7.3 Implementační parametry obvodu násobičky 63
7.4 Implementační parametry obvodu děličky 65
7.5 Implementační parametry obvodu pro konverzi na a z fixed point čísla 67

Kapitola 1

Úvod

S aritmetickými operacemi se setkáváme v každodenním životě. Tyto operace jsou základním prvkem nepřeberného množství aplikací.

Aritmetické operace v pohyblivé řádové čárce jsou využívané pro obecné algoritmy s velkými nároky na přesnost výpočtu. Používají se v matematických koprocesorech, signálových procesorech a ve specializovaných číslicových obvodech. Oproti softwarovým výpočtům s pohyblivou řádovou čárkou jsou operace prováděné na číslicových systémech mnohonásobně rychlejší.

Návrh těchto aritmetických operátorů bude využit pro další vývoj obvodů, které mají vekou výpočetní náročnost a potřebují pro svoji práci velkou přesnost výpočtu. Příkladem obvodů využívajících tyto operace jsou algoritmy pro nelineární transformace pro zpracování signálu.

Kapitola 2

Cíle práce

Cílem práce je navrhnout znovupoužitelné bloky pro aritmetické operace v pohyblivé řádové čárce a konverzi z a na fixed point. Bloky jsou implementované v jazyce VHDL na RTL úrovni na platformě FPGA. Bloky umožňují změnu formátu zpracovávaných hodnot pomocí generických konstant.

Na základě studie existujících implementací a popisu algoritmu operace je proveden návrh jednotlivých bloků aritmetického operátora. Z popisu převodu čísla z pevné řádové čárky na číslo v pohyblivé řádové čárce (a zpět) je vytvořen návrh bloku pro konverzi. Je provedena statická časová analýza pro ověření minimální hodinové frekvence pro připojení k procesoru Plasma tj. 50 MHz. Dalším krokem je provedení verifikace správné funkce navržených bloků.

Posledním krokem je propojení s procesorem Plasma a ověření funkčnosti s tímto procesorem za použití jednoduché aplikace.

Kapitola 3

Reprezentace numerických hodnot

Kapitola popisuje reprezentaci hodnot ve formátech pevné a pohyblivé řádové čárky. Dále jsou v textu popsány základní aritmetické operace v těchto formátech.

3.1 Reprezentace čísel ve formátu pevné řádové čárky

Reprezentace čísla v pevné řádové čárce patří mezi základní formy binární reprezentace číselných hodnot [3]. Při zakódování čísla v tomto formátu je reálné číslo uloženo do N bitů tak, že prvních m hodnot je celá část čísla (integer bits) a zbylá část n bitů je desetinná část čísla (fractional bits) [5]. Toto kódování se také označuje jako kódování Qm.n. Kódování Qm.n je využívané jak pro neznaménkové typy čísel tak i pro znaménkové. Podle typu ukládaných čísel se nastavují hodnoty m a n. Jedná-li se o znaménkový typ je šířka ukládaného čísla N = m + n + 1 jinak je N součet hodnot m a n. Číslo může být uloženo jak v přímém tak v doplňkovém kódu. Na obrázku 3.1 je vidět rozložení 16 bitového čísla (se znaménkem) ve formátu Q3.12.

Obrázek 3.1: 16 bitové číslo (N = 16) ve formátu Q3.12

Mohou nastat případy kdy jedna z částí formátu Q není využita. Celočíselné šestnáctibitové číslo se znaménkem (N = 16) je značeno Q15.0 Desetinné šestnáctibitové číslo se znaménkem (N = 16) je značeno Q0.15, Q.15 nebo Q15.

Výpočet hodnoty čísla z N bitového slova ve formátu Qm.n (kódování v dvojkovém doplňku) je dán následujícím vzorcem. Proměnná x je reálná hodnota, b_z je hodnota na pozici z.

$$x = -b_{N-1} \times 2^m + \sum_{N-2}^{i=0} b_i \times 2^{i-1}$$

Výpočet hodnoty čísla z N bitového slova ve formátu Qm.n (kódování v přímém kódu) je dán následujícím vzorcem. Proměnná x je reálná hodnota, b_z je hodnota na pozici z.

$$x = (-1)^{b_{N-1}} \times \sum_{N-2}^{i=0} b_i \times 2^{i-1}$$

Při převodu čísla do formátu pevné řádové čárky je potřeba hodnotu upravit tak, aby ji bylo možné převést do dvojkové soustavy. K tomuto účelu se využívá zkrácení nebo zaokrouhlení hodnoty. V následujícím příkladu je ukázka převodu do znaménkového formátu Q4.5 (formát Qm.n).

$$x = 8,8 \rightarrow x \times 2^n \rightarrow 8,6 \times 2^5 = 281,6 \rightarrow [\text{zaokrouhlení}]\ 282 \rightarrow 01000,11010$$
$$x = 8,8 \rightarrow x \times 2^n \rightarrow 8,6 \times 2^5 = 281,6 \rightarrow [\text{zkrácení}]\ 281 \rightarrow 01000,11001$$
$$x = -10,3 \rightarrow x \times 2^n \rightarrow -10,3 \times 2^5 = -329,6 \rightarrow [\text{zaokrouhlení}]\ -330$$
$$\text{Dvojkový doplněk } x = 2^{m+n+1} - x \rightarrow 2^{10} - 330 = 347 \rightarrow 10101,10110$$

Následující příklad ukazuje zakódování v přímém kódu Q8.4.

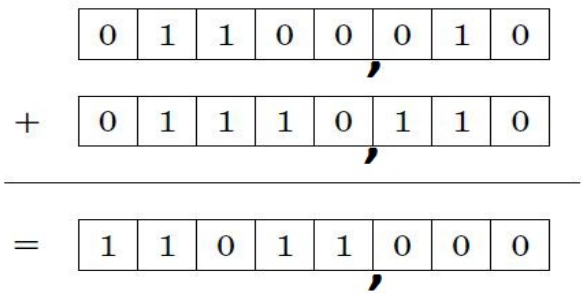
$$13,625_{10} \rightarrow 13,625 \times 2^4 = 218 \rightarrow 00001101,1010_2$$

3.1.1 Aritmetické operace

Tato část je zaměřena na popsání základních aritmetických operací ve formátu pevné řádové čárky. Přesný formát čísel je dán podle typu operací a typu očekávaných hodnot.

3.1.1.1 Součet

Operace součtu může probíhat pouze za předpokladu, že obě vstupní čísla mají stejný formát. Pokud se vstupní formáty liší, musí dojít k úpravě vstupů pomocí rozšiřování či zkracování. Výsledek součtu je o jeden bit větší než šířka vstupního formátu. Většina sčítaček šířku nezvyšuje, proto může nastat přetečení (overflow), což je stav, kdy se nevejde výsledek do počtu bitů formátu čísla. Na obrázku 3.2 je ukázka sčítání čísel bez znaménka a v tomto případě nedochází k přetečení.



Obrázek 3.2: Ukázka sčítání v pevné řádové čárce bez znaménka ve formátu Q5.3 [8]

Operace součtu může být realizována např. paralelní sčítačkou nebo seriovou sčítačkou. Základní elementární logickou funkcí pro tyto obvody je funkce poloviční a plné sčítačky.

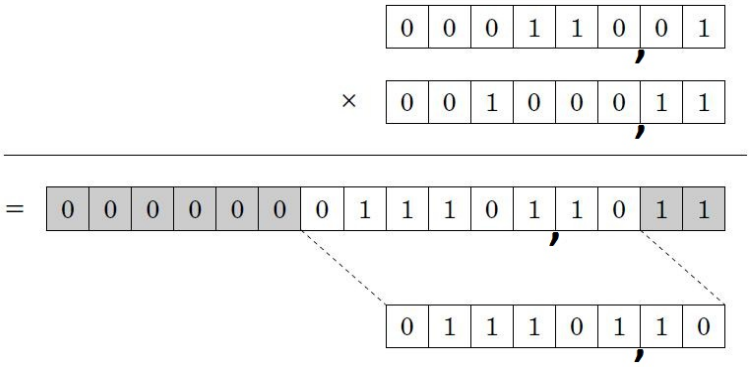
3.1.1.2 Rozdíl

Operace rozdílu probíhá obdobně jako operace součtu. Operace probíhá za předpokladu, že formáty vstupních hodnot jsou totožné. Pokud tomu tak není, musí pomocí rozšiřování či zkracovaní vstupních hodnot dojít ke sjednocení formátu. Jako u sčítání je nutné počítat s tím, že výsledek muže být o jeden bit širší než vstupní formát.

Operace rozdílu je realizována pomocí odčítaček. Pro návrh těchto obvodů se využívá stejných principů jako při návrhu sčítaček. Místo oddělených obvodů pro sčítání a odčítání se používají obvody nazývané sčítačko-odčítačky. Tyto obvody, v závislosti na vstupu řídící proměnné, provádějí operaci sčítání nebo odčítání [3].

3.1.1.3 Součin

Při operaci součinu se počet bitů výsledku zvyšuje a to podle vzorce $Qm_a.n_a \times Qm_b.n_b = Q(m_a + m_b + 1).(n_a + n_b)$. Vstupní formáty čísel mohou být rozdílné. Obrázek 3.3 zobrazuje ukázku násobení. Na obrázku je vidět, jak se zvětší výsledek oproti vstupním formátům.



Obrázek 3.3: Ukázka násobení v pevné řádové čárce ve formátu Q6.2 [8]

U operace součinu je jedním z realizačních předpokladů, zda násobené hodnoty mohou být i záporné. Pokud je realizováno i násobení se zápornými hodnotami, záleží jak jsou záporné hodnoty zakódovány. Zakódovanou hodnotu lze převést na znaménko a absolutní hodnotu čísla. Poté provést násobení a určit znaménko výsledku. Samotné násobení probíhá stejně jako "násobení pod sebe". V prvním kroku se násobí první činitel nejnižším bitem druhého činitele. V druhém kroku se první činitel násobí druhým nejnižším bitem druhého činitele a výsledná hodnota je posunuta vlevo o jeden bit. Takto se pokračuje až do násobení nejvyššího bitu druhého činitele. Poté se výsledky násobení (už posunuté) sečtou a vznikne výsledek násobení. Při realizaci obvodu násobení je výhodnější provádět operaci sčítání postupně a to už po určení dílčího násobení, než při jednom velkém součtu na konci [3]. Při reprezentaci čísel v dvojkovém doplňku je v posledním cyklu hodnota násobení místo přičtení odečtena a tím se docílí správného výsledku i se znaménkem. Realizace násobení může být prováděna např. pomocí sériové nebo paralelní násobičky.

3.1.1.4 Podíl

Operace podílu je realizována modifikací "dělení na papíře". Z tohoto způsobu vychází dělení bez a s návratem přes nulu. Oba tyto algoritmy pracují pouze sériově. Dělení s návratem přes nulu pracuje tak, že se určí zda hodnota dílčího zbytku (v prvním kroku dělence) je větší než dělitel, pak se do výsledku uloží log. 1 jinak log. 0. Pokud ukládaná hodnota byla nula, proběhne pouze rotace dílčího výsledku vlevo, jinak proběhne odečtení dělitele od dílčího zbytku a rotování vlevo. Algoritmus se opakuje, dokud dílčí zbytek není roven nule nebo pokud neproběhl počet očekávaných výpočtů. Dělení záporných čísel se provádí převedením na dělení nezáporných čísel a znaménko výsledku se určuje odděleně. Operace dělení je ve srovnání s násobením podstatně složitější. Pro nezáporné hodnoty lze obvody zrychlit pomocí přeskakování nul a jedniček. Dalšími metodami zrychlení jsou metody SRT a nebo převedením dělení na operaci násobení [3].

3.1.2 Přednosti a zápory formátu pevné řádové čárky

Výhody

- Není potřeba uchovávat pozici řádové čárky. Tato pozice je předem dána podle algoritmu nebo prováděné operace.
- Pokud jsou známé rozsahy vstupních hodnot, dá se jednoduše určit, kde je potřeba zvětšit nebo naopak snížit rozsah.
- S tímto formátem jde dobře dodržet požadovaná přesnost.

Nevýhody

- Při zpracování hodnot s velkou dynamikou tj. velký rozdíl mezi nejvyšší a nejnižší absolutní hodnotou by se muselo pro každé číslo alokovat velké množství bitů.
- Při operacích násobení/dělení vzniká problém s velikostí výsledku. Rozsah výsledku se může až zdvojnásobit než vstupní formát.

3.2 Reprezentace čísel ve formátu pohyblivé řádové čárky

Tato reprezentace se využívá pro zvětšení rozsahu ukládaných racionálních čísel. Z anglického názvu (Floating Point) bývá formát označován zkratkou FP. Základní formáty jsou definovány normou IEEE 754. Tato norma byla vydána v roce 1985 a poté byla v roce 2008 rozšířena o poloviční a čtyřnásobnou přesnost a doplněna o formáty s desítkovou aritmetikou [7]. Další část textu o formátu plovoucí řádové čárky je čerpána z [1, 3]. Reprezentace hodnot je vyjádřena tímto vztahem:

$$X = (-1)^S \times radix^{exp-bias} \times m$$

X značí reprezentovanou reálnou hodnotu. X může nabývat také nečíselné hodnoty NaN (Not a Number), $+\infty$ a $-\infty$.

S je znaménkový bit nabývající hodnot 0 nebo 1.
$radix$ je báze. U IEEE 754-1985 se používá dvojka.
exp je kladná hodnota, od které se odečítá $bias$. Hodnota exponentu ($exp - bias$) je uložena v posunuté formě (aditivní kód). Krajní hodnoty rozsahu jsou použity pro speciální hodnoty čísel.
$bias$ je kladná hodnota, která se volí podle vztahu $bias = 2^{(\text{počet bitů exponentu})-1} - 1$.
m je mantisa.

V tomto formátu jsou reprezentovaná normalizovaná čísla, denormalizovaná čísla a speciální hodnoty. Normalizované číslo je, pokud první explicitně nevyjádřený bit je roven jedné. Denormalizované číslo proti normalizovanému má skrytý bit roven nule a exponent je nastaven na nejnižší hodnotu tj. 0. Speciální hodnota NaN vzniká pokud operace má nejasný výsledek tj. $0/0$, 0^0, nebo odmocnění záporného čísla. Nekonečná hodnota vzniká při přetečení, dělení nulou nebo při vyjádření funkcí typu $\log(0)$. V tabulce 3.1 jsou na formátu single předvedeny všechny možné typy čísel a jejich hraniční hodnoty.

Znaménkový bit	**Exponent**	**Mantisa**	**Význam**
0	$0 < \exp < 255$	> 0	normalizované kladné číslo
1	$0 < \exp < 255$	> 0	normalizované záporné číslo
0	0	> 0	denormalizované kladné číslo
1	0	> 0	denormalizované záporné číslo
0	0	0	kladná nula
1	0	0	záporná nula
0	255	0	$+\infty$
1	255	0	$-\infty$
0	255	> 0	NaN (Not a Number)
1	255	> 0	NaN (Not a Number)

Tabulka 3.1: Typ čísla v závislosti na hraničních hodnotách (formát single)[6]

V tabulce 3.2 je několik příkladů jak se číslo převádí do FP reprezentace.

Hodnota	**Binární výstup**			**Decimální výstup**		
	S	**Exponent**	**Mantisa**	**S**	**Exponent**	**Mantisa**
0,00	0	0000 0000	000 0000 0000 0000 0000	+	-127	0,000000
-1,00	1	0111 1111	000 0000 0000 0000 0000	-	0	1,000000
-999,00	1	1000 1000	111 1001 1100 0000 0000	-	9	1,951172
-6,50	1	1000 0001	101 0000 0000 0000 0000	-	2	1,625000

Tabulka 3.2: Ukázka kódování čísel v pohyblivé řádové čárce (formát single)[6]

Následuje ukázka převodu reálného čísla -6,5 do kódování v pohyblivé řádové čárce.

1. $6,5_{10} = 110,1_2$ číslo se převede do dvojkové soustavy
2. $110,1_2 = 1,101_2 \cdot 2^2$ číslo je posunuto tak, aby bylo normalizované

3. $2+127 = 129 = 10000001_2$ k exponentu se přičte 127 a převede se do binární soustavy
4. mantisa je rovna 101 a pouze se doplní nuly vpravo
5. poněvadž je číslo záporné bude znaménkový bit roven jedné
6. výsledkem je poslední řádek v tabulce 3.2

3.2.1 Základní formáty podle normy IEEE 754

V tabulce 3.3 jsou vypsány typy, které rozlišuje norma IEEE 754. V dalším popisu formátů plovoucí řádové čárky se věnuji formátům ze základní normy z roku 1985 a některým typům rozšířené normy.

IEEE 754-2008	IEEE 754-1985	Bitů	Základ
binary16	-	16	2
binary32	single	32	2
binary64	double	64	2
-	extended	80	2
binary128	-	128	2
decimal32	-	32	10
decimal64	-	64	10
decimal128	-	128	10

Tabulka 3.3: Základní formáty podle normy IEEE 754 [7]

3.2.1.1 Formát single

U jednoduchého formátu single je 32 bitů rozděleno na tři části. Znaménko (s) je 31 bit, exponent je od 30 do 23 bitu a mantisa je na 22 až 0 bitu. Obrázek 3.4 zobrazuje rozmístění hodnot ve formátu single. Formát single je využívan např. v programovacím jazyce C i Java pro datový typ float.

bit	31	30 29 ... 24 23	22 21 ... 3 2 1 0
význam	s	exponent (8 bitů)	mantisa (23 bitů)

Obrázek 3.4: Formát single [6]

3.2.1.2 Formát double

U formátu double je zvětšen rozsah z 32 bitů na 64 bitů. Změna je ve velikosti exponentu a mantisy. Obrázek 3.5 zobrazuje rozmístění hodnot ve formátu double. Formát double je využíván např. v programovacím jazyce C, Java a Pascal pro datový typ double.

Rozdíl mezi formátem double a single je v možnosti reprezentace různých rozsahů. Dalším rozdílem je přesnost čísel. Stejné číslo, které bude mít dlouhý desetinný rozvoj formát single nebude reprezentovat tak přesně jako typ double. Typ double zabírá větší prostor a to u čísel s kratším desetinným rozvojem je nevýhodou. S větší přesností a větším rozsahem typu double je spojena také časová náročnost výpočtů, která je o proti formátu single větší a také spotřeba i plocha logického obvodu je větší.

bit	63	62 ... 52	51 ... 0
význam	s	exponent (11 bitů)	mantisa 52(bitů)

Obrázek 3.5: Formát double [6]

3.2.1.3 Formát extended

Tento formát je jinak nazývaný temporary. Pro uložení čísla využívá 80 bitů. Rozdělení na tři části je takto: 1 bit pro znaménko, 15 bitů pro exponent a 64 bitů pro mantisu. Tento formát je využíván např. u programovacího jazyka Pascal pro datový typ extended nebo u jazyka C pro datový typ long double.

3.2.1.4 Formát binary16

Tento formát je také označován jako half precision (znamý také jako 1.5.10 nebo s10e5 minifloat). Hodnota čísla je uložena do 16 bitů, kde jeden bit je znaménko, pět bitů exponent a mantisa je zbývajících deset bitů. Tento formát byl přidán do normy z důvodu častého využívání v moderních grafických jednotkách, kde minimalizuje využití paměti a sběrnice. Je také používán v grafických knihovnách (např. OpenGL) a obvodech pro dekódování MP3 [16].

3.2.2 Ostatní formáty

Kromě formátů popisovaných normou IEEE 754 existují další formáty pro reprezentaci hodnot v plovoucí řádové čárce. Jedním z formátů je formát real šířky šesti bitů využívaný v programovacím jazyku Turbo Pascal vytvořený firmou Borland [15].

3.2.3 Výjimky

Při práci s plovoucí řádovou čárkou může podle [1] vzniknout výjimka přetečení, podtečení, neplatná operace a nepřesnost.

- Přetečení je signalizováno, pokud se hodnota exponentu dostane na nejvyšší možnou hodnotu. Jedná se o hodnoty $+\infty$ a $-\infty$.

- Podtečení je detekováno, pokud hodnota exponentu je na nejnižší možné hodnotě. Tyto hodnoty jsou reprezentovány denormalizovanými čísly a nulou.
- Výjimka neplatná operace je signalizována jen tehdy, pokud není definovaný výsledek. V těchto případech jsou operandy neplatné pro zadané operace, které mají být provedeny. Operace produkuje výsledek ve formátu plovoucí řádové čárky nazývaný NaN (Not a Number). Neplatná operace nastává v těchto případech: operace s NaN, sčítání/odčítaní $+\infty$ a $-\infty$, násobení 0 a ∞ , dělení $0/0$ a ∞/∞ , odmocnina z čísel menších než 0 aj.
- Nepřesnost vzniká pokud se zaokrouhluje nebo pokud dojde k přetečení nebo podtečení.

3.2.4 Zaokrouhlování

V normě IEEE 754 je specifikováno několik zaokrouhlovacích módů [1]. Tyto módy jsou:

- Zaokrouhlování na nejbližší hodnotu - výsledkem je číslo, které je nejblíže zaokrouhlovanému číslu (např. 10,00 1000 = 10,00; 00,01 1000 = 00,11).
- Zaokrouhlování směrem k $-\infty$ - výsledkem zaokrouhlování není hodnota větší než vstupní číslo (např. 1,11 01 = 1,11; 1,001 = 1,00).
- Zaokrouhlování směrem k $+\infty$ - výsledkem zaokrouhlování není hodnota menší než vstupní číslo (např. 1,11 01 = 10,00; 1,001 = 1,01).
- Zaokrouhlování směrem k nule - nejednodušší zaokrouhlování, guard bity jsou pouze oříznuty (např. 01,11 01 = 01,11; 01,00 01 = 01,00).

3.2.5 Aritmetické operace

3.2.5.1 Součet a rozdíl

Vstupem jsou dvě čísla X a Y totožného formátu a výstupem je hodnota $Z = X \pm Y$, kde Z je totožného formátu jako vstupní hodnoty. Výpočet probíhá dle tohoto algoritmu [17, 6]:

1. V prvním kroku se kontroluje, zda hodnoty čísel nepatří do kategorie speciálních hodnot tj. NaN nebo nekonečno. Pokud ano výsledek výpočtu je dle tabulky 3.4. U operace odčítání se musí i u výsledku se speciálními hodnotami přihlížet ke kombinaci znamének, aby výsledek měl správné znaménko.
2. Následně se porovnávají hodnoty exponentů a zjišťuje se, které hodnoty jsou větší. Pokud jsou hodnoty exponentu shodné, následující krok se přeskakuje.
3. Nyní se posune mantisa menšího čísla o rozdíl většího exponentu s menším.
4. Dalším krokem je součet nebo odečtení mantis operandů. Zde se musí dávat pozor na přenos. Pokud nastane přenos, je celá mantisa rotována doprava a nejvyšší bit mantisy je nahrazen přenosem.

Hodnota X	**Hodnota Y**	**Výsledek operace X+Y**
konečná	$+\infty$	$+\infty$
konečná	$-\infty$	$-\infty$
$+\infty$	konečná	$+\infty$
$-\infty$	konečná	$-\infty$
$+\infty$	$+\infty$	$+\infty$
$-\infty$	$-\infty$	$-\infty$
$+\infty$	$-\infty$	NaN
$-\infty$	$+\infty$	NaN
NaN	libovolná	NaN
libovolná	NaN	NaN

Tabulka 3.4: Výsledky součtu nebo rozdílu se speciálními hodnotami [6]

5. Pokud mantisa nemá nejvyšší bit roven jedné, musí nastat posuv doleva a to i několikrát, současně při této rotaci se exponent snižuje (normalizace).

6. V dalším postupu se musí vyjádřit znaménko výsledku. V případě sčítaní se kopíruje znaménko většího čísla a u odečítaní se bere v potaz, zda je číslo menšitel nebo menšenec.

7. Pokud dojde k přetečení exponentu, je výsledkem nekonečno. Znaménko nekonečna se určuje podle šestého kroku. Pokud dojde k podtečení exponentu, je výsledkem nula a znaménko určuje šestý krok.

8. Následuje zaokrouhlení, které podle normy IEEE 754 jsou: Zaokrouhlení k nejbližší zobrazitelné hodnotě, zaokrouhlení k nekonečnu, zaokrouhlení k mínus nekonečnu a zaokrouhlení k nule.

3.2.5.2 Součin

Vstupem jsou dvě čísla X a Y totožného formátu a výstupem je hodnota $Z = X \times Y$, kde Z je totožného formátu jako vstupní hodnoty. Postupuje se podle algoritmu [17, 6]:

1. Výsledné znaménko vznikne operací nonekvivalence znaménkových bitů operandů.

2. Stejně jako u operace sčítaní nebo odčítaní se zjišťuje, zda do operace nevstupují speciální hodnoty. Pokud ano, výsledky jsou podle tabulky 3.5.

3. Exponent výsledku se rovná součtu exponentů operandů. Od této hodnoty je ještě nutné odečíst bias, aby byl výsledek správný.

4. Výsledná mantisa je součinem obou mantis operandů. Pokud výsledná mantisa přeteče, je mantisa rotována doprava a exponent je zvýšen o počet rotací. Pokud není nejvyšší bit mantisy roven jedné, probíhá proces normalizace, což znamená, že mantisa rotuje doleva a exponent se snižuje.

Hodnota X	**Hodnota Y**	**Výsledek operace X × Y**
nenulová kladná	$+\infty$	$+\infty$
nenulová kladná	$-\infty$	$-\infty$
nenulová záporná	$+\infty$	$-\infty$
nenulová záporná	$-\infty$	$+\infty$
nula	$+\infty$	NaN
nula	$-\infty$	NaN
$+\infty$	nenulová kladná	$+\infty$
$-\infty$	nenulová kladná	$-\infty$
$+\infty$	nenulová záporná	$-\infty$
$-\infty$	nenulová záporná	$+\infty$
$+\infty$	nula	NaN
$-\infty$	nula	NaN
$+\infty$	$+\infty$	$+\infty$
$+\infty$	$-\infty$	$-\infty$
$-\infty$	$+\infty$	$-\infty$
$-\infty$	$-\infty$	$+\infty$
NaN	libovolná	NaN
libovolná	NaN	NaN

Tabulka 3.5: Výsledky součinu se speciálními hodnotami [6]

5. Pokud dojde k přetečení exponentu, je výsledkem nekonečno. Znaménko nekonečna se určuje podle prvního kroku. Pokud dojde k podtečení exponentu, je výsledkem nula a znaménko určuje první krok.

6. Nakonec proběhne zaokrouhlení podle nastaveného zaokrouhlovacího módu.

3.2.5.3 Podíl

Vstupem jsou dvě čísla X a Y totožného formátu a výstupem je hodnota Z = X / Y, kde Z je totožného formátu jako vstupní hodnoty. Podle autorů článků [17, 6] podíl probíhá obdobně jako násobení, ale rozdíl je v tom, že exponenty se nesčítají, ale odčítají a u mantis místo násobení probíhá dělení. U dělení musí být zachovány tyto výsledky:

$$X/0^{+} = +\infty \quad X/0^{-} = -\infty \quad 0/0 = NaN$$

3.2.6 Přednosti a zápory formátu plovoucí řádové čárky

Výhody

- Je podporován mnoha architekturami díky podpoře FP operací hardwarovým FPU jednotek.
- FPU jednotky podporují minimálně dva formáty dle normy IEEE 754. To je výhodné pro přenos numerických hodnot mezi různými zařízeními.

Nevýhody

- Základní operace s čísly jsou poměrně složité a časově náročné.
- Komplikuje a zpomaluje převod mezi FP formátem a celočíselnými formáty.

3.2.7 Porovnání reprezentací

Oba formáty mají své výhody a nevýhody. Pro FP je výhodou velká dynamika ukládaných hodnot, ale operace s nimi jsou složitější zatímco pro pevnou řádovou čárku je výhodou jednodušší práce s operacemi, ale nedokáže jednoduše reprezentovat dynamické hodnoty. Při určení, který formát je pro danou aplikaci vhodnější, je důležité se zaměřit s jakými hodnotami se bude pracovat. Dalším aspektem je typ operací, které se musí provádět. Má-li aplikace dodržovat přesné zaokrouhlování, je lepší využít pevné řádové čárky. Pokud aplikace má pracovat s hodně rozdílnými hodnotami, je lepší volit plovoucí řádovou čárku.

Například při součtu $1+2^{-30}$ probíhajícím na 32 bitech bude výsledkem součtu ve formátu plovoucí řádové čárky pouze 1 a zbytek se ve výsledku neprojeví a to z důvodu velkého rozdílu exponentů. U pevné řádové čárky bych volil formát bez znaménka Q1.31 a výsledek by nebyl také přesný, ale při součtu by se projevil alespoň na nejnižším bitu a výsledek je proto přesnější než ve FP formátu.

	Výhody	**Nevýhody**
Pevná řádová čárka	Rychlejší aritmetické operace. Menší náročnost implementace reprezentace. Plocha zabraná pro reprezentaci operací je menší. Menší spotřeba energie.	Menší přesnost aritmetických operací. Složitější ukládání dynamických hodnot.
Plovoucí řádová čárka	Větší přesnost aritmetických operací (záleží na konkrétních hodnotách). Větší dynamický rozsah hodnot. Pro vědecké výpočty výhodnější. Pro uživatele je tento formát jednoduchý na použití (neošetřuje přetečení a podtečení).	Pomalé a složité aritmetické operace. Složitý převod mezi formáty. Plocha zabraná pro reprezentaci operací je větší. Větší spotřeba energie. Složitější návrh obvodů.

Tabulka 3.6: Porovnání reprezentací hodnot v FP a FX formátech

Kapitola 4

Systémový návrh

V této kapitole jsou popsané jednotlivé specifikace bloků aritmetických operací a převodníků mezi formáty. U aritmetických operací je popis dvou existujících řešení, porovnání rozdílů a vyvození závěrů pro návrh. Následuje popis algoritmu operace a specifikaci obvodu. Nakonec je popsáno připojení k procesoru Plasma, popis procesoru Plasma a kompilace programu v procesoru.

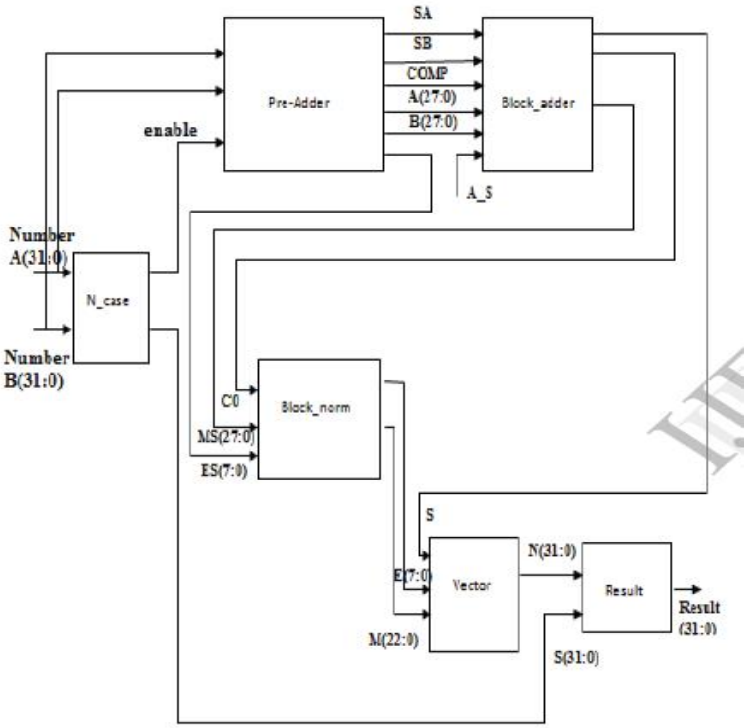
Každý obvod umožňuje konfiguraci v nastavení šířky čísel, s kterými pracuje, což obvody činí univerzálními a použitelnými v široké škále aplikací. Aritmetický obvod musí provádět danou aritmetickou operaci a musí pracovat i se speciálními a denormalizovanými hodnotami. Obvod detekuje přetečení a podtečení. Převodníky převádí z formátu pevné řádové čárky do pohyblivé řádové čárky a obráceně. Pokud dojde k překročení rozsahu výstupního formátu, je výstupem nejbližší odpovídající hodnota v rozsahu. Minimální frekvence, které obvody dosáhnou je 50 MHz.

4.1 Sčítačka

4.1.1 Analýza existujících řešení

4.1.1.1 Popis řešení autorů Preethi Sudha Gollamudi a M. Kamaraju

Implementace podle článku [9] pracuje jako již popisovaný algoritmus v aritmetických operacích FP 3.2.5.1. Při operaci rozdílu implementace převádí číslo, které je záporné do doplňkového kódu místo do přímého. S tím je spojeno rozdílné určení výsledného znaménka. Obrázek 4.1 je blokové schéma této implementace. Schéma má šest základních bloků. Blok `N_case` určuje, zda se jedná o speciální případy. `Pre-Adder` porovnává hodnoty exponentů a upraví hodnotu mantis tak, aby mohlo proběhnout sčítání v `Block_adder`. `Block_norm` normalizuje číslo pokud je to možné. Blok `Vector` provede sjednocení hodnot do čísla v FP formátu. `Result` podle toho zda, se jedná o speciální čísla nebo ne a rozhodne, jaké budou výstupní hodnoty.



Obrázek 4.1: Blokový diagram sčítačky pro čísla v pohyblivé řádové čárce [9]

4.1.1.2 Popis řešení autorů G. Govindu, L. Zhuo, S. Choi a V. Prasanna

V tomto příkladu popisující článek [10] využívá tři standardní etapy algoritmu a to: denormalizace/posuv, sčítaní mantis/odčítání mantis a normalizace/zaokrouhlení. Algoritmus pracuje na stejném principu jako popisované obecné řešení v 3.2.5.1. Toto řešení neřeší v prvním kroku speciální případy. Na obrázku 4.2 je blokové schéma této implementace.

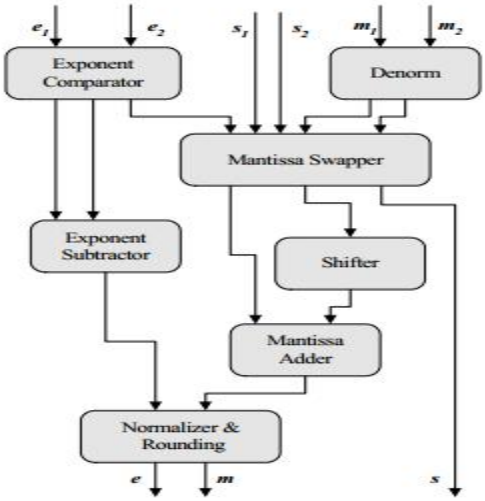
V první etapě se přidává skrytý bit o hodnotě jedna, pokud není exponent čísla roven nule. Poté se prohodí mantisy v závislosti na velikosti exponentů. Pomocí posuvu jsou mantisy posunuty na stejné velikosti exponentů.

V druhé části se mantisy v pevné řádové čárce sečtou. Dále pokud dojde k přenosu v nejvyšším řádu, tak se přičte log. 1 k výslednému exponentu.

V posledním části se pracuje s prioritním kodérem a posuvem, aby mantisa byla v normalizovaném tvaru. V této části je využíváno zaokrouhlení podle čtyř nejnižších bitů mantisy.

4.1.1.3 Porovnání implementací sčítaček

V tabulce 4.1 jsou porovnány dvě popisované implementace a také další dvě, které jsou porovnávány v článku [10]. Hlavním rozdílem v implementacích je, že druhé řešení nepočítá se součtem čísel denormalizovaných, nekonečna a NaN. První řešení pracuje s kombinačními



Obrázek 4.2: Blokový diagram sčítačky/odčítačky [10]

	Řešení v 4.1.1.1	**Řešení v 4.1.1.2**	**Nallatech**	**Quixilica**
Počet stupňů pipeline	0	19	14	11
Obsazených řezů (slices)	281	551	290	306
Flip Flop	32	801	-	-
LUT	504	548	-	-
Hodinová frekvence (MHz)	59	250	184	169
Technologie	Virtex 4	Virtex 2Pro	-	-

Tabulka 4.1: Porovnání 32 bitových sčítaček v pohyblivé řádové čárce

obvody a druhé řešení ke kombinačním obvodům přidává ještě pipeline registry. První řešení pracuje pouze s 32 bitovými čísly, druhé pracuje jak s 32 bity tak 48 i 64 bity.

Ve vlastním návrhu využívám obou popisovaných implementací. Podle první implementace se věnuji všem možným vstupním hodnotám a to i speciálním případům. Jejich nepodporováním by se významně snížil komfort uživatele bloku pro FP operaci. V prvním návrhu využívám pro získání výsledků pouze kombinační bloky viz první zkoumaná implementace a v dalším návrhu zapojím pipeline registry tak, abych zvýšil rychlost a efektivnost navrhovaného bloku.

4.1.2 Algoritmus sčítaní

Tento algoritmus vychází z normy IEEE 754 popisovaném ve článcích [2, 6].

1. Na začátku budou z čísel x a y vyjmuty prvky: znaménko, mantisa a exponent. Prvním krokem je nalezení těchto prvků a správné uložení.
2. Nalézt pokud se jedná o speciální případy. To je, pokud se alespoň jedno z čísel rovná nule nebo nekonečnu a nebo NaN. Nastane-li tento případ, výsledek lze určit ihned v tomto kroku.
3. Rozhodnutí, zda čísla patří do kategorie normalizovaných, denormalizovaných (skrytý byt je roven nule a exponet je nula) a nebo jsou smíšené.
4. Podle do jaké kategorie patří čísla se postupuje takto:
   - U čísel normalizovaných se exponenty čísel porovnají a určí se větší číslo. Exponent většího čísla se bere v tuto chvíli jako výstupní. Mantisa nižšího čísla se posouvá o [E2-E1] bitů, aby se dorovnal rozdíl v exponentech.
   - Jsou-li čísla denormalizovaná, tak se provede porovnání mantis a znamének pro určení většího čísla. Výstupní exponent je roven nule.
   - Pokud jde o čísla smíšená, musí se určit denormalizované číslo a to se musí převést na normalizované číslo za pomoci rotace. A exponent se o tento počet rotací zvýší. A pak se postupuje jako s normalizovanými čísly.
5. Výpočet výstupního znaménka ze znaménka čísel a operace.
6. Sčítání a pokud nastane přetečení na nejvyšším bitu, musí proběhnout rotace a zvětšení exponentu. Detekuje se, zda nedošlo k přetečení tj. hodnota exponentu je rovna maximální hodnotě.
7. Standardizace (normalizace) mantisy za pomoci posuvu vlevo a snížení exponentu.
8. Proběhne zaokrouhlení k nejbližší hodnotě.
9. Detekce přetečení a podtečení exponentu (z předchozích kroků i z vypočtených výsledků). Pokud nastalo přetečení, je výsledkem nekonečno. Pokud exponent podtekl, je výsledkem nula nebo denormalizované číslo. Zapsání výsledku do formátu plovoucí řádové čárky.

Standardní sčítačky umožňují řešit i operaci odčítání. Odčítání je možné řešit dvěma způsoby. Prvním způsobem je implementace operace odčítaní v bloku, kde dochází k samotnému sčítání. Druhou možností je, že menšitel bude vynásoben hodnotou -1 a poté pomocí sčítačky přičten. Ve své architektuře budu odčítání provádět v samotném bloku sčítání.

4.1.3 Specifikace obvodu sčítačky

Obvod obsahuje čtyři úrovně pipeline registrů. První stupeň pipeline je na vstupních hodnotách. Další stupeň je po provedení kroků jedna až čtyři v popsaném algoritmu 4.1.2. Třetí úroveň pipeline je po krocích pět až osm. Poslední stupeň je po posledním kroku algoritmu tj. na výstupech z bloku. Rozmístění pipiline registrů je provedeno tak, aby byla snížena nejdelší kombinační cesta a tím také snížena minimální hodinová perioda.

Obvod pracuje tak, že v každém hodinovém cyklu přijímá vstupní hodnoty a výsledek vrací s latencí čtyř cyklů viz blokové schéma obvodu 5.1.

V tabulce 4.2 je popsáno rozhraní sčítačky.

Signál	Směr	Typ	Popis
numberA	vstup	STD_LOGIC_VECTOR ((widthMantissa+widthExponent) downto 0)	Vstupní číslo zadané ve formátu FP
numberB	vstup	STD_LOGIC_VECTOR ((widthMantissa+widthExponent) downto 0)	Vstupní číslo zadané ve formátu FP
add_sub	vstup	STD_LOGIC	Určuje zda se bude provádět součet (0) nebo rozdíl (1)
clk	vstup	STD_LOGIC	Hodinový signál
rst	vstup	STD_LOGIC	Reset obvodu
numberS	výstup	STD_LOGIC_VECTOR ((widthMantissa+widthExponent) downto 0)	Výstupní hodnota součtu/ rozdílu ve formátu FP
overflow	výstup	STD_LOGIC	Aktivní pokud nastalo přetečení
underflow	výstup	STD_LOGIC	Aktivní pokud nastalo podtečení

Tabulka 4.2: Rozhraní sčítačky

Detekce přetečení a podtečení

- Příznak `overflow` je nastaven, pokud hodnota exponentu je rovna nebo větší než maximální hodnota exponentu. Je-li tento příznak nastaven, je výsledkem hodnota $+\infty$ nebo $-\infty$. Při hodnotě NaN není příznak `overflow` nastaven.
- Příznak `underflow` je nastaven, pokud hodnota exponentu je menší než minimální hodnota exponentu nebo pokud se jedná o denormalizované hodnoty. Je-li exponent menší než minimální hodnota exponentu, je výsledkem hodnota 0 s příznakem `underflow`.

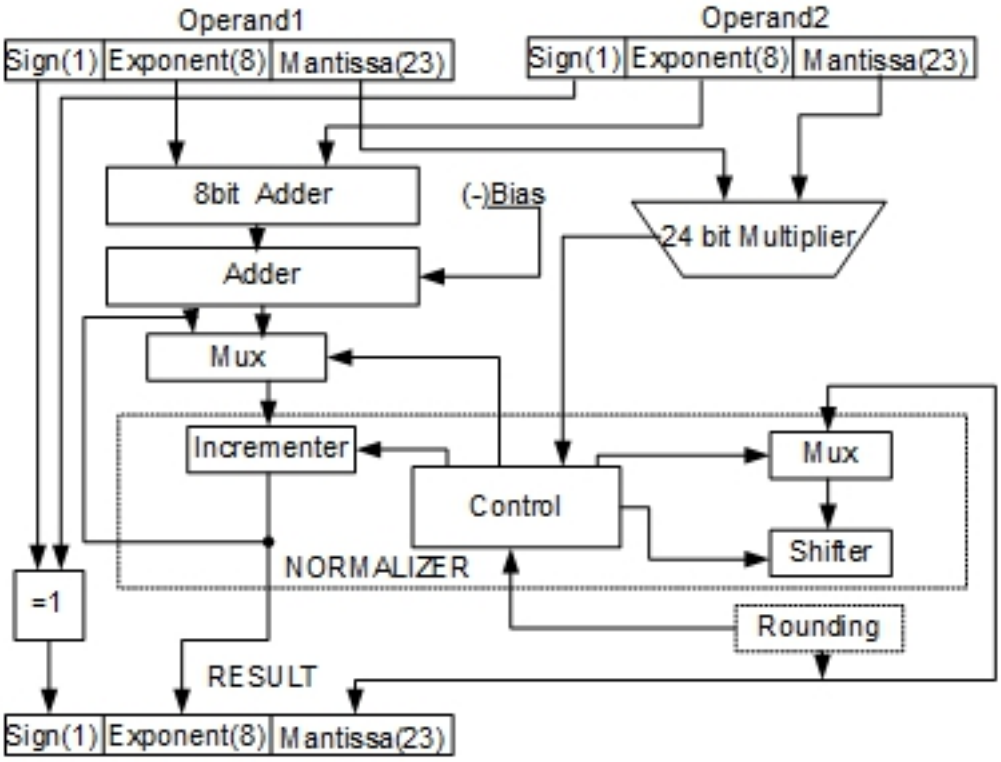
4.2 Násobička

4.2.1 Analýza existujících řešení

4.2.1.1 Popis řešení autorů L. Rajesh, Prashant. V. Joshi a Dr.S.S. Manvi

Implementace popisována v článku [12] používá stejný algoritmus jako popisovaný algoritmus v 3.2.5.1. Jediným rozdílem je, že algoritmus implementace neřeší násobení speciálních případů. Z popisu algoritmu v tomto článku je zřejmé, že vstupní hodnoty musí být v normalizovaném tvaru. Článek vysvětluje, jak snížit zpoždění způsobené vlivem přenosu bitu přenosu. Touto úpravou je, že se snaží počítat přenos v každé fázi výpočtu. Při násobení nezáporných čísel se vytváří částečné výsledky pro každou číslici v násobičce. Tyto hodnoty

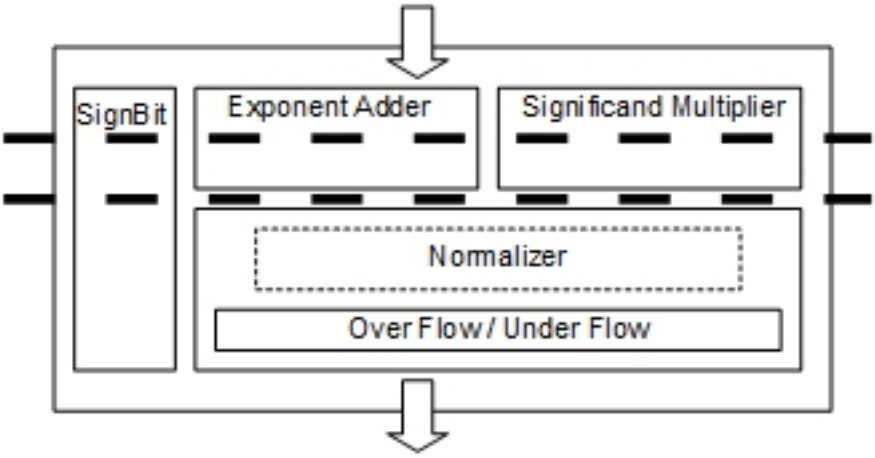
jsou nakonec sečteny, aby vznikl výsledek násobení, který může být až dvojnásobně široké číslo. Na obrázku 4.3 je blokové schéma této implementace. Řešení také popisuje, kam vložit pipeline, aby byla rozdělena nejdelší kritická cesta a tím zvýšena maximální pracovní frekvence. Podle obrázku 4.4 jsou pipeline umístěny takto: Ve středu násobičky mantisy a uprostřed sčítačky mantisy (před odečítáním hodnoty bias). Další úroveň je po dokončení operací násobení mantis a součet exponentů. Poslední pipeline registr je na výstupech z násobičky.



Obrázek 4.3: Blokový diagram násobičky [12]

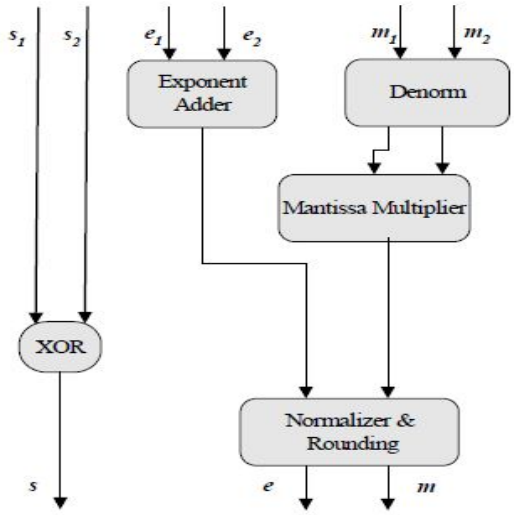
4.2.1.2 Popis řešení autorů G. Govindu, L. Zhuo, S. Choi a V. Prasanna

Ve článku [10] popisujícím násobičku ve FP formátu využívají algoritmus 3.2.5.1. Tato implementace neřeší násobení speciálních hodnot a denormalizovaných čísel. Blokové schéma násobičky 4.5 se dá rozdělit na tři části. První částí je denormalizace, která před vstupní hodnotu mantis vloží skrytý bit. Druhou částí je násobení už upravených mantis a sečtení exponentů. Při sečtení exponentů dojde také k odečítání hodnoty bias. Poslední třetí částí je normalizace a zaokrouhlení. Normalizace probíhá pomocí posuvu a poněvadž se pracuje



Obrázek 4.4: Násobička čísel ve formátu plovoucí řádové čárky s pipeline [12]

pouze s normalizovanými čísly, bude posuv nanejvýš o dva bity. Zaokrouhlení proběhne stejně jako u sčítačky.



Obrázek 4.5: Blokový diagram násobičky [10]

4.2.1.3 Porovnání implementací násobiček

Obě popisované implementace umožňují násobení čísel ve formátu plovoucí řádové čárky, ale ani jedna z implementací se nevěnuje násobení speciálních čísel a denormalizovaných hodnot. V tabulce 4.3 jsou základní údaje pro porovnání. Popisované implementace umožňují násobení 32 bitových čísel.

	Řešení v 4.2.1.1	**Řešení v 4.2.1.2**	**Nallatech**	**Quixilica**
Počet stupňů pipeline	3	8	6	6
Obsazených řezů (slices)	-	182	126	326
Flip Flop	288	324	-	-
Hodinová frekvence (MHz)	200	250	188	152
Technologie	Spartan 3	Virtex 2Pro	-	-

Tabulka 4.3: Porovnání 32 bitových násobiček v pohyblivé řádové čárce

Ve vlastním návrhu čerpám z těchto implementací z příkladu jak počítá implementovaná násobička. Pak dále využívám bloková schémata pro RTL návrhy.

4.2.2 Algoritmus násobení

Tento algoritmus vychází z normy IEEE 754 popisovaném ve článcích [2, 6].

1. Na začátku budou z čísel x a y vyjmuty prvky: znaménko, mantisa a exponent. Prvním krokem je nalezení těchto prvků a správné uložení.

2. Dalším krokem je nalézt speciální případy. Pokud alespoň jedno číslo je nula nebo nekonečno a nebo NaN, bude výstup určen už v tomto kroku a to podle tabulky 3.5.

3. Znaménko výsledku vznikne operací nonekvivalence znaménkových bitů vstupních čísel.

4. Mantisa výsledku vznikne součinem mantis vstupních čísel. U tohoto kroku se musí násobit mantisy i se skrytým bitem. Proto se musí určit, zda číslo není denormalizované, jinak by byl skrytý bit roven nule. K výsledku součinu se určí konečné umístění čárky výsledku v mantise.

5. Exponent výsledku vznikne součtem exponentů a odečtením hodnoty bias. Detekuje se, zda nedošlo k přetečení nebo podtečení. To nastává, pokud hodnota exponentu je větší nebo rovna maximální hodnotě nebo naopak, pokud je nižší než minimální hodnota exponentu.

6. V tomto kroku proběhne normalizace zjištěných výsledků tj. upravení mantisy tak, aby první bit před čárkou byl roven jedné. Pokud řádová čárka v mantise není ihned za první jedničkou zleva, musí probíhat rotace. Pokud se rotuje vpravo (jednička je před čárkou, ale je tam i další hodnota), exponent se zvyšuje. Rotuje se vlevo (před čárkou není jednička pouze nuly), tím se exponent snižuje dokud se hodnota exponentu nedostane na minimum tj. vznikne denormalizované číslo a nebo se při rotaci narazí na jedničku.

7. Proběhne zaokrouhlení k nejbližší hodnotě.

8. Detekce přetečení a podtečení exponentu (z předchozích kroků i z vypočtených výsledků). Pokud nastalo přetečení, je výsledkem nekonečno. Pokud exponent podtekl, je výsledkem nula nebo denormalizované číslo. Zapsání výsledku do formátu plovoucí řádové čárky.

4.2.3 Specifikace obvodu násobičky

Stejně jako sčítačka má obvod čtyři stupně pipeline registrů. První pipeline je na vstupních hodnotách. Další stupeň pipeline je po prvních třech krocích algoritmu 4.2.2. Třetí úroveň pipeline je po čtvrtém a pátém kroku algoritmu. Poslední pipeline registry jsou na výstupech z obvodu tj. po provedení zbylých kroků algoritmu. Rozmístění pipiline registrů je provedeno tak, aby byla snížena nejdelší kombinační cesta a tím také snížena minimální hodinová perioda.

Obvod pracuje tak, že v každém hodinovém cyklu přijímá vstupní hodnoty a výsledek vrací s latencí čtyř cyklů viz blokové schéma obvodu 5.2.

V tabulce 4.4 je popsáno rozhraní násobičky.

Signál	**Směr**	**Typ**	**Popis**
numberA	vstup	STD_LOGIC_VECTOR ((widthMantissa+widthExponent) downto 0)	Vstupní číslo zadané ve formátu FP
numberB	vstup	STD_LOGIC_VECTOR ((widthMantissa+widthExponent) downto 0)	Vstupní číslo zadané ve formátu FP
clk	vstup	STD_LOGIC	Hodinový signál
rst	vstup	STD_LOGIC	Reset obvodu
numberM	výstup	STD_LOGIC_VECTOR ((widthMantissa+widthExponent) downto 0)	Výstupní hodnota násobení ve formátu FP
overflow	výstup	STD_LOGIC	Aktivní pokud nastalo přetečení
underflow	výstup	STD_LOGIC	Aktivní pokud nastalo podtečení

Tabulka 4.4: Rozhraní násobičky

Detekce přetečení a podtečení je totožná jako u sčítačky viz 4.1.3.

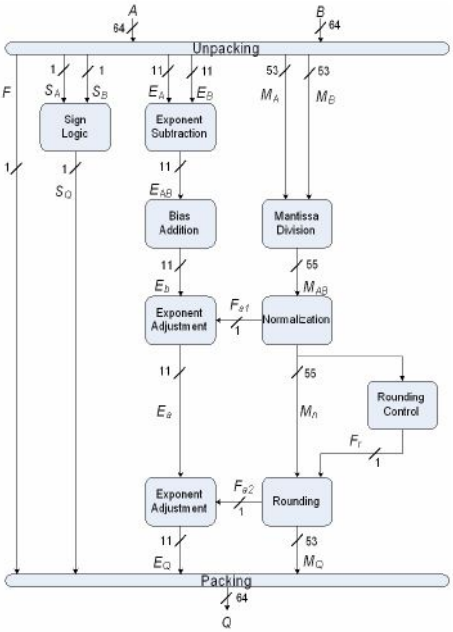
4.3 Dělička

4.3.1 Analýza existujících řešení

4.3.1.1 Popis řešení autorů Anuja J. Thakkar a Abdel Ejnioui

Článek [13] na obrázku 4.6 popisuje implementovanou děličku. Postup probíhá obdobně jako v popisovaném obecném algoritmu 3.2.5.3. Implementace v první fázi v obou číslech detekuje jednotlivé části formátu plovoucí řádové čárky a podle hodnot určí, zda se nejedná

o speciální hodnoty nebo nulu. Pokud ano, je nastavený příznak F a výpočet je ukončen. Pro ostatní případy probíhá výpočet. Proběhne výpočet znaménka pomocí operace XOR, odečtení exponentů a přičtení hodnoty bias. V další části proběhne dělení mantis a na závěr výsledek bude normalizován, zaokrouhlen a jednotlivé části formátu jsou vráceny do jedné posloupnosti čísel.



Obrázek 4.6: Dělička s přesností double ve formátu plovoucí řádové čárky [13]

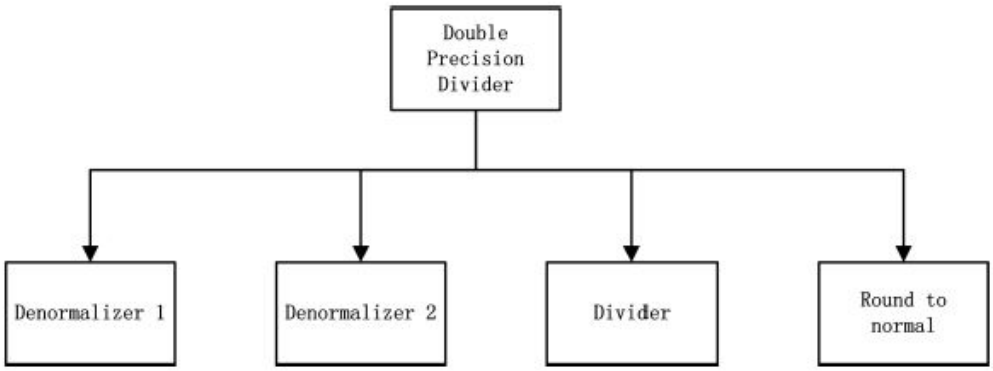
4.3.1.2 Popis řešení autorů Xin Fang a Miriam Leeser

Článek [14] popisující druhou implementaci děličky využívá obecně popisovaný algoritmus pro dělení 3.2.5.3. Tato implementace pro určení výsledku dělení mantis nejprve vypočte převrácenou hodnotu dělitele a poté vynásobí hodnotou dělence. Tento přístup je založen na rozšířené Taylorově řadě využívající LUT (vyhledávacích tabulek) a násobiček což jsou zdroje snadno přístupné v FPGA obvodech.

Blokové schéma 4.7 zobrazuje nejvyšší úroveň propojení komponent děličky. Bloky `Denormalizer` mají za úkol přidat k mantise skrytý bit jedničky, poněvadž tato implementace předpokládá, že vstupní hodnoty jsou v normalizovaném tvaru. Blok `Divider` má provádí výpočet znaménka, exponentu a mantisy. Poslední blok `Round to Normal` zaokrouhlí podle jedné metody ze čtyř podle IEEE standardu a poté vrátit normalizovanou hodnotu čísla ve formátu plovoucí řádové čárky.

4.3.1.3 Porovnání implementací děliček

V tabulce 4.5 jsou základní parametry implementovaných děliček v přesnosti double ve formátu plovoucí řádové čárky. První implementace reaguje i na speciální hodnoty vstupu,



Obrázek 4.7: Propojení komponent děličky [14]

ale nereaguje na denormalizované hodnoty. Druhá implementace počítá pouze s normalizovanými hodnotami a ostatní vstupy neřeší.

	Řešení v 4.3.1		**Řešení v 4.3.1.2**	**IP Core from Xilinx**
	Sekvenční dělička	Pipeline dělička		
Počet stupňů pipeline	-	-	14	8
Obsazených řezů (slices)	222	2915	6957	3210
Flip Flop	274	5734	934	722
Hodinová frekvence (MHz)	65,2	67,1	148	74
Technologie	Virtex XCV1000		Virtex 6	Virtex 6

Tabulka 4.5: Porovnání 64 bitových děliček v pohyblivé řádové čárce

Pro svou implementaci jsem se nechal inspirovat hlavně řešením autorů Anuja J. Thakkar a Abdel Ejnioui.

4.3.2 Algoritmus dělení

1. Na začátku budou z čísel x a y vyjmuty prvky: znaménko, mantisa a exponent. Prvním krokem je nalezení těchto prvků a správné uložení.

2. Dalším krokem je nalézt speciální případy. Pokud alespoň jedno číslo je nula nebo nekonečno a nebo NaN, bude výstup určen už v tomto kroku.

3. Znaménko výsledku vznikne operací nonekvivalence znaménkových bitů vstupních čísel.

4. Proběhne výpočet počtu nul před první jedničkou vlevo v obou mantisách. Mantisy v tomto kroku jsou i se skrytým bitem.

5. Exponent je vypočten tak, že exponenty jsou odečteny a je připočten bias. Dále je přičten počet míst před první jedničkou v mantise dělence a odečten počet míst před první jedničkou v mantise dělitele. Detekuje se, zda nedošlo k přetečení nebo podtečení. To nastává, pokud hodnota exponentu je větší nebo rovna maximální hodnotě nebo naopak, pokud je nižší než minimální hodnota exponentu.

6. Obě mantisy i se skrytými bity jsou zarovnány rotací a je provedeno děleni.

7. V tomto kroku proběhne normalizace zjištěných výsledků tj. upravení mantisy tak, aby první bit byl roven jedné za pomoci posuvu vlevo a snížení exponentu.

8. Proběhne kontrola na přetečení a podtečení (z předchozích kroků i z vypočtených výsledků) a je vypsán výsledek ve formátu plovoucí řádové čárky.

4.3.3 Specifikace obvodu děličky

Obvod je řízen stavovým automatem (čítačem), který je spuštěn po aktivní hodnotě signálu `start`. Po proběhnutí výpočtu je čítačem nastaven signál `done`, který označuje dokončení výsledku a obvod je ve stavu vyčkávání na další započetí výpočtu. Délka výpočtu je závislá na vstupních datech. V případě speciálních hodnot se jedná o jeden hodinový cyklus a nebo u výpočtu s čísly (u kterých je nutné dělení mantis) je délka výpočtu rovna součtu počtu bitů mantisy, skrytého bitu, čtyř zaokrouhlovacích bitů a režii, která je dva hodinové cykly.

V tabulce 4.6 je popsáno rozhraní děličky.

Detekce přetečení a podtečení je totožná jako u sčítačky viz 4.1.3.

4.4 Konverze z fixed point čísla

4.4.1 Algoritmus konverze

1. Kontrola, zda se nejedná o nulovou hodnotu. Pokud ano, jsou všechny části FP čísla nulové (FX formát nemá zápornou nulu).

2. Zjištění znaménka vstupního čísla. Je vypočítána odpovídající absolutní hodnota vstupního čísla.

3. Nalezení první jedničky v absolutní hodnotě čísla. Určení počtu rotací vlevo, aby na první pozici byla hodnota jedna.

4. Ověření, zda je dané číslo zobrazitelné tj. zda počet rotací není vyšší než hodnota součtu bias a šířky představující celou část čísla. Pokud by tato situace nastala, vznikne denormalizované číslo nebo nulová hodnota.

5. Určení hodnoty exponentu. Exponent je roven součtu hodnoty bias a šířky představující celou část vstupní hodnoty. Tato hodnota je snížena o hodnotu počtu rotací (hodnota nemůže být vyšší než součet viz předchozí krok).

Signál	Směr	Typ	Popis
numberA	vstup	STD_LOGIC_VECTOR ((widthMantissa+widthExponent) downto 0)	Vstupní číslo zadané ve formátu FP
numberB	vstup	STD_LOGIC_VECTOR ((widthMantissa+widthExponent) downto 0)	Vstupní číslo zadané ve formátu FP
clk	vstup	STD_LOGIC	Hodinový signál
rst	vstup	STD_LOGIC	Reset obvodu
start	vstup	STD_LOGIC	Start výpočtu. Pokud je dělička neaktivní započte tímto signálem výpočet
done	výstup	STD_LOGIC	Dokončení výpočtu. Pokud je výpočet dokončen je tento signál v log. 1 po dobu jednoho cyklu
numberD	výstup	STD_LOGIC_VECTOR ((widthMantissa+widthExponent) downto 0)	Výstupní hodnota děleni ve formátu FP
overflow	výstup	STD_LOGIC	Aktivní pokud nastalo přetečení
underflow	výstup	STD_LOGIC	Aktivní pokud nastalo podtečení

Tabulka 4.6: Rozhraní děličky

6. Mantisa (se skrytým bitem) vznikne rotací absolutní hodnoty vlevo podle pozice první jedničky.
7. Zaokrouhlení k nejbližší hodnotě.
8. Ověření, zda nedošlo k přetečení při zaokrouhlování. Nastane-li přetečení dojde k zvýšení exponentu.
9. Kontrola, zda exponent není na maximální hodnotě. Pokud ano, mantisa je nulována a výstupem musí být hodnota nekonečna.
10. Složení jednotlivých částí do výsledného formátu pohyblivé řádové čárky.

4.4.2 Specifikace obvodu pro konverzi z fixed point čísla

Obvod převádí čísla ve formátu pevné řádové čárky do formátu pohyblivé řádové čárky. Záporné vstupní hodnoty jsou převáděny, pokud je číslo zakódováno v doplňkovém kódu. Vstupní hodnoty, které jsou nad rozsah FP výstupu jsou převedeny do hodnoty kladného nebo záporného nekonečna. Naopak hodnoty, které jsou tak malé, že je FP formát nemůže zobrazit, převede obvod do denormalizovaného tvaru a pokud nejde ani toto, tak je výstupem nula. Šířku vstupního i výstupního čísla je možné nastavit pomocí generických konstant. Omezením je, že šířka FX čísla je minimálně široká jako mantisa FP výstupní hodnoty. Obvod obsahuje dva stupně pipeline. První stupeň pipeline je na vstupních hodnotách. Druhá úroveň je na výstupech z obvodu.

V tabulce 4.7 je popsáno rozhraní obvodu pro konverzi z FX do FP.

Signál	Směr	Typ	Popis
numberFX	vstup	STD_LOGIC_VECTOR ((widthM + widthN) downto 0)	Vstupní číslo zadané ve formátu FX (QwidthM.widthN)
clk	vstup	STD_LOGIC	Hodinový signál
rst	vstup	STD_LOGIC	Reset obvodu
numberFP	výstup	STD_LOGIC_VECTOR ((widthMantissa+widthExponent) downto 0)	Výstupní hodnota FP převedená z FX

Tabulka 4.7: Rozhraní konverze z fixed point čísla

4.5 Konverze z floating point čísla

4.5.1 Algoritmus konverze

1. Na začátku je ze vstupní hodnoty čísla zjištěno znaménko, exponent a mantisa.
2. Proběhne kontrola, zda se nejedná o hodnotu nula. Pokud ano, je vytvořen výstup s hodnotou nula.
3. Podle exponentu je vložen skrytý bit na na nejvyšší bit absolutní hodnoty (je-li exponent roven nule je skrytý bit nula, jinak jedna). Za skrytý bit je vložena hodnota mantisy.
4. Z exponentu se vypočte hodnota rotace takto: odečtením hodnoty bias a šířky celočíselné části FX čísla. Je-li hodnota vyšší než nula, došlo k přetečení a je nastavena maximální absolutní hodnota výstupního FX čísla. Pokud je absolutní hodnota rotace vetší než šířka FX výstupního čísla, došlo k podtečení a výstupem je nula.
5. Pokud bylo znaménko záporné, je hodnota převedena do doplňkového kódu a to je výstupem. Jinak je výstup roven absolutní hodnotě.

4.5.2 Specifikace obvodu pro konverzi z floating point čísla

Obvod převádí čísla ve formátu pohyblivé řádové čárky do formátu pevné řádové čárky. Záporné vstupní hodnoty jsou převáděny do doplňkového kódu. Vstupní hodnoty, které jsou nad rozsah FX výstupu jsou převedeny na maximální hodnotu FX výstupu s adekvátním znaménkem. Naopak hodnoty, které jsou tak malé, že je FX formát nemůže zobrazit, převede obvod na číslo s hodnotou nula. Šířku vstupního i výstupního čísla je možné nastavit pomocí generických konstant. Omezením je, že šířka FX čísla je minimálně široká jako mantisa FP výstupní hodnoty. Obvod obsahuje dva stupně pipeline. První stupeň pipeline je na vstupních hodnotách. Druhá úroveň je na výstupech z obvodu.

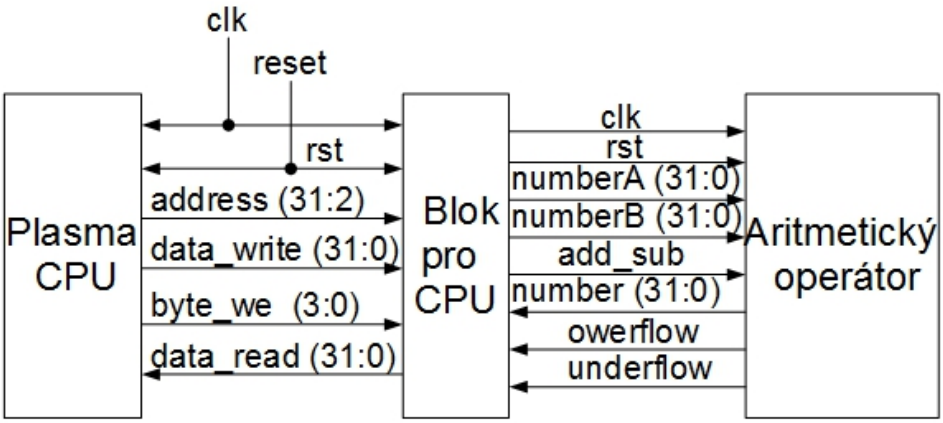
V tabulce 4.7 je popsáno rozhraní obvodu pro konverzi z FP do FX.

Signál	Směr	Typ	Popis
numberFP	vstup	STD_LOGIC_VECTOR ((widthMantissa+widthExponent) downto 0)	Vstupní číslo zadané ve formátu FP
clk	vstup	STD_LOGIC	Hodinový signál
rst	vstup	STD_LOGIC	Reset obvodu
numberFX	výstup	STD_LOGIC_VECTOR ((widthM + widthN) downto 0)	Výstupní hodnota FX (QwidthM.widthN) převedená z FP

Tabulka 4.8: Rozhraní konverze z floating point čísla

4.6 Integrace s procesorem Plasma

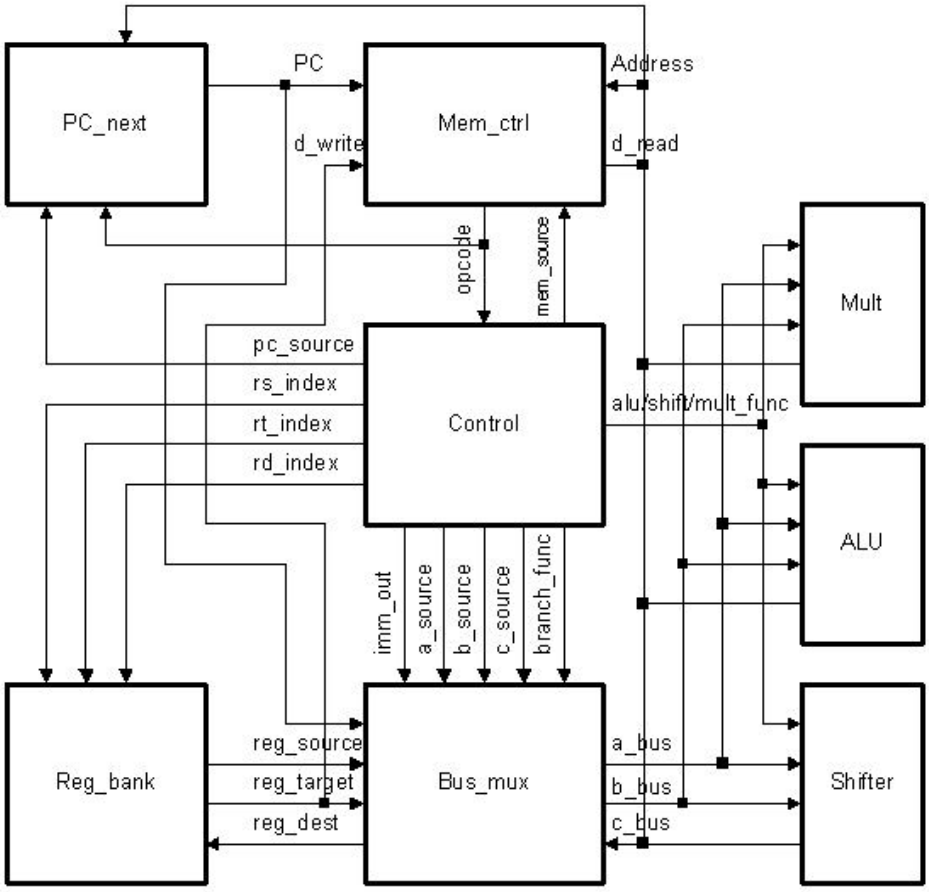
Každý navrhovaný blok je připojen k procesoru přes obvod, který spravuje práci mezi procesorem a aritmetickým obvodem viz obrázek (pro sčítačku) 4.8. Obvod detekuje zápis a čtení na adrese patřící připojenému aritmetickému operátoru a patřičně na tuto akci reaguje zapsáním dat obvodu nebo předáním požadovaných dat procesoru. Stejný obvod je využíván i pro konverzi na a z fixed point čísla. Drobným rozdílem jsou vstupní a výstupní signály pro obvod konverze (to i pro jednotlivé aritmetické bloky).



Obrázek 4.8: Blokové schéma připojení obvodu aritmetické operace k procesoru Plasma

4.6.1 Procesor Plasma

Procesor Plasma je 32 bitový RISC procesor. Vykonává všechny MIPS I(TM) instrukce kromě "unaligned load and store operations" [18]. Procesor je implementovaný ve VHDL buď s dvoustupňovým nebo třístupňovým pipeline. Jádrem procesoru je blok `mlite_cpu`, který má blokové schéma na obrázku 4.9.



Obrázek 4.9: Blokové schéma procesoru Plasma [18]

4.6.2 Kompilace programu

Kompilace programu je prováděna pomocí dávkového souboru `makefile`, který definuje zdrojové soubory pro vytvoření spustitelného souboru. Soubory jsou poté převedeny do textového souboru `code.txt`, který obsahuje instrukce převedené do hexadecimálního kódu. Pod textem je uvedena část souboru `makefile`, která provádí kompilaci programu ukazující funkčnost sčítání.

```
adder:
$(AS_MIPS) -o boot.o boot.asm
$(GCC_MIPS) adder.c
$(GCC_MIPS) no_os.c
$(LD_MIPS) -Ttext 0 -e entry -Map test.map -s -N -o test.axf \
boot.o adder.o no_os.o
-$(DUMP_MIPS) --disassemble test.axf > test.lst
$(CONVERT_BIN)
$(CP) code.txt $(VHDL_DIR)
```

Kapitola 5

RTL návrh

Kapitola je zaměřena na popis jednotlivých bloků implementovaných obvodů. Dále jsou zde popsané připojovací bloky pro procesor Plasma.

5.1 Sčítačka

5.1.1 Popis bloků

Celý obvod je rozdělen do pěti základních bloků. Jedná se o bloky: `n_case`, `pre_adder`, `block_adder`, `block_norm` a `block_result`. Všechny bloky pracují s generickými konstantami, které umožňují, aby vstupní a výstupní signály měly různé šířky podle šířky vstupních hodnot.

Blok n_case

Blok `n_case` má vstupem dvě čísla `numberA`, `numberB` již popsané v tabulce 4.2 a signál `add_sub` určující operaci sčítaní/odčítání. Výstupem je číslo `numberS` stejného formátu jako jsou vstupní čísla a signál `enable`, který zprovozňuje sčítací blok. Blok ze vstupních hodnot zjistí znaménko, exponent a mantisu. Dále proběhne identifikace do jaké kategorie patří jednotlivá čísla. Možné kategorie jsou: nula, denormalizované, normalizované, nekonečno a NaN. Signál `enable` je aktivní, když obě čísla jsou normalizovaná nebo denormalizovaná. Pokud se jedná o ostatní možné typy čísel, tak tento obvod vypočte výslednou hodnotu a nastaví ji do `numberS` a v tomto případě bude signál `enable` v log. 0. Jak obvod reaguje na vstupní typy čísel popisuje tabulka 5.1. Aby výstupy byly korektní i pro odčítání musí se znaménko pro číslo B měnit podle typu operace. Blok implementuje první až třetí krok algoritmu 4.1.2.

Blok pre_adder

Úkolem bloku je příprava dat pro sčítačku. Vstupem bloku jsou čísla `numberA` a `numberB` (vstupní čísla obvodu sčítačky), typ čísla A a B a signál `enable`. Výstupem jsou znaménka A a B, mantisy A a B, společný exponent a bit, který říká jaké číslo je větší. V tomto bloku podle toho o jaké čísla se jedná, proběhne porovnání exponentů a posuv mantis, aby mohl v dalším bloku dojít k součtu mantis. Mohou nastat tři situace popsané v algoritmu sčítání. Pro denormalizované čísla se porovnají mantisy a určí se vyšší hodnota. Pro normalizované hodnoty musí proběhnout porovnání exponentů a podle rozdílu hodnot

Znaménka čísel A a B	**Číslo A**	**Číslo B**	**Znaménko výstupu**	**Číslo S (výstup)**	**Enable (výstup)**
Nepodstatné	Nula	Číslo B	Znaménko čísla B	Číslo B	0
Nepodstatné	Číslo A	Nula	Znaménko čísla A	Číslo A	0
Nepodstatné	∞	Číslo B	Znaménko čísla A	∞	0
Nepodstatné	Číslo A	∞	Znaménko čísla B	∞	0
Se shodují	∞	∞	Znaménko čísla A i B	∞	0
Jsou rozdílná	∞	∞	1	NaN	0
Nepodstatné	NaN	Číslo B	1	NaN	0
Nepodstatné	Číslo A	NaN	1	NaN	0
Nepodstatné	Číslo A	Číslo B	Není určeno	Není určeno	1

Tabulka 5.1: Závislost výstupů na vstupech v bloku n_case sčítačky (podle[6])

exponentů bude probíhat rotace mantisy čísla s menší hodnotou exponentu. Pro smíšené hodnoty čísel se upraví denormalizované číslo tak, aby bylo normalizované. To znamená, že mantisa rotuje vlevo, dokud není na prvním nejvyšším bitu hodnota jedna a současně se zvyšuje exponent o jedna. Pak proběhne srovnání exponentů totožně jak u normalizovaných hodnot. Blok `pre_adder` je z tohoto důvodu rozdělen na další tři bloky podle toho o jaká čísla se jedná. Toto rozdělení je lepší pro syntézu a také pro přehlednost. Blok implementuje první a čtvrtý krok algoritmu 4.1.2.

Blok block_adder

Vstupem pro tento blok jsou výstupy z bloku `pre_adder` a navíc signál `add_sub`, který určuje požadovanou operaci. Blok určuje znaménko operace a také výsledné znaménko viz tabulka 5.2. Po určení znaménka v bloku probíhá operace odpovídající tomuto znaménku. Pokud při součtu/odečtu dojde k přenosu na nejvyšším bitu je tato informace předávána do dalšího bloku pomocí signálu `carry`. Blok implementuje pátý krok a část šestého kroku algoritmu 4.1.2.

Blok block_norm

Blok normalizace provede rotaci doprava a přidá jedničku na nejvyšší bit, pokud nastal přenos při sčítaní. Při této operaci se také zvýší o jedna exponent. Nenastane přenos a na nejvyšším bitu není log. 1, musí se provádět rotace vlevo, aby na nejvyšším bitu byla jednička. Exponent se sníží o počet rotací. V bloku také proběhne zaokrouhlení. Zaokrouhluje se k nejbližší zobrazitelné hodnotě. Pokud guard bity jsou větší než hodnota "1000", přičte se k mantise jedna. Pokud jsou menší než hodnota "1000" odříznou se pouze guard bity a mantisa se nezmění. Jestli se guard bity rovnají hodnotě "1000" a bit rozšířené mantisy na pozici 4 je roven jedné, tak se k hodnotě mantisy přičte jedna. Zde probíhá kontrola na přetečení. Přetečení je v tomto místě detekováno pokud se hodnota exponentu dostane na svoji maximální hodnotu. Podtečení v tomto bloku nemusí být detekováno, poněvadž

Znaménko A	Vstupní operace	Znaménko B	Výsledná operace	Výsledné znaménko	Podmínka
+	+	+	+	+	
+	+	-	-	+	A > B
+	+	-	-	-	B > A
+	-	+	-	+	A > B
+	-	+	-	-	B > A
+	-	-	+	+	
-	+	+	-	-	A > B
-	+	+	-	+	B > A
-	+	-	+	-	
-	-	+	+	-	
-	-	-	-	-	A > B
-	-	-	-	+	B > A

Tabulka 5.2: Výstupní znaménko a znaménko operace podle vstupních znamének v bloku block_adder

operace v tomto bloku exponent nemohou odečíst pod jeho minimální hodnotu. Detekování podtečení na denormalizovaných číslech je až v následujícím bloku. Blok implementuje zbytek šestého kroku a sedmý krok algoritmu 4.1.2.

Blok block_result

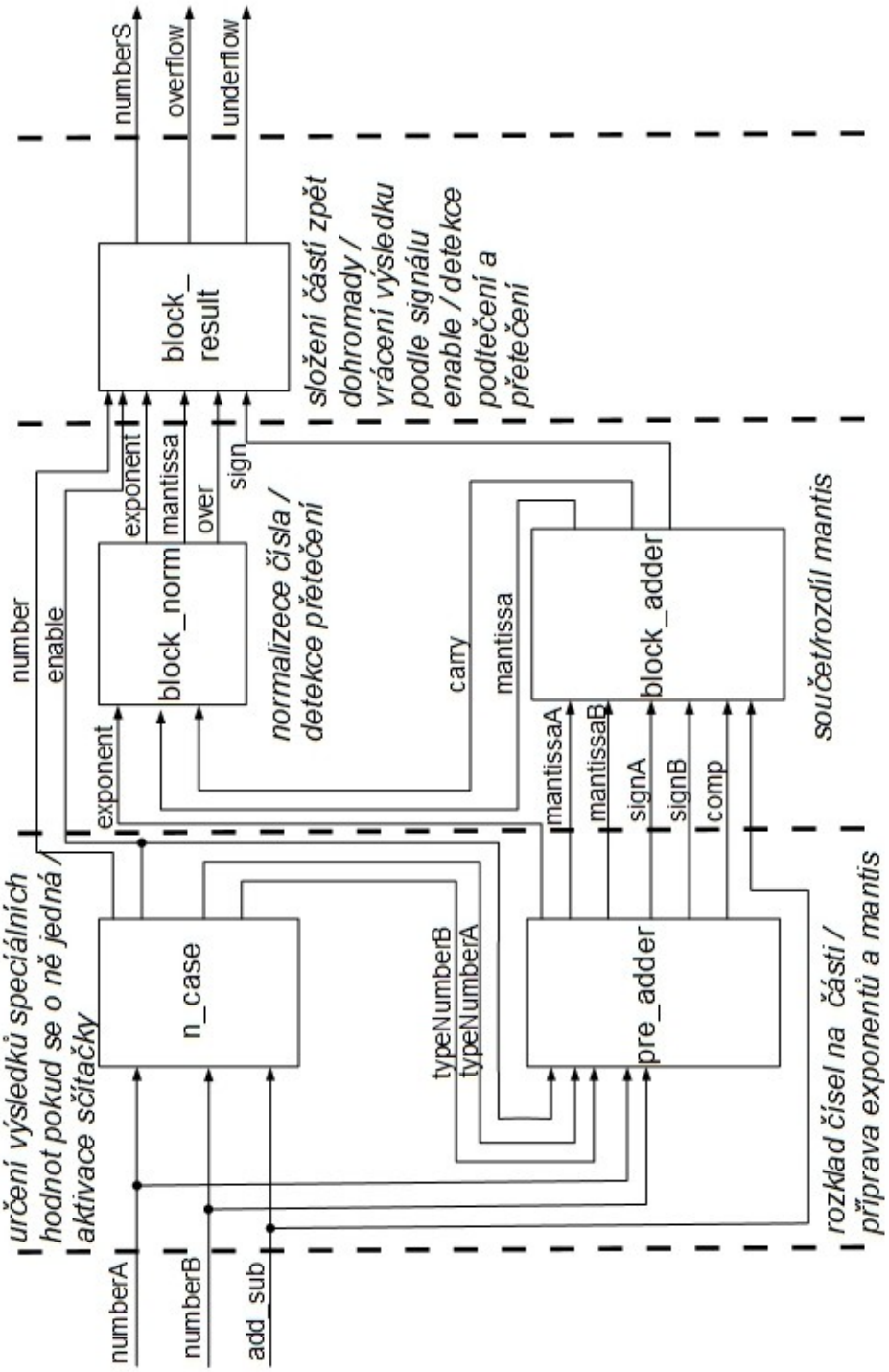
Poslední blok má za úkol podle vstupu `enable` vrátit celý výsledek ve FP formátu a také identifikovat příznaky podtečení a přetečení. Pokud je `enable` roven log. 1, spojí se výsledek ze sčítačky do jednoho čísla, jinak je výsledek z bloku `n_case`. Pokud se jedná o data z bloku `n_case`, jsou příznaky detekovány pouze podle kombinací maximálních a minimálních hodnot exponentů a mantis. Jedná-li se o data ze sčítačky, tak se sleduje příznak přetečení z předchozího bloku. Pokud je tento příznak v log. 0, proběhne kontrola, zda se jedná o denormalizované hodnoty. Pokud ano nastaví se podtečení. Výstupem tohoto bloku je hodnota součtu/rozdílu, příznak podtečení a přetečení popsané v tabulce 4.2. Blok implementuje devátý krok algoritmu 4.1.2.

Propojení bloků

Jednotlivé bloky jsou propojeny podle obrázku 5.1. V obrázku jsou čárkovaně zakreslena místa, kde jsou vloženy pipeline registry. V obrázku není zakreslen signál `clk` a `rst`. Signál `clk` je zaveden do každého bloku a každého pipeline registru. Signál `rst` je zaveden do pipeline registrů pro reset odvodu.

5.1.2 Integrace s procesorem Plasma

Sčítačka je integrována jako periferie mapovaná do adresního prostoru procesoru. V tabulce 5.3 je namapování adresového prostoru v procesoru Plasma. Prostor na adrese `0x60000010` převede nahranou hodnotu ve FX formátu do FP formátu na adresu `0x60000020`. Tento převod funguje i obráceným směrem a to tak, že nahraná data v FP formátu na adresu `0x60000020` jsou převedena na FX formát na adresu `0x60000010`. Tento převod je prováděn za pomoci balíčku `float_pkg`, který je v knihovně `ieee_proposed` (`ieee`).



Obrázek 5.1: RTL schéma sčítačky

Adresa	Význam
0x00000000 - 0x0000ffff	Vnitřní RAM (8KB)
0x10000000 - 0x100fffff	Vnější RAM (1MB)
0x20000000	Uart Write Uart Read
0x20000010	IRQ Mask
0x20000020	IRQ Status
0x20000030	GPIO0 Out Set bits
0x20000040	GPIO0 Out Clear bits
0x20000050	GPIOA In
0x20000060	Counter
0x20000070	Ethernet transmit count
0x60000010	FX hodnota
0x60000020	FP hodnota
0x60000030	Vstupní hodnota A FP sčítačky
0x60000040	Vstupní hodnota B FP sčítačky
0x60000050	Výstupní hodnota FP sčítačky
0x60000060	Registr FP sčítačky
0x60000070	Vstupní hodnota A FP násobičky
0x60000080	Vstupní hodnota B FP násobičky
0x60000090	Výstupní hodnota FP násobičky
0x600000A0	Registr FP násobičky
0x600000B0	Vstupní hodnota A FP děličky
0x600000C0	Vstupní hodnota B FP děličky
0x600000D0	Výstupní hodnota FP děličky
0x600000E0	Registr FP děličky
0x60000100	Vstupní hodnota FX čísla konvertoru z FX do FP
0x60000110	Výstupní hodnota FP čísla konvertoru z FX do FP
0x60000120	Registr konvertoru z FX do FP
0x60000100	Vstupní hodnota FP čísla konvertoru z FP do FX
0x60000110	Výstupní hodnota FX čísla konvertoru z FP do FX
0x60000120	Registr konvertoru z FP do FX

Tabulka 5.3: Mapovaní adresového prostoru procesoru Plasma

Pro integraci s procesorem Plasma je vytvořen blok, který spojuje rozhraní procesoru a sčítačky viz obrázek 4.8. V tabulce 5.4 jsou popsány vstupní a výstupní signály bloku. Tento blok detekuje zápis na paměťová místa pro sčítačku. Pracuje s registrem příznaků viz tabulka 5.5, který má pět bitů a pracují takto:

- Nahráním log. 1 do nultého bitu `start` započne výpočet na nahraných vstupních operandech.
- Bit `done` je nastaven do log. 1 pokud je dokončen výpočet a data jsou na výstupu. Bit je nulován po přečtení paměťového místa s výsledkem součtu/rozdílu.

- Třetí bit `overflow` detekuje přetečení ve formátu plovoucí řádové čárky.
- Bit `underflow` detekuje podtečení ve formátu plovoucí řádové čárky.
- Pátý bit `add_sub` určuje, zda se bude sčítat nebo odčítat. Pokud je nastavena log. 0 obvod sčítá, jinak se odčítá.

Při resetu obvodu jsou všechny hodnoty registru nastaveny do log. 0.

Výpočty pomocí sčítačky mohou probíhat ve dvou režimech. V prvním režimu je po nahraní log. 1 do bitu `start` spuštěn výpočet a dokončení je detekováno bitem `done`.

Druhý režim nevyužívá `start` a `done` bitu. Uživatel na vstupy sčítačky nahrává hodnoty čísel a s latencí čtyř hodinových cyklů očekává na zadané hodnoty výsledek součtu.

Signál	**Směr**	**Typ**	**Popis**
address	vstup	STD_LOGIC_VECTOR (31 downto 2)	Adresa pro zápis a čtení z paměťových míst určených pro sčítačku
data_write	vstup	STD_LOGIC_VECTOR (31 downto 0);	Sběrnice pro vstupní data
byte_we	vstup	STD_LOGIC_VECTOR (3 downto 0)	Detekce zápisu dat
clk	vstup	STD_LOGIC_VECTOR	Hodinový signál
rst	vstup	STD_LOGIC_VECTOR	Reset obvodu
data_read	výstup	STD_LOGIC_VECTOR (31 downto 0)	Sběrnice pro výstupní data

Tabulka 5.4: Rozhraní sčítačky s procesorem Plasma

bit 4	bit 3	bit 2	bit 1	bit 0
add/sub	**underflow**	**overflow**	**done**	**start**
R/W	R	R	R	R/W

Tabulka 5.5: Registr pro řízení sčítání s procesorem Plasma

5.2 Násobička

5.2.1 Popis bloků

Celý obvod je rozdělen do pěti základních bloků. Jedná se o bloky: `n_case`, `expAdder`, `mantissaMul`, `block_norm` a `block_result`. Všechny bloky pracují s generickými konstantami, které umožňují, aby vstupní a výstupní signály měly různé šířky podle šířky vstupních hodnot.

Blok n_case

Blok má za úkol odfiltrovat speciální typy násobení a pokud se o ně jedná, vrátí adekvátní výsledek signálem `number`. Pokud se nejedná o násobení se speciálními typy čísel, tak blok

určí hodnotu výsledného znaménka a oddělí ze vstupů exponenty a mantisy. Výsledné znaménko je vždy určeno pomocí operace exkluzivního logického součtu (XOR). Výstup `enable` je aktivní, pokud se nejedná o násobení se speciálními typy čísel a mají se aktivovat bloky pro zpracovaní normalizovaných a denormalizovaných čísel. V tabulce 5.6 je vidět závislost výstupů na vstupech v bloku `n_case`. Blok implementuje první až třetí krok algoritmu 4.2.2.

Číslo A	**Číslo B**	**Číslo S (výstup)**	**Enable (výstup)**
nenulová kladná	$+\infty$	$+\infty$	0
nenulová kladná	$-\infty$	$-\infty$	0
nenulová záporná	$+\infty$	$-\infty$	0
nenulová záporná	$-\infty$	$+\infty$	0
nula	$+\infty$	NaN	0
nula	$-\infty$	NaN	0
$+\infty$	nenulová kladná	$+\infty$	0
$-\infty$	nenulová kladná	$-\infty$	0
$+\infty$	nenulová záporná	$-\infty$	0
$-\infty$	nenulová záporná	$+\infty$	0
$+\infty$	nula	NaN	0
$-\infty$	nula	NaN	0
$+\infty$	$+\infty$	$+\infty$	0
$+\infty$	$-\infty$	$-\infty$	0
$-\infty$	$+\infty$	$-\infty$	0
$-\infty$	$-\infty$	$+\infty$	0
NaN	libovolná	NaN	0
libovolná	NaN	NaN	0
nenulová kladná	nenulová kladná	není určena	1
nenulová záporná	nenulová kladná	není určena	1
nenulová kladná	nenulová záporná	není určena	1
nenulová záporná	nenulová záporná	není určena	1

Tabulka 5.6: Závislost výstupů na vstupech v bloku n_case násobičky (podle[6])

Blok mantissaMul

Blok na vstupu přijme hodnoty mantis a typy obou čísel. Přidá k hodnotám mantis skryté bity a vynásobí je mezi sebou. Výstupem tohoto bloku, je hodnota vynásobených mantis, která má dvojnásobnou šířku než šířka mantisy plus skryté bity. Blok implementuje čtvrtý krok algoritmu 4.2.2.

Blok expAdder

Blok `expAdder` nejprve provede součet obou exponentů. Proběhne zvětšení exponentu z důvodu posuvu desetinné čárky při násobení mantis. Poté ověří, zda odečtením hodnoty bias nedojde k podtečení. Pokud ano, je nastaven příznak podtečení. Pokud k podtečení nedojde, je odečtena hodnota bias od součtu exponentů. Nyní je aktuální hodnota exponentu testována na přetečení a pokud nastalo, je příznak přetečení aktivován. Blok implementuje pátý krok algoritmu 4.2.2.

Blok block_norm

Blok má úkol převést výsledky z předchozích bloků do normalizovaného tvaru. Pokud to není možné, jsou čísla převedena do denormalizovaného tvaru. Blok ve výsledku z násobení mantis vyhledá první jedničku zleva. Je-li to možné provede odpovídající počet rotací a snížení exponentu tj. vznikne normalizované číslo. Pokud není nalezena jednička nebo pokud nelze provést rotaci nebo snížení exponentu vznikne denormalizované číslo. Poté ještě proběhne zaokrouhlení podle posledních čtyř bitů výstupu mantisy a odstranění skrytého bitu. Blok implementuje šestý krok algoritmu 4.2.2.

Blok block_result

V bloku dojde podle signálu `enable` k navrácení výsledku buď z bloku `n_case` nebo z bloku `block_norm`. Blok také detekuje příznaky přetečení a podtečení, ke kterým přiřazuje správné hodnoty mantisy a exponentu. Podle algoritmu 4.2.2 implementuje poslední osmý krok.

Propojení bloků

Jednotlivé bloky jsou propojeny podle obrázku 5.2. V obrázku jsou čárkovaně zakreslena místa, kde jsou vloženy pipeline registry. V obrázku není zakreslen signál `clk` a `rst`. Signál `clk` je zaveden do každého bloku a každého pipeline registru. Signál `rst` je zaveden do pipeline registrů pro reset odvodu.

5.2.2 Integrace s procesorem Plasma

Integrace k procesoru Plasma probíhá stejně jako u sčítačky mapováním do adresního prostoru procesoru viz tabulka 5.3. Registr pro násobičku je stejný jako pro sčítačku 5.5, ale místo bitu `add/sub` je prázdné místo. Je vytvořen blok viz obrázek 4.8, který spojuje rozhraní procesoru a násobičky. Tento blok je totožný jako blok sčítačky, obsahuje pouze místo komponenty sčítačky komponentu násobičky a má stejné rozhraní viz 5.4. I násobička pracuje ve dvou režimech stejně jako sčítačka. Jeden režim pracuje s příznaky `start` a `done`. Druhý režim pracuje bez příznaků tak, že neustále počítá výsledek násobení se čtyř hodinovou latencí proti vstupním hodnotám.

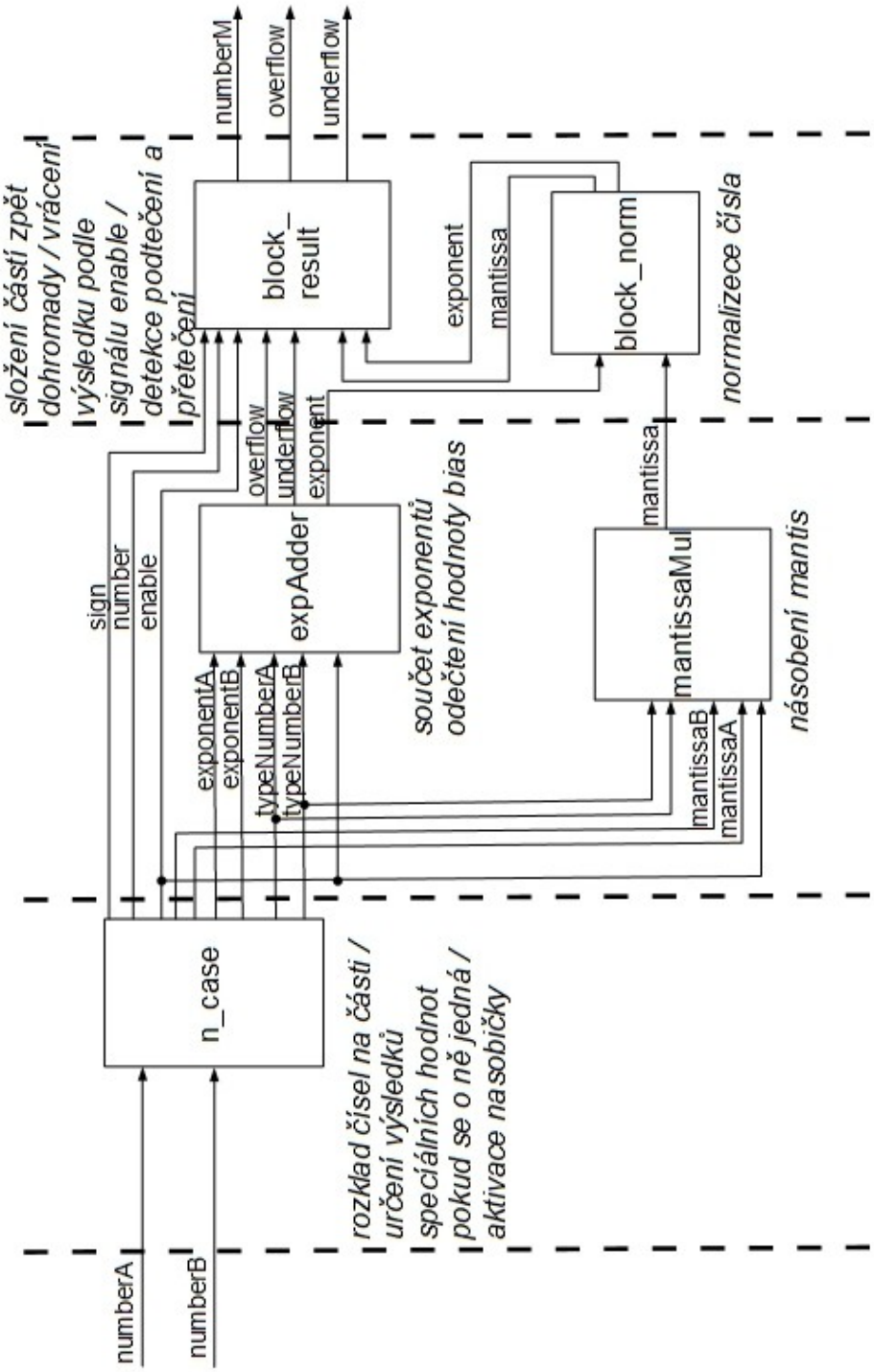
5.3 Dělička

5.3.1 Popis bloků

V RTL návrhu je obvod děličky rozdělen na šest bloků. Jedná se o bloky: `n_case`, `expSub`, `pre_mantissaDiv`, `mantissaDiv`, `block_norm` a `block_result`. Činnost jednotlivých bloků je řízena čítačem (stavovým automatem). Všechny bloky pracují s generickými konstantami, které umožňují, aby vstupní a výstupní signály měly různé šířky podle šířky vstupních hodnot.

Blok n_case

Blok `n_case` zpracuje vstupní hodnoty čísel a nalezne hodnoty znamének, exponentů a mantis. Z těchto hodnot určí o jaké typy čísel se jedná. Pokud detekuje speciální hodnoty, určí výsledek v tomto bloku a signál `enable` je nastaven v tomto případě do log. 0. Pokud se jedná o hodnoty, se kterými musí proběhnout dělení, je signál `enable` nastaven do log.



Obrázek 5.2: RTL schéma násobičky

1. Tabulka 5.7 ukazuje, jak reaguje obvod `n_case` na vstupní hodnoty. Blok implementuje první až třetí krok algoritmu 4.3.2.

Číslo A	**Číslo B**	**Znaménko výstupu**	**Číslo S (výstup)**	**Enable (výstup)**
Denormalizované číslo	Číslo B	Zn. A XOR zn. B	Nula	0
Číslo A	Denormalizované číslo	Zn. A XOR zn. B	Nula	0
Denormalizované číslo	Denormalizované číslo	Není určeno	Není určeno	1
Číslo A	∞	Zn. A XOR zn. B	Nula	0
∞	Číslo B	Zn. A XOR zn. B	∞	0
Číslo A	Nula	1	NaN	0
∞	∞	1	NaN	0
Číslo A	NaN	1	NaN	0
NaN	Číslo B	1	NaN	0
Číslo A	Číslo B	Není určeno	Není určeno	1

Tabulka 5.7: Závislost výstupů na vstupech v bloku n_case děličky

Blok pre_mantissaDiv

Blok podle typu čísel přidá k mantise skrytý bit a čtyři zaokrouhlovací bity. Takto upravená mantisa je převedena rotací vlevo do tvaru, že první bit vlevo je roven hodnotě jedna. Blok vrátí upravené mantisy a počet nutných rotací, aby se mantisy dostaly do tvaru s log. 1 na prvním bitu vlevo. Blok implementuje čtvrtý a část šestého kroku algoritmu 4.3.2.

Blok mantissaDiv

Tento blok provádí sériové dělení s návratem přes nulu. Blok je řízen signálem `we_div`, který určuje zda vstupní hodnota dělence je mantisa nebo předchozí výstup tohoto obvodu. Blok implementuje část šestého kroku algoritmu 4.3.2.

Blok expSub

Blok `expSub` provede odečtení hodnot exponentů a připočte hodnotu bias. Dále je přičten počet rotací mantisy dělence a odečten počet rotací dělitele. Tyto počty rotací jsou předány z bloku `pre_mantissaDiv`. Výstupem bloku je hodnota exponentu a příznaky přetečení a podtečení. Blok implementuje pátý krok algoritmu 4.3.2.

Blok block_norm

Popisovaný blok provádí normalizaci. Poté ještě proběhne zaokrouhlení k nejbližší hodnotě podle nejnižších čtyř bitů. Blok implementuje sedmý krok algoritmu 4.3.2.

Blok block_result

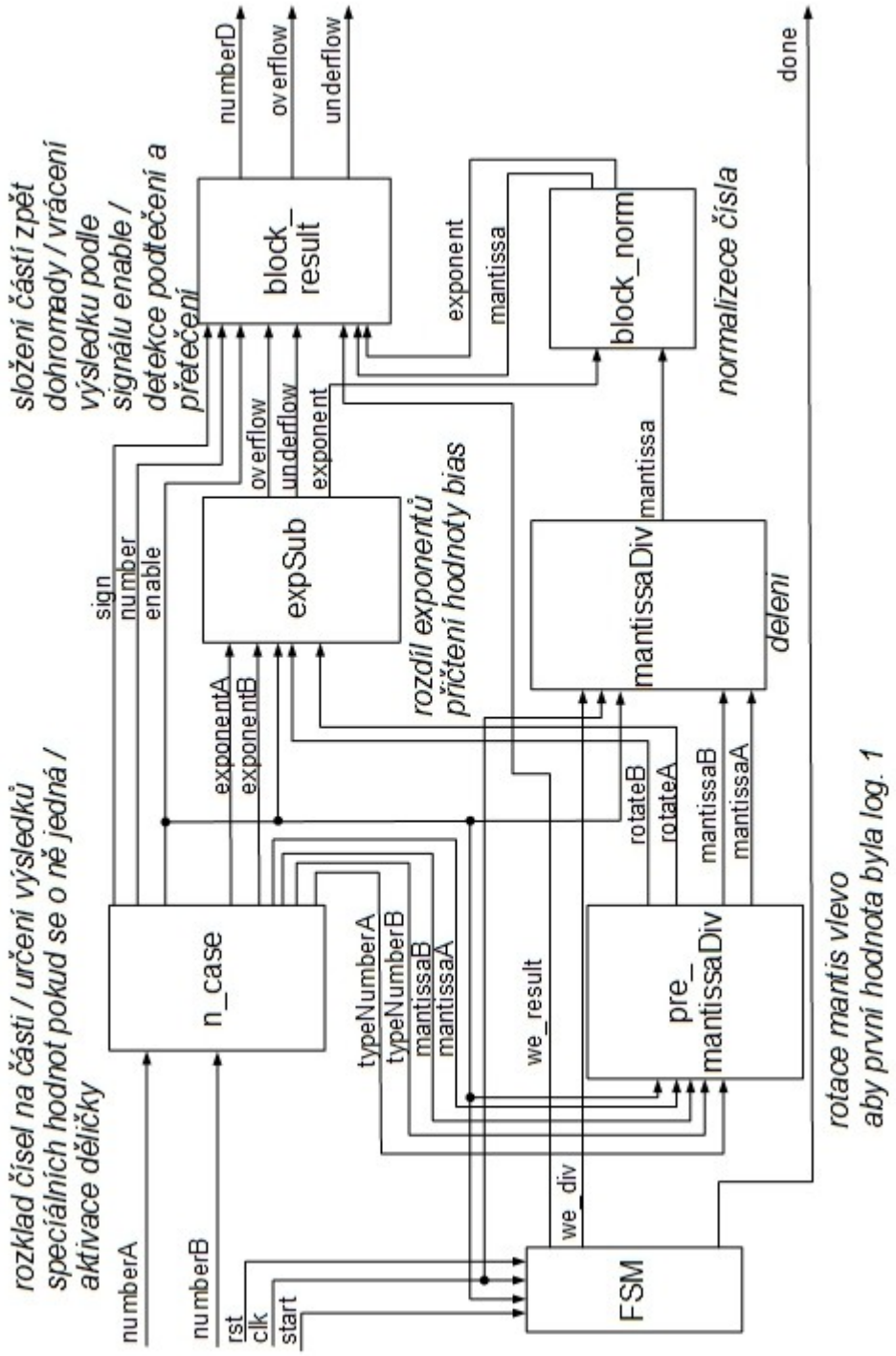
Blok podle vstupu `we_result` vrátí hodnotu z bloku `n_case` nebo z `block_norm`. Blok zanechává výstup nezměněn, dokud není vypočtena další hodnota. Tento blok implementuje osmý krok algoritmu 4.3.2.

Propojení bloků

Bloky jsou propojeny podle obrázku 5.3. Řízení obvodu zajišťuje stavový automat, který má pět stavů. Automat má výchozí stav `TWAIT` (stav po resetu). Po přijmutí signálu `start` blok `n_case` zpracuje hodnoty a automat se podle signálu `enable` přesune do dalšího stavu. Buď do stavu `TRESULTSPEC`, který vypíše výsledky vypočtené z bloku `n_case` nebo `TDIV0`, kterým započne výpočet dělení mantis (hodnoty mantis před dělením upraví blok `pre_mantissaDiv`). Následující stav `TDIV` provádí dělení předchozího výsledku z bloku `mantissaDiv`. Tento stav se opakuje, dokud podíl nemá stejný počet bitů jako je šířka mantisy. Pak nastane stav `TRESULT`, který dopočte zbývající hodnoty přes bloky `expSub` a `block_norm`. Správné výsledky zobrazí blok `block_result`. V tabulce 5.8 jsou zapsány stavy automatu a reakce řídících signálů na tyto stavy.

Stav	**Příští stav**	**we_div**	**we_result**	**Poznámka**
TWAIT	start='1' & enable='1' -> TDIV0	0	00	Počáteční stav. Automat z tohoto stavu jde po nastavení signálu `start` a podle výsledků z bloku `n_case` pokračuje do dalšího stavu.
	start='1' & enable='0' -> TRESULT-SPEC			
TDIV0	TDIV	0	00	Blok `mantissaDiv` provede první dělení na vstupních mantisách.
TDIV	cnt < widthMantissa-1+5 -> TDIV	1	00	Blok `mantissaDiv` provádí další dělení už s předchozími výsledky z tohoto bloku. Dělení končí po zaplnění všech bitů výstupní mantisy - o to se stará čítač `cnt`.
	cnt >= widthMantissa-1+5 -> TRE-SULT			
TRESULT	TWAIT	1	10	Vypíše na výstup hodnoty z bloku `n_case` a nastaví log. 1 signálu `done`.
TRESULTSPEC	TWAIT	0	01	Vypíše na výstup hodnoty z bloku `block_norm` a nastaví log. 1 signálu `done`.

Tabulka 5.8: Řídící signály a reakce čítače děličky



Obrázek 5.3: RTL schéma děličky

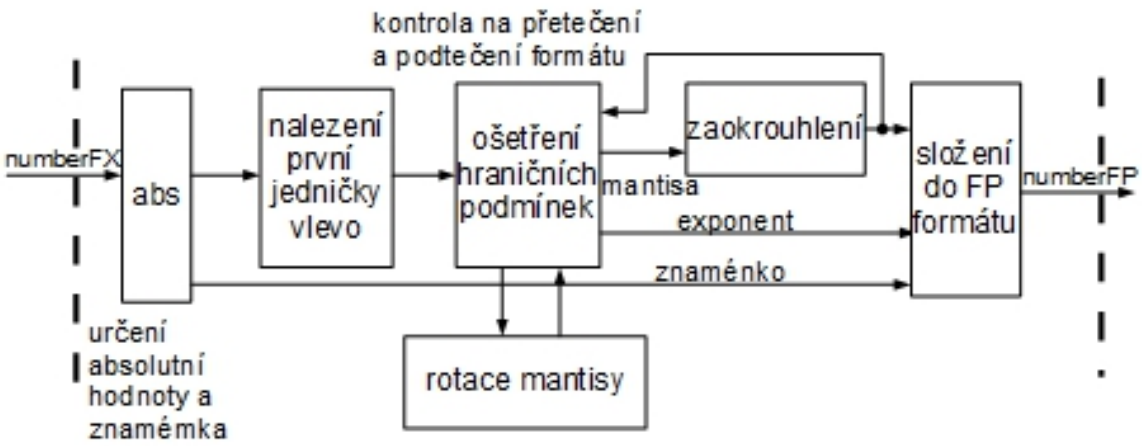
5.3.2 Integrace s procesorem Plasma

Integrace k procesoru Plasma probíhá stejně jako u sčítačky mapováním do adresního prostoru procesoru viz tabulka 5.3. Registr pro děličku je stejný jako pro sčítačku 5.5, ale místo bitu `add/sub` je prázdné místo. Je vytvořen blok viz obrázek 4.8, který spojuje rozhraní procesoru a děličky. Tento blok je totožný jako blok sčítačky obsahuje, pouze místo komponenty sčítačky komponentu děličky a má stejné rozhraní viz 5.4. Dělička pracuje pouze v jednom režimu. Tento režim pracuje s příznaky `start` a `done`. Bit `start` označuje žádost o započetí výpočtu. Bit `done` označuje, že výpočet je dokončen.

5.4 Konverze z fixed point čísla

5.4.1 Popis bloku

Blok pracuje s generickými konstantami, které umožňují, aby vstupní a výstupní signály měly různé šířky. Na obrázku 5.4 je blokové schéma obvodu, které vychází z popsaného algoritmu 4.4.1. Na obrázku je vidět umístění pipeline registrů.



Obrázek 5.4: Blokové schéma konverze z fixed point čísla

5.4.2 Integrace s procesorem Plasma

Stejně jako předchozí obvody je tento obvod připojen k procesoru Plasma namapováním do adresního prostoru viz tabulka 5.3. Pro propojení je vytvořen blok viz obrázek 4.8, který spojuje rozhraní procesoru a převodníků formátů (tento blok je pro oba typy převodníky). Tento blok má totožné rozhraní jako blok pro sčítačku viz 5.4, ale místo sčítačky jsou připojeny obvody pro konverzi.

Registr pro konverzi z FX obsahuje dva bity a to `start` a `done` viz tabulka 5.9. Po zadání bitu `start` obvod po dokončení výpočtu přepne do log. 1 bit `done`. Obvod může pracovat ve dvou režimech. Jeden režim využívá bity `start` a `done` v registru pro řízení výpočtu. Druhý režim pracuje bez registru a uživatel musí po vložení vstupní hodnoty očekávat výstup s latencí dvou cyklů.

bit 1	bit 0
done	**start**
R	R/W

Tabulka 5.9: Registr pro řízení konverze z FX čísla s procesorem Plasma

5.5 Konverze z floating point čísla

5.5.1 Popis bloku

Blok pracuje s generickými konstantami, které umožňují, aby vstupní a výstupní signály měly různé šířky. Na obrázku 5.5 je blokové schéma obvodu, které vychází z popsaného algoritmu 4.5.1. Na obrázku je vidět umístění pipeline registrů.

Obrázek 5.5: Blokové schéma konverze z floating point čísla

5.5.2 Integrace s procesorem Plasma

Blok je integrován stejným způsobem jako blok konverze z FX čísla a přes stejný blok je připojen k procesoru viz 5.4.2. Adresní prostor obvodu je zapsán v tabulce 5.3.

Kapitola 6

Verifikace

Kapitola popisuje verifikační plán pro jednotlivé obvody. V další části jsou popsány výsledky verifikací.

Verifikace probíhá pro aritmetické operace na číslech ve formátu FP single (šířka 32 bitů). Pro obvod sčítačky je provedena simulace na RTL úrovni, která ověřuje funkčnost i pro jiné šířky čísel. Pro ostatní aritmetické operace tyto simulace z časových důvodů nebyly provedeny.

Verifikace pro konverzi z a na FX číslo probíhá ze znaménkového formátu Q31.0 do formátu FP single a nazpět. Pro tyto obvody jsou provedeny simulace na RTL úrovni, které pracují s velmi širokými formáty FX čísel tak, aby nastaly situace, kdy je číslo převedeno na hodnotu nekonečna (kladného nebo záporného) nebo na denormalizované číslo.

Simulace probíhají v programu ModelSim za využití knihovny `ieee` a v programu ISim za použití knihovny `ieee_proposed`.

6.1 Verifikační plán

Simulace pro všechny navrhované obvody probíhá na RTL úrovni a poté na hradlové úrovni. Verifikace probíhá na jednotlivých blocích (pouze na RTL úrovni) a pak na celém systému (jak na RTL úrovni tak i na GATE LEVEL).

Ověření funkčnosti aritmetických bloků probíhá za pomoci kombinací vstupních čísel. Ověřuje kombinace všech speciálních hodnot (nekonečno, nula a NaN) a dále náhodná čísla z řady normalizovaných čísel, náhodná z denormalizovaných a náhodná kombinace normalizovaného a denormalizovaného čísla.

6.1.1 Verifikační plán pro sčítačku

Po vytvoření bloku na RTL úrovni je vytvořena simulace, která kontroluje správnost výstupů vůči očekávaným výstupům. Tato kontrola je manuální a musí se zadávat vhodné příklady a ručně (očima) ověřovat, zda výsledky odpovídají vstupům. Blok `n_case` dostává na své vstupy hodnoty z tabulky 5.1 a očekává výstupy, které jsou napsány v této tabulce. U bloku `pre_adder` jsou na vstup dodány hodnoty všech možných kombinací typů čísel (normalizované/denormalizované) a je očekáván výsledek jednoho exponentu vyššího čísla a podle toho

upravené mantisy. Blok `block_adder` dostává na vstup všechny možné kombinace vstupních znamének a kombinace mantis pro ověření, zda blok provede správnou operaci a zda vrátí odpovídající znaménko viz tabulka 5.2. U bloku normalizace jsou vstupní hodnoty různé mantisy a ve výsledku se kontroluje, zda mantisa byla správně pootočená a exponent zvýšen. Dále se musí kontrolovat, zda proběhlo zaokrouhlení podle očekávání. Na posledním bloku se ověřuje, zda při aktivní hodnotě `enable` je vráceno dobře složené číslo ze sčítačky a pokud je `enable` neaktivní, zda je výsledkem hodnota z bloku `n_case`.

Pro verifikaci celého obvodu sčítačky je na obrázku 6.1 nakresleno blokové schéma testbenche. Na vstup testbenche budou přivedena dvě reálná čísla z generátoru stimulů, která budou náhodně generována. Na počátku simulace budou generovány speciální hodnoty a poté se zpracovávají všechny možné kombinace čísel (záporná a kladná čísla, normalizované a denormalizované hodnoty). Obě čísla budou převedena na FP formát a předána sčítačce, aby provedla součet. Výsledek ze sčítačky bude porovnán s výsledkem součtu v reálných hodnotách, který pro porovnání bude převeden do FP formátu. Výsledkem bude jasné rozhodnutí zda se součet/rozdíl shoduje. Dále proběhne další porovnání a to tak, že vstupní čísla v reálném formátu jsou převedena do FP formátu, sečtena/odečtena a porovnána s výsledkem z implementované sčítačky. Simulace provádí verifikaci správné funkce pipeline, poněvadž v celém obvodu se zpracovávají čtyři vstupní hodnoty naráz. Tímto schématem testbenche proběhne verifikace i na hradlové úrovni. Verifikace je úspěšná, pokud se výsledky shodují s výpočtem přes FP formát a rozdílné hodnoty se liší maximálně ve čtyřech nejnižších bitech.

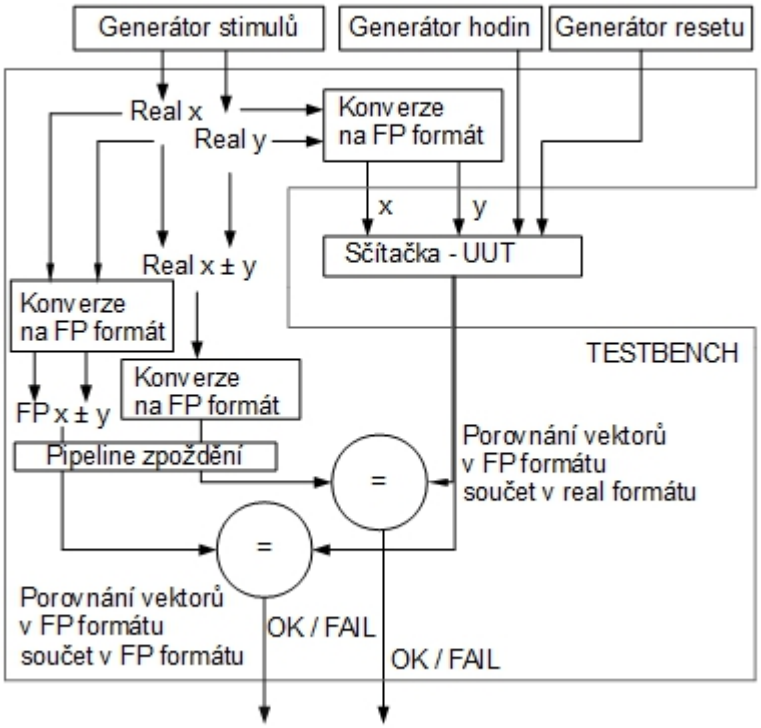
6.1.2 Verifikační plán pro násobičku

Tento verifikační plán je totožný jako verifikační plán pro sčítačku. Na RTL úrovni jsou simulovány všechny bloky. Výsledky jednotlivých simulací obvodů jsou kontrolovány ručně (očima). Výsledný blok je verifikován podle totožného schématu 6.1 jako sčítačka, ale operace součtu/rozdílu jsou nahrazeny násobením. Výsledný blok je verifikován jak na RTL úrovni tak na hradlové úrovni. Verifikace je úspěšná, pokud se výsledky shodují s výpočtem přes FP formát a rozdílné hodnoty se liší maximálně ve čtyřech nejnižších bitech.

6.1.3 Verifikační plán pro děličku

Po vytvoření bloku na RTL úrovni je vytvořena simulace, která kontroluje správnost výstupů vůči očekávaným výstupům. Tato kontrola je manuální a musí se zadávat vhodné příklady a ručně (očima) ověřovat, zda výsledky odpovídají vstupům.

Pro verifikaci celého obvodu děličky je na obrázku 6.2 nakresleno blokové schéma testbenche. Na vstup testbenche budou přivedena dvě reálná čísla z generátoru stimulů, která budou náhodně generována. Na počátku simulace budou generovány speciální hodnoty a poté se zpracovávají všechny možné kombinace čísel (záporná a kladná čísla, normalizované a denormalizované hodnoty). Generátor stimulů reaguje na stav `done` a započíná výpočet příznakem `start`. Obě čísla budou převedena na FP formát a předána děličce, aby provedla dělení. Výsledek z dělení bude porovnán s výsledkem dělení v reálných hodnotách, který pro porovnání bude převeden do FP formátu. Výsledkem bude jasné rozhodnutí, zda se dělení shoduje. Dále proběhne další porovnání a to tak, že vstupní čísla v reálném formátu

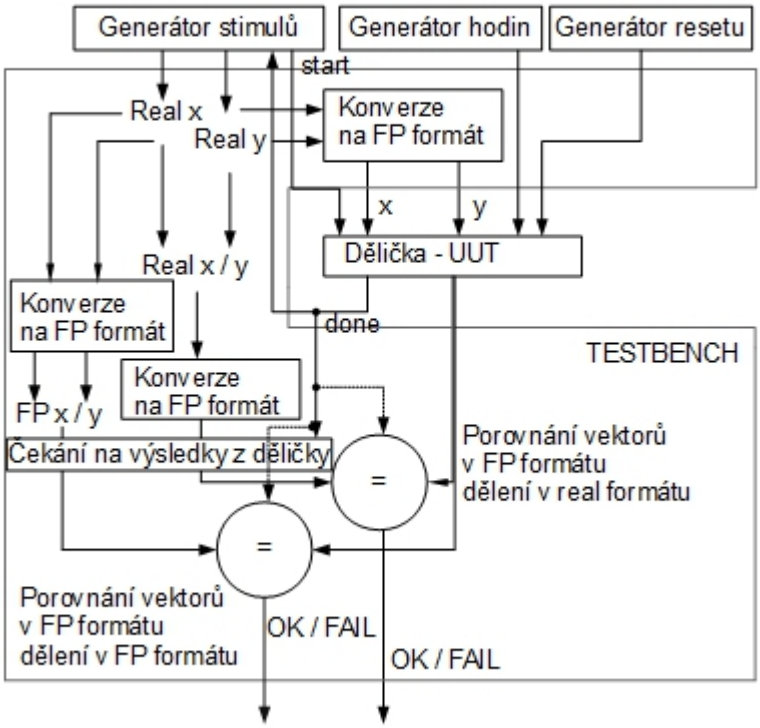


Obrázek 6.1: Postup verifikace pro sčítačku. Srovnání modelem [11]

jsou převedena do FP formátu, vydělena a porovnána s výsledkem z implementované děličky. Simulace provádí verifikaci správné funkce řízení pomocí signálu `start` a `done`. Tímto schématem testbenche proběhne verifikace i na hradlové úrovni. Verifikace je úspěšná, pokud se výsledky shodují s výpočtem přes FP formát a rozdílné hodnoty se liší maximálně ve čtyřech nejnižších bitech.

6.1.4 Verifikační plán pro konverzi z FX čísla

Pro verifikaci obvodu pro konverzi z FX čísla je na obrázku 6.3 nakresleno blokové schéma testbenche. Na vstupu testbenche je generátor stimulů, který generuje náhodná reálná čísla. Tyto čísla jsou převedena do FX formátu a předána testovanému bloku, který vypočte odpovídající hodnotu FP čísla. Tato hodnota je porovnána s hodnotou vypočtenou pomocí balíčků. Výsledkem testbenche je, zda se čísla shodují. Simulace provádí verifikaci správné funkce pipeline, poněvadž zpracovává dvě hodnoty naráz. Verifikace je úspěšná, pokud se výsledky shodují s výpočtem přes FP formát a rozdílné hodnoty se liší maximálně ve čtyřech nejnižších bitech.



Obrázek 6.2: Postup verifikace pro děličku. Srovnání modelem [11]

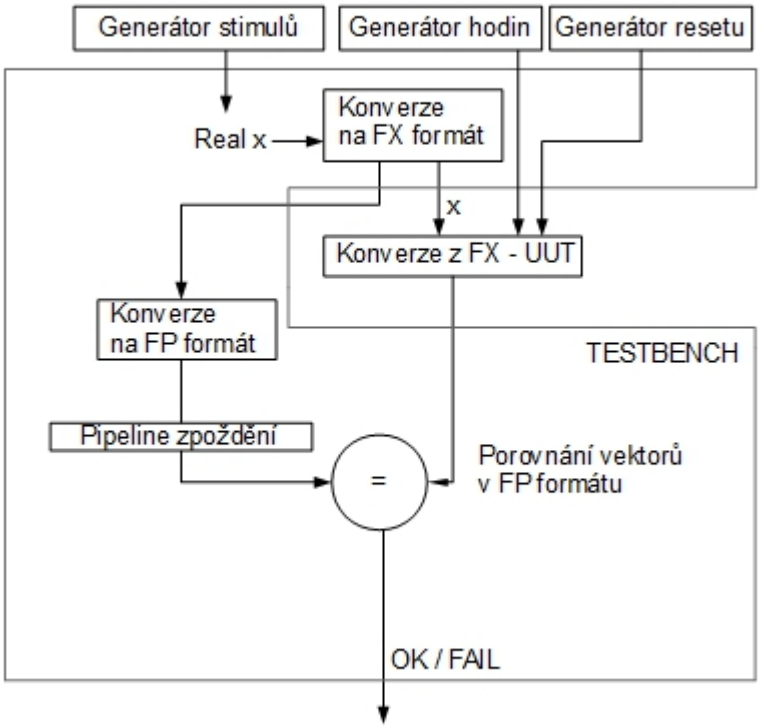
6.1.5 Verifikační plán pro konverzi z FP čísla

Pro verifikaci obvodu je na obrázku 6.4 nakresleno blokové schéma testbenche. Popis schématu je totožný s popisem schématu pro testbench pro konverzi z FX čísla viz 6.1.4. Verifikace je úspěšná, pokud se výsledky shodují s výpočtem přes FX formát a rozdílné hodnoty se liší maximálně ve čtyřech nejnižších bitech.

6.2 Verifikace na RTL úrovni

6.2.1 Verifikace sčítačky na RTL úrovni

Na obrázku 6.5 je vidět část simulace sčítačky. Prvních osm signálů jsou data ze vstupů a výstupů sčítačky a zbytek signálů je pro možnosti porovnávání. Z obrázku je vidět práce pipeline, které zpožďují výsledek o čtyři hodinové cykly oproti zapsaným vstupům. Pro názornost popisuji jednu operaci sčítačky zobrazenou v obrázku. Prvním vstupem na obrázku jsou hodnoty 0x8001167B a 0x00006F13 a prováděnou operací je odčítání. Po čtyřech hodinových cyklech na `numberS` je výsledek 0x8001858E (-9.99E-41 - 3.9846E-41 = -1.39746E-40). Hodnota převáděná z typu float se shoduje, ale hodnota převedená z typu real je rozdílná

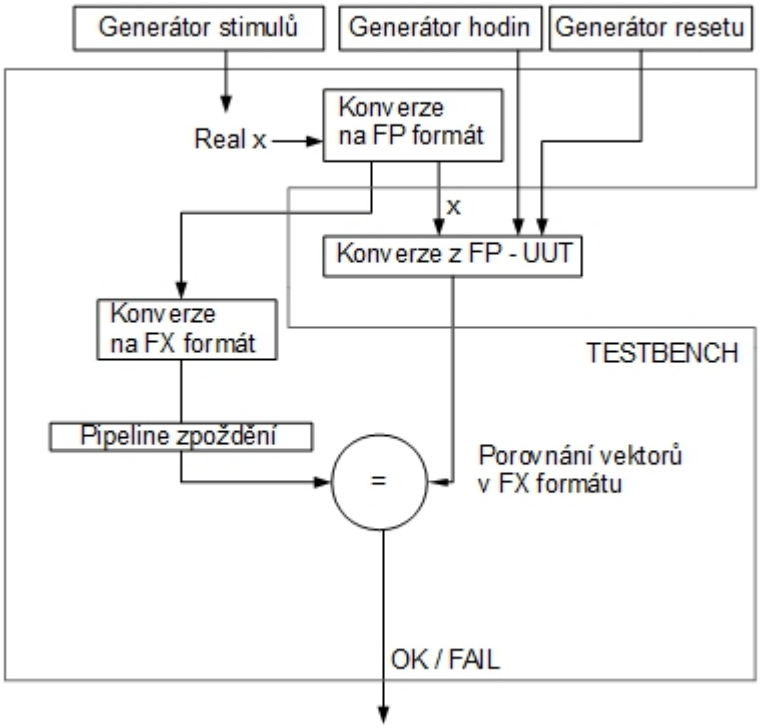


Obrázek 6.3: Postup verifikace pro konverzi z FX čísla. Srovnání modelem [11]

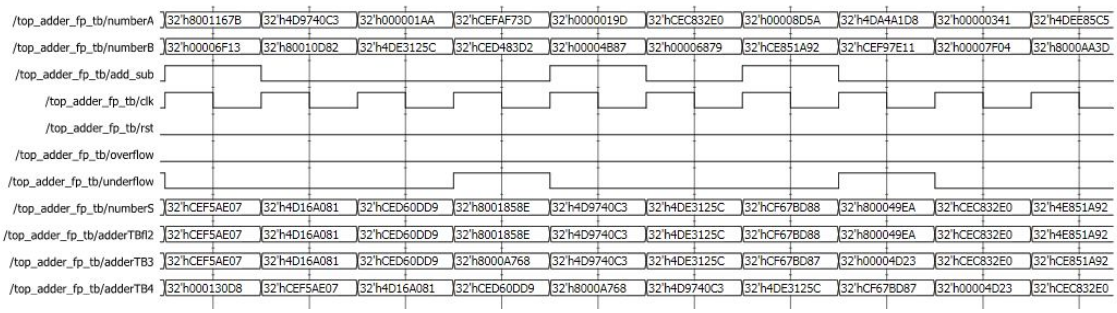
(0x8000A768). Tento velký rozdíl je dán přesností typu real, který neumožňuje přesněji pracovat s takto malými čísly. Takto velký rozdíl je výjimkou pro hodně malá a hodně velká čísla. Jinak se rozdíl v hodnotách projevuje na posledních čtyřech bitech a toto je dáno zaokrouhlováním. Výstupem simulace po 31832 kombinacích náhodných čísel a testování speciálních hodnot je následující záznam v logu simulátoru.

```
Pocet prubehu testovani nahodnych cisel 31832 a pocet shodujicich se vysledku
s realnym scitanim/odcitanim je 17295 a pocet shodujidich se vysledku s float
scitanim/odcitanim 31827 (probehla kontrola specialnich cisel - vysledkem je,
ze na 22 vystupech je shoda v 22 hodnotach)
```

Z výsledků simulace na RTL úrovni je viditelné, že při porovnání s výpočtem součtu/rozdílu ve float formátu dosahuje implementace sčítačky shodu v 99,9 % testovaných hodnot. Při porovnání s výpočtem součtu/rozdílu ve formátu real je shoda pouze 54,3 %. Jak už bylo v předchozí části popsáno, je rozdíl dán u obou typů zaokrouhlováním a u porovnání s hodnotami získanými po součtu typem real, je dán nedostatečnou přesností tohoto typu reprezentovat takto malé a velké hodnoty. Verifikaci proběhla správně, sčítačka pracuje korektně jak s číselnými, tak i se speciálními hodnotami.



Obrázek 6.4: Postup verifikace pro konverzi z FP čísla. Srovnání modelem [11]



Obrázek 6.5: Ukázka simulace sčítačky na RTL úrovni

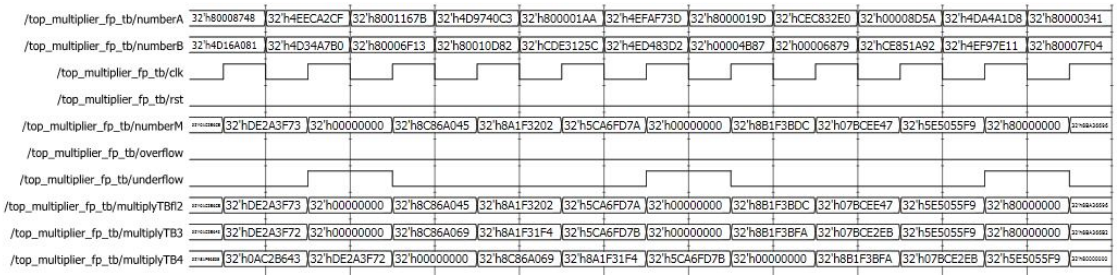
6.2.2 Verifikace násobičky na RTL úrovni

Na následujícím obrázku 6.6 je výřez ze simulace pro násobičku. Prvních sedm signálů jsou vstupy a výstupy násobičky, následující signály jsou pro porovnávání s výstupní hodnotou násobičky. Pro názornost funkce pipeline popíšu hodnoty, které jsou na obrázku jako první zpracované. Jedná se o hodnotu 0x80008748 a 0x4d16a081. Výsledkem za čtyři hodinové

cykly je hodnota 0x4d16a081 (-4.853E-41 × 1.57943824E8 = -7.664977E-33). Při porovnání s hodnotou vypočtenou přes formát float je shoda, přes formát real je neshoda. Výsledkem RTL simulace je následující textový výpis z logu simulátoru.

```
Pocet prubehu testovani nahodnych cisel 79667 a pocet shodujicich se vysledku
s realnym nasobenim je 33316 a pocet shodujidich se vysledku s float
nasobenim 79655 (probehla kontrola specialnich cisel - vysledkem je,
ze na 20 vystupech je shoda v 20 hodnotach)
```

Z výsledků simulace je viditelné, že výstupy násobičky se shodují z 99,9 % s hodnotami násobení ve formátu float. S hodnotami násobení ve formátu real se výsledky násobičky shodují z 41,8 %. Důvody rozdílných hodnot jsou totožné jako u sčítačky viz 6.2.1. Verifikace proběhla úspěšně.



Obrázek 6.6: Ukázka simulace násobičky na RTL úrovni

6.2.3 Verifikace děličky na RTL úrovni

Na obrázku 6.7 je vidět část simulace děličky. Prvních devět signálů jsou vstupy a výstupy děličky. Ostatní signály jsou určeny k porovnávání výstupů. Popis první vstupní simulace obrázku: Hodnota 0xCEA27466 je dělena 0xCEF66560. Po zapsání log. 1 do signálu `start` je proveden výpočet. Dokončení je signalizováno signálem `done`. Výsledkem je 0x3F28C96E (-1.36276864E9 / -2.06691942E9 = 0.6593236). Výsledkem RTL simulace je následující textový výpis z logu simulátoru.

```
Pocet prubehu testovani nahodnych cisel 6059 a pocet shodujicich se vysledku
s realnym delenim je 4024 a pocet shodujidich se vysledku s float delenim
5914 (probehla kontrola specialnich cisel - vysledkem je, ze na 20 vystupech
je shoda v 20 hodnotach)
```

Z výsledků simulace je viditelné, že výstupy děličky se shodují z 97,6 % s hodnotami dělení ve formátu float. S hodnotami dělení ve formátu real se výsledky děličky shodují z 66,4 %. Důvody rozdílných hodnot jsou totožné jako u sčítačky viz 6.2.1. Verifikace proběhla úspěšně.

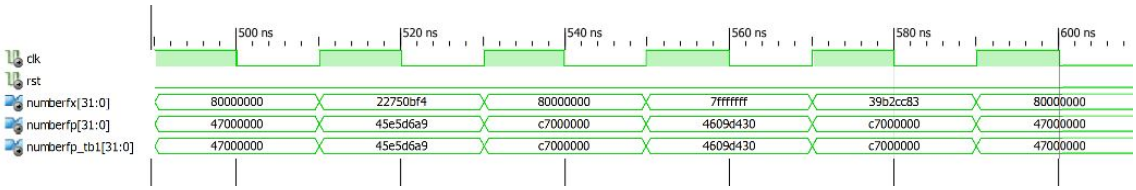
Obrázek 6.7: Ukázka simulace děličky na RTL úrovni

6.2.4 Verifikace konverze z FX čísla na RTL úrovni

Na obrázku 6.8 je ukázka části simulace konverze z FX čísla. První příklad ze simulace je tento: hodnota čísla v FX formátu 0x80000000 je převedena za dva hodinové cykly na číslo v FP formátu 0xC7000000 (-32768). Výsledkem RTL simulace je následující textový výpis z logu simulátoru.

```
Pocet prubehu testovani nahodnych cisel 22239 a pocet shodujicich se vysledku
je 22086
```

Z výpisu logu je vidět, že shoda porovnávaných hodnot je v 99,3 %. Rozdílné hodnoty jsou způsobeny zaokrouhlováním a hodnoty se liší pouze v nejnižších čtyřech bitech. Verifikace proběhla úspěšně.



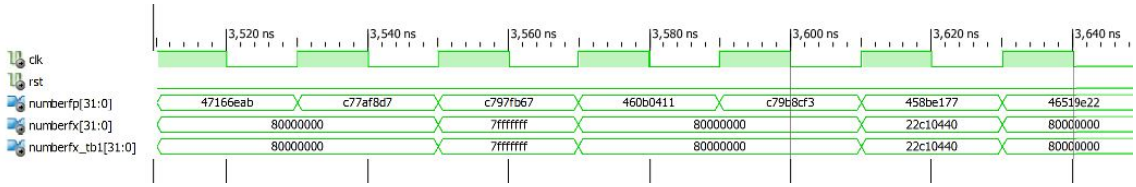
Obrázek 6.8: Ukázka simulace konverze z FX čísla na RTL úrovni

6.2.5 Verifikace konverze z FP čísla na RTL úrovni

Na obrázku 6.9 je ukázka části simulace konverze z FP čísla. První příklad ze simulace je tento: hodnota čísla v FP formátu 0x47166eab (38510.668) je převedena za dva hodinové cykly na číslo v FX formátu 0x7fffffff (32767.9999). V tomto případě došlo k přetečení a hodnota FX čísla je maximální kladná co v tomto formátu může být. Výsledkem RTL simulace je následující textový výpis z logu simulátoru.

```
Pocet prubehu testovani nahodnych cisel 12236 a pocet shodujicich se vysledku
je 12231
```

Z výpisu logu je vidět, že shoda porovnávaných hodnot je v 99,9 %. Rozdílné hodnoty jsou způsobeny tím, že navrhovaný převodník neprovádí zaokrouhlování. Rozdíly jsou v nejnižších čtyřech bitech. Verifikace proběhla úspěšně.



Obrázek 6.9: Ukázka simulace konverze z FP čísla na RTL úrovni

6.3 Verifikace na GATE LEVEL úrovni

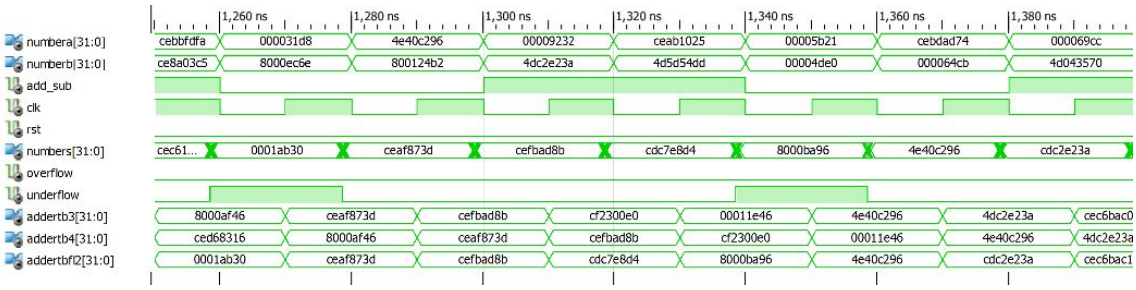
Pro verifikaci na GATE LEVEL úrovni je použita 50 MHz pracovní frekvence. Pro všechny rozdílné hodnoty výsledků simulace platí stejné odůvodnění jako v RTL úrovni verifikace.

6.3.1 Verifikace sčítačky na GATE LEVEL úrovni

Obrázek 6.10 je krátký výsek ze simulace pro sčítačku. Následující výpis je z logu simulátoru po simulaci na GATE LEVEL úrovni.

```
Pocet prubehu testovani nahodnych cisel 2824 a pocet shodujicich se vysledku
s realnym scitanim/odcitanim je 1510 a pocet shodujidich se vysledku s float
scitanim/odcitanim 2824 (probehla kontrola specialnich cisel - vysledkem je,
ze na 22 vystupech je shoda v 22 hodnotach)
```

Verifikace proběhla úspěšně.



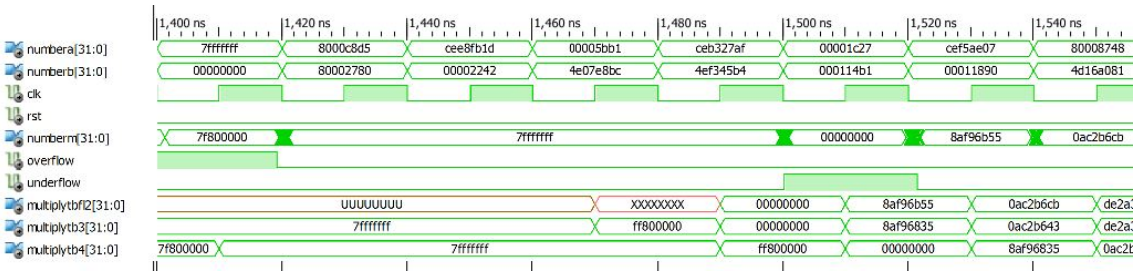
Obrázek 6.10: Ukázka simulace sčítačky na GATE LEVEL úrovni

6.3.2 Verifikace násobičky na GATE LEVEL úrovni

Obrázek 6.11 je krátký výsek ze simulace pro násobičku. Následující výpis je z logu simulátoru po simulaci na GATE LEVEL úrovni.

```
Pocet prubehu testovani nahodnych cisel 457 a pocet shodujicich se vysledku
s realnym nasobenim je 192 a pocet shodujidich se vysledku s float nasobenim
457 (probehla kontrola specialnich cisel - vysledkem je, ze na 20 vystupech
je shoda v 20 hodnotach)
```

Verifikace proběhla úspěšně.



Obrázek 6.11: Ukázka simulace násobičky na GATE LEVEL úrovni

6.3.3 Verifikace děličky na GATE LEVEL úrovni

Obrázek 6.12 je krátký výsek ze simulace pro děličku. Následující výpis je z logu simulátoru po simulaci na GATE LEVEL úrovni.

```
Pocet prubehu testovani nahodnych cisel 900 a pocet shodujicich se vysledku
s realnym delenim je 609 a pocet shodujidich se vysledku s float delenim 885
(probehla kontrola specialnich cisel - vysledkem je, ze na 20 vystupech je
shoda v 20 hodnotach)
```

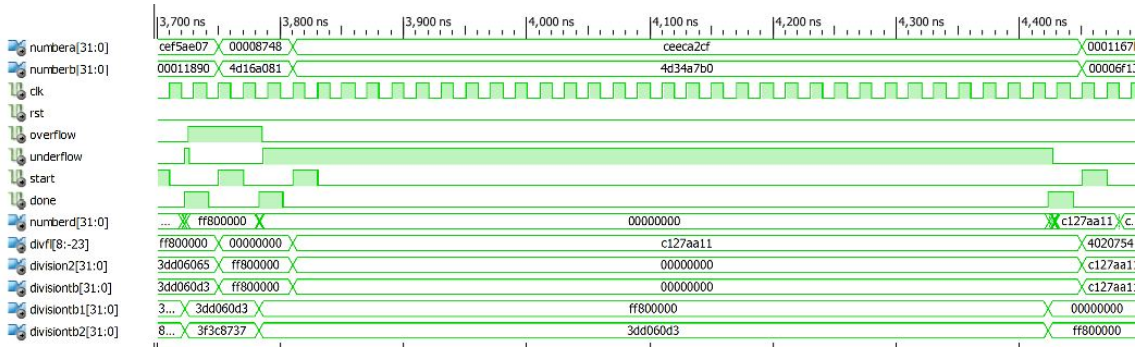
Verifikace proběhla úspěšně.

6.3.4 Verifikace konverze z FX čísla na GATE LEVEL úrovni

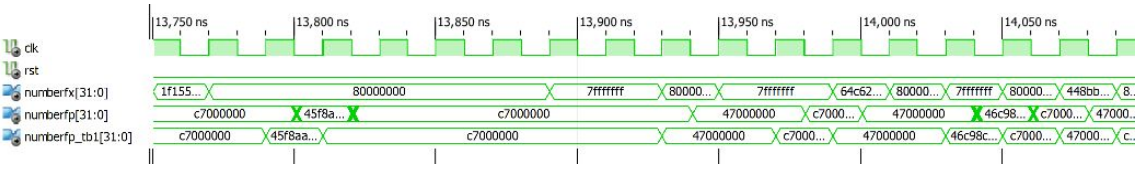
Obrázek 6.13 je krátký výsek ze simulace pro konverzi z FX čísla. Následující výpis je z logu simulátoru po simulaci na GATE LEVEL úrovni.

```
Pocet prubehu testovani nahodnych cisel 2388 a pocet shodujicich se vysledku
je 2370
```

Verifikace proběhla úspěšně.



Obrázek 6.12: Ukázka simulace děličky na GATE LEVEL úrovni



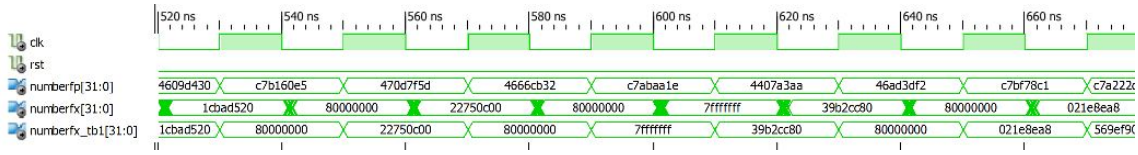
Obrázek 6.13: Ukázka simulace konverze z FX čísla na GATE LEVEL úrovni

6.3.5 Verifikace konverze z FP čísla na GATE LEVEL úrovni

Obrázek 6.14 je krátký výsek ze simulace pro konverzi z FP čísla. Následující výpis je z logu simulátoru po simulaci na GATE LEVEL úrovni.

```
Pocet prubehu testovani nahodnych cisel 7507 a pocet shodujicich se vysledku
je 7500
```

Verifikace proběhla úspěšně.



Obrázek 6.14: Ukázka simulace konverze z FP čísla na GATE LEVEL úrovni

Kapitola 7

Implementace

V této kapitole jsou popsány všechny implementace navrhovaných obvodů. Každý obvod je implementován do FPGA řady Spartan6 dle tabulky 7.1.

Family	Spartan6
Device	XC6SLX16
Package	CSG324
Speed	-2

Tabulka 7.1: Zvolené FPGA pro imlementaci obvodů

7.1 Popis implementace sčítačky

Syntéza obvodu

Při syntéze obvodu jsou generována následující varování. Varování upozorňuje na nepoužití vstupního signálu `clk` v jednotlivých blocích. Signál `clk` je v obvodu využíván pouze v registrech určených pro pipelining. Do ostatních bloků je zaveden pro případ budoucí potřeby.

```
WARNING:Xst:647 - Input <clk> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a
sub-block and the hierarchy of this sub-block is preserved.
```

Syntéza obvodu pro procesor Plasma

Pro připojení obvodu k procesoru Plasma je vytvořen připojovací blok, jehož výstup ze syntézy je níže pod textem. Několikrát se zde objevuje `Range is empty (null range)`, které vyvolává použití balíčků `fixed_pkg` a `float_pkg`. Varování `Choice with meta-value 'U' is ignored for synthesis` upozorňuje, že defaultní hodnoty při syntéze budou ignorovány. Dalším varování `Input <address<27:8>> is never used` upozorňuje, že nejsou využívány všechny signály sběrnice `address`. Posledním varováním je už známé nepoužití signálu `clk`.

```
WARNING:HDLCompiler:746 - "N:/P.20131013/rtf/vhdl/src/ieee_proposed/
fixed_pkg_c.vhd" Line 1472: Range is empty (null range)
WARNING:HDLCompiler:746 - "N:/P.20131013/rtf/vhdl/src/ieee_proposed/
float_pkg_c.vhd" Line 1022: Range is empty (null range)
WARNING:HDLCompiler:314 - "N:/P.20131013/rtf/vhdl/xst/src/std_1164.vhd" Line
1037: Choice with meta-value 'U' is ignored for synthesis
WARNING:Xst:647 - Input <address<3:2>> is never used. This port will be
preserved and left unconnected if it belongs to a top-level block or it belongs
to a sub-block and the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <address<27:8>> is never used. This port will be preserved
and left unconnected if it belongs to a top-level block or it belongs to a
sub-block and the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <address<31:31>> is never used. This port will be preserved
and left unconnected if it belongs to a top-level block or it belongs to a
sub-block and the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <clk> is never used. This port will be preserved and left
unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
```

Časová analýza

Níže je zpráva vytvořená po statické časové analýze. Z této zprávy plyne, že obvod by měl správně pracovat ještě za minimální hodinové periody 17,2 ns, tedy cca 58,2 MHz.

```
Timing constraint: Default period analysis for net "clk_BUFGP"
  1028763324 paths analyzed, 937 endpoints analyzed, 0 failing endpoints
  0 timing errors detected. (0 setup errors, 0 hold errors)
  Minimum period is  17.166ns.
 ---------------------------------------------------------------------------------
 Delay (setup path):     17.166ns (data path - clock path skew + uncertainty)
   Source:               comp_i1_1 (FF)
   Destination:          mantissaOut_i2_11 (FF)
   Data Path Delay:      17.148ns (Levels of Logic = 14)
   Clock Path Skew:      0.017ns (0.454 - 0.437)
   Source Clock:         clk_BUFGP rising
   Destination Clock:    clk_BUFGP rising
   Clock Uncertainty:    0.035ns
```

Časová analýza obvodu pro procesor Plasma

Časová analýza je prováděna pouze na bloku propojující procesor Plasma a obvodu sčítačky. Výsledkem je následující zpráva. Minimální hodinová perioda, podle této zprávy je 16,4 ns, tedy cca 60,9 MHz. Tato frekvence je postačující, aby obvod sčítačky mohl pracovat s procesorem Plasma, který pracuje na frekvenci 50 MHz.

```
 Timing constraint: Default period analysis for net "clk_BUFGP"
  1139901980 paths analyzed, 949 endpoints analyzed, 0 failing endpoints
  0 timing errors detected. (0 setup errors, 0 hold errors)
```

```
 Minimum period is  16.340ns.
-------------------------------------------------------------------------------
Delay (setup path):     16.340ns (data path - clock path skew + uncertainty)
  Source:               adder/numberB_i0_18 (FF)
  Destination:          adder/mantissaAout_i1_17 (FF)
  Data Path Delay:      16.299ns (Levels of Logic = 11)
  Clock Path Skew:      -0.006ns (0.518 - 0.524)
  Source Clock:         clk_BUFGP rising
  Destination Clock:    clk_BUFGP rising
  Clock Uncertainty:    0.035ns
```

Implementační parametry

V tabulce 7.2 jsou zapsány základní implementační parametry obvodu sčítačky. V tabulce jsou pro porovnání zapsány i hodnoty pro sčítačku, která nevyužívá pipeline.

	Bez pipeline	**4 st. pipeline**
Počet použitých registrů	0	276
Počet použitých LUT (tabulek pro kombinační funkce)	1051	1179
Celkem obsazených řezů (slices)	376	418
Nejdelší kombinační cesta (ns)	55,73	17,2
Maximální frekvence (MHz)	17,9	58,2

Tabulka 7.2: Implementační parametry obvodu sčítačky

7.1.1 Příklad softwaru pro výpočet na procesoru Plasma

V příloze D.1 je příklad programu v jazyce C pro výpočet součtu/rozdílu za použití vytvořené sčítačky.

V první části programu jsou definována paměťová místa pro operandy a registr. Tato paměťová místa jsou definována pro komunikaci se sčítačkou v procesoru Plasma.

Hlavní funkce `main` obsahuje cyklus pro výpočet více součtů. Jeden výpočet součtu je deklarován vždy tímto způsobem. Převedením čísla z formátu integer do FP a nahráním vstupních operandů do paměťových míst zapsáním jedničky do registru příznaku `run`, který značí požadavek na výpočet. Pokud je požadován rozdíl operandů, musí být nastaven příznak `add/sub` v registru na úroveň log. 1. Dalším krokem je čekaní na výsledek, který je indikovaný v registru příznakem `done` v log. 1. Po dokončení výpočtu je výsledek převeden zpět do formátu integer a je proveden informační zápis o výpočtu. Níže je výsledný výstup po dokončení programu.

```
0000000004 - 0000000002 = 0000000002
 Podteceni NE  Preteceni NE
0000000007 - 0000000007 = 0000000000
 Podteceni NE  Preteceni NE
0000000010 - 0000000012 = -0000000002
 Podteceni NE  Preteceni NE
```

7.2 Popis implementace násobičky

Syntéza obvodu

Syntéza obvodu vypíše totožné varovaní jako u sčítačky o nepoužití hodinového signálu `clk` v jednotlivých blocích.

Syntéza obvodu pro procesor Plasma

Tato syntéza vypíše stejné varování jako obvod pro propojení sčítačky s procesorem Plasma. Navíc vypisuje následující varování. Tyto varování upozorňují, že dané signály se vůbec nemění.

```
WARNING:Xst:1710 - FF/Latch <mulFP/typeNumberA_i1_2> (without init value)
has a constant value of 0 in block <mul_fpga_top>. This FF/Latch will be
trimmed during the optimization process.
WARNING:Xst:1895 - Due to other FF/Latch trimming, FF/Latch <mulFP/
typeNumberA_i1_0> (without init value) has a constant value of 0 in block
<mul_fpga_top>. This FF/Latch will be trimmed during the optimization process.
WARNING:Xst:1895 - Due to other FF/Latch trimming, FF/Latch <mulFP/
numberS_i1_30> (without init value) has a constant value of 0 in block
<mul_fpga_top>. This FF/Latch will be trimmed during the optimization process.
WARNING:Xst:1895 - Due to other FF/Latch trimming, FF/Latch <mulFP/
numberS_i1_29> (without init value) has a constant value of 0 in block
<mul_fpga_top>. This FF/Latch will be trimmed during the optimization process.
```

Časová analýza

Z následující zprávy o výsledcích statické časové analýzy je určena minimální hodinová perioda, která je rovna 18,9 ns, tedy cca 52,9 MHz. Při počítání této analýzy byla využita vlastnost vyvážení registrů (register balancing).

```
Timing constraint: Default period analysis for net "clk_BUFGP"
  167608880 paths analyzed, 645 endpoints analyzed, 0 failing endpoints
  0 timing errors detected. (0 setup errors, 0 hold errors)
  Minimum period is  18.907ns.
 --------------------------------------------------------------------------------
 Delay (setup path):     18.907ns (data path - clock path skew + uncertainty)
   Source:               i_mantissaMul/Mmux_mantissaB_i25_FRB (FF)
   Destination:          mantissaM_i2_49 (FF)
   Data Path Delay:      18.816ns (Levels of Logic = 4)
   Clock Path Skew:      -0.056ns (0.421 - 0.477)
   Source Clock:         clk_BUFGP rising
   Destination Clock:    clk_BUFGP rising
   Clock Uncertainty:    0.035ns
```

Časová analýza obvodu pro procesor Plasma

Z následující zprávy statické časové analýzy bloku propojující procesor Plasma a obvodu násobičky je určena minimální hodinová perioda. Tato perioda je 19,6 ns, tedy cca 51 MHz.

Tato frekvence je postačující, aby obvod násobičky mohl pracovat s procesorem Plasma, který pracuje na frekvenci 50 MHz. Při počítání této analýzy byla využita vlastnost vyvážení registrů (register balancing).

```
Timing constraint: Default period analysis for net "clk_BUFGP"
 260031354 paths analyzed, 756 endpoints analyzed, 0 failing endpoints
 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum period is  19.600ns.
--------------------------------------------------------------------------------
Delay (setup path):     19.600ns (data path - clock path skew + uncertainty)
  Source:               mulFP/mantissaM_i2_39 (FF)
  Destination:          mulFP/underflow (FF)
  Data Path Delay:      19.504ns (Levels of Logic = 18)
  Clock Path Skew:      -0.061ns (0.418 - 0.479)
  Source Clock:         clk_BUFGP rising
  Destination Clock:    clk_BUFGP rising
  Clock Uncertainty:    0.035ns
```

Implementační parametry

Tabulka 7.3 zobrazuje výsledné hodnoty implementace při 4 stupňovém pipeline a pro porovnání také hodnoty obvodu bez pipeline. Pro získaní lepší minimální hodinové periody při 4 stupňovém pipeline je využita funkce pro vyvážení registrů (register balancing).

	Bez pipeline	**4 st. pipeline**
Počet použitých registrů	2	395
Počet použitých LUT (tabulek pro kombinační funkce)	681	1288
Celkem obsazených řezů (slices)	269	524
Nejdelší kombinační cesta (ns)	76,14	19,6
Maximální frekvence (MHz)	13,13	51

Tabulka 7.3: Implementační parametry obvodu násobičky

7.2.1 Příklad softwaru pro výpočet na procesoru Plasma

V příloze D.2 je příklad ukázkového programu pro násobičku pracující s procesorem Plasma.

V první části programu jsou definována paměťová místa pro operandy a registr. Tato paměťová místa jsou definována pro komunikaci s násobičkou v procesoru Plasma.

Hlavní funkce obsahuje cyklus pro provedení více výpočtů za sebou. Jeden cyklus probíhá takto: Vstupní hodnoty jsou z formátu integer převedeny do formátu plovoucí řádové čárky. Poté jsou hodnoty zapsány na paměťové místa, kde jsou vstupy násobičky. Do registru je zapsán `start` bit a program čeká na vypočtení výsledku kontrolu bitu `done` v registru. Poté proběhne převod hodnot zpět do formátu integer a informační zápis do souboru. Výsledný informační soubor po dokončení programu je zobrazen pod textem.

```
0000000004 * 0000000002 = 0000000008
Podteceni NE  Preteceni NE
0000000304 * 0000000007 = 0000002128
Podteceni NE  Preteceni NE
0000000604 * 0000000012 = 0000007248
Podteceni NE  Preteceni NE
```

7.3 Popis implementace děličky

Syntéza obvodu

Syntéza vypisuje varování o nepoužití signálu `clk` v jednotlivých blocích. Dále vypisuje několik upozornění `Due to other FF/Latch trimming, FF/Latch <i_mantissaDiv/p_q_2> has a constant value of 0 in block <top_division_fp>`, která varují, že dané signály jsou neměněné a jejich hodnota je konstantní a proto budou optimalizovány (znění celého varování je pod textem).

```
Due to other FF/Latch trimming, FF/Latch <i_mantissaDiv/p_q_2> has a constant
value of 0 in block <top_division_fp>. This FF/Latch will be trimmed during
the optimization process.
```

Syntéza obvodu pro procesor Plasma

Syntéza vypisuje varování jako bloky předchozích aritmetických operací. Jedná se o varovaní: defaultní hodnoty jsou při syntéze ignorovány, část signálu `address` není použita, signál `clk` není použit v blocích, hodnoty `divFP/i_mantissaDiv/p_q` jsou konstantní a budou optimalizovány.

Časová analýza

Z následující statické časové analýzy je určena minimální hodinová perioda, která je 12.8 ns tj. cca 77,9 MHz.

```
Timing constraint: Default period analysis for net "clk_BUFGP"
  2276075 paths analyzed, 718 endpoints analyzed, 0 failing endpoints
  0 timing errors detected. (0 setup errors, 0 hold errors)
  Minimum period is  12.835ns.
 --------------------------------------------------------------------------------
 Delay (setup path):     12.835ns (data path - clock path skew + uncertainty)
   Source:               i_mantissaDiv/v_q_24 (FF)
   Destination:          i_block_result/numberResult_ii_29 (FF)
   Data Path Delay:      12.765ns (Levels of Logic = 11)
   Clock Path Skew:      -0.035ns (0.410 - 0.445)
   Source Clock:         clk_BUFGP rising
   Destination Clock:    clk_BUFGP rising
   Clock Uncertainty:    0.035ns
```

Časová analýza obvodu pro procesor Plasma

Z následující zprávy statické časové analýzy bloku propojující procesor Plasma a obvodu děličky je určena minimální hodinová perioda, která je 19,7 ns, tedy cca 50 MHz. Při provádění této analýzy byla využita vlastnost vyvážení registrů (register balancing). Z výsledků analýzy je možné říct, že tato frekvence by měla být postačující pro připojení k procesoru Plasma.

```
 Timing constraint: Default period analysis for net "clk_BUFGP"
 697657715 paths analyzed, 755 endpoints analyzed, 0 failing endpoints
 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum period is  19.745ns.
--------------------------------------------------------------------------------
 Delay (setup path):   19.745ns (data path - clock path skew + uncertainty)
 Source:               divFP/i_n_case/GND_12_o_mantissaA_i[22]_equal_11_o
 <22>5_FRB (FF)
 Destination:          divFP/i_block_result/numberResult_ii_9 (FF)
 Data Path Delay:      19.722ns (Levels of Logic = 14)
 Clock Path Skew:      0.012ns (0.462 - 0.450)
 Source Clock:         clk_BUFGP rising
 Destination Clock:    clk_BUFGP rising
 Clock Uncertainty:    0.035ns
```

Implementační parametry

Tabulka 7.4 zobrazuje hodnoty implementace děličky řízené čítačem.

Počet použitých registrů	154
Počet použitých LUT (tabulek pro kombinační funkce)	841
Celkem obsazených řezů (slices)	294
Nejdelší kombinační cesta (ns)	12,83
Maximální frekvence (MHz)	77,9

Tabulka 7.4: Implementační parametry obvodu děličky

7.3.1 Příklad softwaru pro výpočet na procesoru Plasma

V příloze D.3 je ukázka programu v jazyce C pro ověření integrace děličky s procesorem Plasma.

Na začátku programu jsou definovaná paměťová místa pro operandy a registr. Tato paměťová místa jsou definována pro komunikaci děličky s procesorem Plasma.

Hlavní funkce programu obsahuje cyklus, který umožňuje provést více ukázkových výpočtů. Výpočet pro jednoho dělení je prováděn takto: Po převodu vstupních hodnot na FP čísla je nastaven `start` bit v registru a program čeká na dokončení výpočtu kontrolou bitu `done` v log. 1. Po dokončení výpočtu je proveden převod zpět do integeru a zapíšou se informace o výpočtu. Výsledek programu je pod textem. U posledního výpočtu je prováděna operace 14 děleno 4 a výsledkem jsou 4. Výsledek je roven 4, poněvadž dělička vypočítala 3,5 a po provedení převodu je výsledek roven 4.

```
0000000010 / 0000000002 = 0000000005
 Podteceni NE  Preteceni NE
0000000012 / 0000000003 = 0000000004
 Podteceni NE  Preteceni NE
0000000014 / 0000000004 = 0000000004
 Podteceni NE  Preteceni NE
```

7.4 Popis implementace konverze na a z fixed point čísla

Syntéza obvodu

Syntéza obvodu pro konverzi z fixed point čísla nevypsala žádné chyby ani varování. Stejně tak i při syntéze obvodu pro konverzi na fixed point číslo nevznikly žádné chyby ani varování.

Syntéza obvodu pro procesor Plasma

Při syntéze obvodu pro procesor Plasma vznikly tato varování.

```
WARNING:HDLCompiler:314 - "N:/P.20131013/rtf/vhdl/xst/src/std_1164.vhd" Line
1037: Choice with meta-value 'U' is ignored for synthesis
WARNING:Xst:647 - Input <address<3:2>> is never used. This port will be
preserved and left unconnected if it belongs to a top-level block or it belongs
to a sub-block and the hierarchy of this sub-block is preserved.
```

První varování upozorňuje, že výchozí hodnoty jsou při syntéze ignorovány. Další upozornění je z důvodu, že není použita část sběrnice `address`. Toto varování se několikrát opakuje.

Časová analýza

Z statické časové analýzy je určena minimální hodinová perioda konverze z FX čísla, která je 14,55 ns tj. cca 68,7 MHz. Z analýzy pro konverzi na FX číslo je zjištěna minimální hodinová perioda 10,6 ns tedy cca 94,3 MHz.

Časová analýza obvodu pro procesor Plasma

Z následující časové analýzy je určena minimální hodinová parioda, která je 12,8 ns tedy cca 78 MHz. Tato frekvence je postačující pro připojení obvodu k procesoru Plasma.

```
 Timing constraint: Default period analysis for net "clk_BUFGP"
  8924515 paths analyzed, 395 endpoints analyzed, 0 failing endpoints
  0 timing errors detected. (0 setup errors, 0 hold errors)
  Minimum period is  12.743ns.
 -------------------------------------------------------------------------------
 Delay (setup path):     12.743ns (data path - clock path skew + uncertainty)
   Source:               FXtoFP/numberFX_i_0 (FF)
   Destination:          FXtoFP/numberFP_19 (FF)
   Data Path Delay:      12.658ns (Levels of Logic = 13)
   Clock Path Skew:      -0.050ns (0.305 - 0.355)
   Source Clock:         clk_BUFGP rising
   Destination Clock:    clk_BUFGP rising
   Clock Uncertainty:    0.035ns
```

Implementační parametry

Tabulka 7.5 zobrazuje hodnoty obvodu pro konverzi na a z fixed point čísla.

	z FX na FP	z FP na FX
Počet použitých registrů	65	64
Počet použitých LUT (tabulek pro kombinační funkce)	427	200
Celkem obsazených řezů (slices)	172	76
Nejdelší kombinační cesta (ns)	14,55	10,6
Maximální frekvence (MHz)	68,7	94,3

Tabulka 7.5: Implementační parametry obvodu pro konverzi na a z fixed point čísla

7.4.1 Příklad softwaru pro výpočet na procesoru Plasma

V příloze D.4 je ukázka programu v jazyce C pro ověření integrace konvertorů z a na FX čísla s procesorem Plasma.

V první části programu jsou definovány paměťová místa pro operandy a registry. Tato paměťová místa jsou definována v procesoru Plasma pro práci s obvodem pro konverzi z a na FX čísla.

Další část obsahuje cyklus, který umožňuje provádět více operací převodu. Převod probíhá zapsáním hodnoty na paměťové místo pro převod z FX do FP formátu. Pak proběhne zápis bitu `start` a čeká se na výsledek, který je značen log. 1 v bitu `done`. Získané převedené číslo je předáno na vstup převodníku FP na FX. Zapíše se startovací bit a vyčká se na výsledek. Poté proběhne informační zápis do souboru. Vstupní hodnota by se měla shodovat s výstupní. Následuje informační výpis z ukázkového programu.

```
0000000010 -> 1092616192(prevod FX na FP) -> 0000000010(prevod FP na FX)
0000000012 -> 1094713344(prevod FX na FP) -> 0000000012(prevod FP na FX)
0000000014 -> 1096810496(prevod FX na FP) -> 0000000014(prevod FP na FX)
```

Kapitola 8

Závěr

Cílem této diplomové práce bylo navrhnout a ověřit správnou funkci základních znovupoužitelných bloků pro aritmetické operace (násobení, sčítání, dělení, konverze na a z fixed point čísla). Provést integraci bloků s procesorem Plasma a demonstrovat jejich funkci.

V práci jsou navrženy znovupoužitelné bloky pro aritmetické operace v FPGA Spartan6. Všechny bloky umožňují nastavení formátu zpracovávaných hodnot pomocí generických konstant.

Na všech blocích byla provedena verifikace (jak na RTL úrovni tak na hradlové úrovni) a ověřena jejich funkčnost při 32 bitové šířce zpracovávaných hodnot. Verifikace proběhla úspěšně na všech verifikovaných blocích.

Každý blok byl pomocí propojovacího obvodu připojen k procesu Plasma. Připojení k procesoru je provedeno namapováním do paměti. Každému bloku byl určeno specifické adresní místo pro vstupy, výstup a registr. Propojení proběhlo pouze na úrovni testbenche, který je ve zdrojových kódech procesoru. Po kompilaci programu pro daný typ operace byla spuštěna simulace a vykonání přeloženého programu.

Literatura

[1] STANDARDS COMMITTEE OF THE IEEE COMPUTER SOCIETY, *An American National Standard, IEEE Standard for Binary Floating-Point Arithmetic,* New York: The Institute of Electrical and Electronics Enginneers, 1985
http://twins.ee.nctu.edu.tw/~tjlin/courses/co01/IEEE754.pdf
[cit. 2014-2-24]

[2] *Floating Point Tutorial*
http://www.rfwireless-world.com/Tutorials/floating-point-tutorial.html
[cit. 2014-2-24]

[3] doc. Ing. Alois Pluháček, CSc., *Projektování logiky počítačů,*
1. vyd Praha: České vysoké učení technické v Praze, 1992, 185s, ISBN 80-01-00813-4

[4] TIŠNOVSKÝ PAVEL, *Binární reprezentace numerických hodnot v FX formátu*
http://www.root.cz/clanky/binarnireprezentace-numerickych-hodnot-v-fx-formatu/#ic=serial-box&icc=text-title
[cit. 2014-1-10]

[5] *Algoritmy I - cvičení z předmětu ALGI*
http://www.cs.vsb.cz/ochodkova/courses/ALGI/algI_cv2.pdf
[cit. 2014-1-10]

[6] TIŠNOVSKÝ PAVEL, *Norma IEEE 754 a příbuzní: formáty plovoucí řádové tečky*
http://www.root.cz/clanky/norma-ieee-754-a-pribuzni-formaty-plovouci-radovetecky/#ic=serial-box&icc=text-title
[cit. 2014-1-10]

[7] *IEEE 754 In Wikipedia*
http://cs.wikipedia.org/wiki/IEEE_754
[cit. 2014-1-10]

[8] ANTON CERVIN, *Fix Point Implementation of Control Algorithms*
http://www.artist-embedded.org/docs/Events/2009/EmbeddedControl/SLIDES/FixPoint.pdf
[cit. 2014-2-10]

[9] Preethi Sudha Gollamudi, M. Kamaraju, *Design Of High Performance IEEE- 754 Single Precision (32 bit) Floating Point Adder Using VHDL,* International Journal of

Engineering Research & Technology, pp.2264-2275, Vol. 2 Issue 7, July - 2013, ISSN: 2278-0181
http://www.ijert.org/browse/volume-2-2013/july-2013-edition?download=4553%3Adesign-of-high-performance-ieee-754-single-precision-32-bit-floating-point-adder-usi
start=330
[cit. 2014-2-10]

[10] Gokul Govindu, Ling Zhuo, Seonil Choi and Viktor Prasanna, *Analysis of High-performance Floating-point Arithmetic on FPGAs,* pp.1-8, Vol. 3, 2004
http://pdf.aminer.org/000/215/858/pipelined_floating_point_arithmetic_optimzed_for_fpga_architectures.pdf
[cit. 2014-2-10]

[11] ŠŤASTNÝ JAKUB, *FPGA prakticky,*
1. vyd Praha: BEN - technická literatura, 2010, 200s, ISBN 978-80-7300-261-9

[12] L. Rajesh, Prashant .V. Joshi and Dr.S.S. Manvi, *An FPGA based Implementation of Floating-point Multiplier,* International Conference on Computing and Control Engineering, pp.1-5, 12 & 13 April, 2012, ISBN 978-1-4675-2248-9
http://www.iccce.co.in/Papers/ICCCECE372.pdf
[cit. 2014-3-6]

[13] Anuja J. Thakkar, Abdel Ejnioui, *Design and Implementation of Double Precision Floating Point Division and Square Root on FPGAs,* Aerospace Conference, pp.1-7, 2006, ISBN 0-7803-9545-X
[cit. 2014-4-10]

[14] Xin Fang and Miriam Leeser, *Vendor Agnostic, High Performance, Double Precision Floating Point Division for FPGAs,* High Performance Extreme Computing Conference, pp.1-5, 10-12 Sept. 2013, ISBN 978-1-4799-1364-0
[cit. 2014-4-10]

[15] *Data Types in Turbo Pascal*
http://www.dfstermole.net/Pascal/hdatatypes.html#real
[cit. 2014-4-18]

[16] Finnbarr P. Murphy, *Half-Precision Floating Point Format,* pp.1-3, 04-21-2014
http://blog.fpmurphy.com/2008/12/half-precision-floating-point-format_14.html?output=pdf
[cit. 2014-4-19]

[17] MAHENDRA KUMAR SONI, *FPGA Implementation of IEEE 754 Standard Based Arithmetic Unit for Floating Point Numbers,* pp.1-72, June 2009
http://dspace.thapar.edu:8080/dspace/bitstream/10266/981/1/thesis_60761008.pdf
[cit. 2014-4-1]

[18] Steve Rhoads, *Plasma - most MIPS I(TM) opcodes,* zdrojové kódy, 25 Sep 2001,
http://opencores.org/project,plasma,overview
[cit. 2014-4-25]

Příloha A

Seznam použitých zkratek

FP Floating Point (pohyblivá řádová čárka)

FPGA Field Programmable Gate Array

FX Fixed Point (pevná řádová čárka)

LUT LookUp Table (vyhledávací tabulka)

NaN Not a Number

RISC Reduced Instruction Set Computer (procesor s redukovaným souborem instrukcí)

RTL Register Transfer Level

SRT Sweeney Robertson Tocher (metoda dělení)

VHDL Very high speed integrated circuit Hardware Description Language

Příloha B

Použitý software, verze aplikací

- ISE Design Suite 14.7
- ModelSim PE Student Edition 10.3
- ISim 14.7(nt64)
- GNU Make version 3.81
- Plasma Version 3
- GCC ELF compiler for Lite MIPS (Plasma CPU) 2.90.23 980102 (egcs-1.0.1 release)

Příloha C

Obsah přiloženého CD

`./Text/`	Diplomová práce ve formátu .pdf
`./Sčítačka/Design/`	Zdrojové kódy sčítačky
`./Sčítačka/Tb/`	Testbenche pro sčítačku
`./Násobička/Design/`	Zdrojové kódy násobičky
`./Násobička/Tb/`	Testbenche pro násobičku
`./Dělička/Design/`	Zdrojové kódy děličky
`./Dělička/Tb/`	Testbenche pro děličku
`./Konverze z FX/Design/`	Zdrojové kódy pro konverzi z FX čísla
`./Konverze z FX/Tb/`	Testbenche pro konverzi z FX čísla
`./Konverze z FP/Design/`	Zdrojové kódy pro konverzi z FP čísla
`./Konverze z FP/Tb/`	Testbenche pro konverzi z FP čísla
`./Procesor Plasma/`	Zdrojové kódy procesor Plasma

Příloha D

Zdrojové kódy ukázkových programů

D.1 Příklad kódu na využití sčítačky s procesorem Plasma

```
/*adder.c*/
int putchar(int value);
int puts(const char *string);

#define MemoryRead(A) (*(volatile unsigned int*)(A))
#define MemoryWrite(A,V) (*(volatile unsigned int*)(A))=(V)

#define fx_space  0x60000010
#define fp_space  0x60000020
#define op1_space 0x60000030
#define op2_space 0x60000040
#define op3_space 0x60000050
#define register  0x60000060

char *itoa10s(unsigned long num)
{
   static char buf[12];
   int i;
   buf[10] = 0;
   if ((num & 0x80000000) == 0x80000000)
   {
     num = -num;
     puts("-");
   }
   for (i = 9; i >= 0; --i)
   {
      buf[i] = (char)((num % 10) + '0');
      num /= 10;
   }
   return buf;
```

```
}

int main()
{
  unsigned long i=0;
  for(i = 0; i<3; i++)
  {
    MemoryWrite(fx_space, 4+3*i); //zapsani cisla A pro prevod FX na FP
    MemoryWrite(op1_space, MemoryRead(fp_space)); //zapsani cisla A
    MemoryWrite(fx_space, 2+5*i); //zapsani cisla B pro prevod FX na FP
    MemoryWrite(op2_space, MemoryRead(fp_space)); //zapsani cisla B
    MemoryWrite(register, 17); //zapsani do registru hodnota run
    while (1) // cekani na vysledek
    {
      if ((MemoryRead(register) & 2 )== 2)
      break;
    }
    MemoryWrite(fp_space, MemoryRead(op1_space)); //zapsani op1 pro prevod
    puts(itoa10s(MemoryRead(fx_space)));
    if ((MemoryRead(register) & 16 )== 16)
      puts(" - ");
    else
      puts(" + ");
    MemoryWrite(fp_space, MemoryRead(op2_space)); //zapsani op2 pro prevod
    puts(itoa10s(MemoryRead(fx_space)));
    puts(" = ");
    MemoryWrite(fp_space, MemoryRead(op3_space)); //zapsani op3 pro prevod
    puts(itoa10s(MemoryRead(fx_space)));
    putchar ('\n');
    puts(" Podteceni ");
    if ((MemoryRead(register) & 8 )== 8)
      puts("ANO ");
    else
      puts("NE ");
    puts(" Preteceni ");
    if ((MemoryRead(register) & 4 )== 4)
      puts("ANO ");
    else
      puts("NE ");
    putchar ('\n');
  }
}
```

D.2 Příklad kódu na využití násobičky s procesorem Plasma

```
/*multiplier.c*/
int putchar(int value);
int puts(const char *string);

#define MemoryRead(A) (*(volatile unsigned int*)(A))
#define MemoryWrite(A,V) (*(volatile unsigned int*)(A))=(V)

#define fx_space  0x60000010
#define fp_space  0x60000020
#define op1_space 0x60000070
#define op2_space 0x60000080
#define op3_space 0x60000090
#define register  0x600000A0

char *itoa10s(unsigned long num)
{
   static char buf[12];
   int i;
   buf[10] = 0;
   if ((num & 0x80000000) == 0x80000000)
   {
     num = -num;
     puts("-");
   }
   //else puts(" ");
   for (i = 9; i >= 0; --i)
   {
      buf[i] = (char)((num % 10) + '0');
      num /= 10;
   }
   return buf;
}


int main()
{
  unsigned long i=0;
  for(i = 0; i<3; i++)
  {
    MemoryWrite(fx_space, 4+300*i); //zapsani cisla A pro prevod FX na FP
    MemoryWrite(op1_space, MemoryRead(fp_space)); //zapsani cisla A
    MemoryWrite(fx_space, 2+5*i); //zapsani cisla B pro prevod FX na FP
    MemoryWrite(op2_space, MemoryRead(fp_space)); //zapsani cisla B
    MemoryWrite(register, 1); //zapsani do registru hodnota run
```

```
    while (1) // cekani na vysledek
    {
      if ((MemoryRead(register) & 2 )== 2)
      break;
    }
    MemoryWrite(fp_space, MemoryRead(op1_space)); //zapsani op1 pro prevod
    puts(itoa10s(MemoryRead(fx_space)));
    puts(" * ");
    MemoryWrite(fp_space, MemoryRead(op2_space)); //zapsani op2 pro prevod
    puts(itoa10s(MemoryRead(fx_space)));
    puts(" = ");
    MemoryWrite(fp_space, MemoryRead(op3_space)); //zapsani op3 pro prevod
    puts(itoa10s(MemoryRead(fx_space)));
    putchar ('\n');
    puts(" Podteceni ");
    if ((MemoryRead(register) & 8 )== 8)
      puts("ANO ");
    else
      puts("NE ");
    puts(" Preteceni ");
    if ((MemoryRead(register) & 4 )== 4)
      puts("ANO ");
    else
      puts("NE ");
    putchar ('\n');
  }
}
```

D.3 Příklad kódu na využití děličky s procesorem Plasma

```
/*div.c*/
int putchar(int value);
int puts(const char *string);

#define MemoryRead(A) (*(volatile unsigned int*)(A))
#define MemoryWrite(A,V) (*(volatile unsigned int*)(A))=(V)

#define fx_space  0x60000010
#define fp_space  0x60000020
#define op1_space 0x600000B0
#define op2_space 0x600000C0
#define op3_space 0x600000D0
#define register  0x600000E0

char *itoa10s(unsigned long num)
```

```
{
   static char buf[12];
   int i;
   buf[10] = 0;
   if ((num & 0x80000000) == 0x80000000)
   {
     num = -num;
     puts("-");
   }
   //else puts(" ");
   for (i = 9; i >= 0; --i)
   {
      buf[i] = (char)((num % 10) + '0');
      num /= 10;
   }
   return buf;
}

int main()
{
  unsigned long i=0;
  for(i = 0; i<3; i++)
  {
    MemoryWrite(fx_space, 10+2*i); //zapsani cisla A pro prevod FX na FP
    MemoryWrite(op1_space, MemoryRead(fp_space)); //zapsani cisla A
    MemoryWrite(fx_space, 2+1*i); //zapsani cisla B pro prevod FX na FP
    MemoryWrite(op2_space, MemoryRead(fp_space)); //zapsani cisla B
    MemoryWrite(register, 1); //zapsani do registru hodnota run
    while (1) // cekani na vysledek
    {
      if ((MemoryRead(register) & 2 )== 2)
      break;
    }
    MemoryWrite(fp_space, MemoryRead(op1_space)); //zapsani op1 pro prevod
    puts(itoa10s(MemoryRead(fx_space)));
    puts(" / ");
    MemoryWrite(fp_space, MemoryRead(op2_space)); //zapsani op2 pro prevod
    puts(itoa10s(MemoryRead(fx_space)));
    puts(" = ");
    MemoryWrite(fp_space, MemoryRead(op3_space)); //zapsani op3 pro prevod
    puts(itoa10s(MemoryRead(fx_space)));
    putchar ('\n');
    puts(" Podteceni ");
    if ((MemoryRead(register) & 8 )== 8)
      puts("ANO ");
    else
```

```
      puts("NE ");
    puts(" Preteceni ");
    if ((MemoryRead(register) & 4 )== 4)
      puts("ANO ");
    else
      puts("NE ");
    putchar ('\n');
  }
}
```

D.4 Příklad kódu na využití konverze s procesorem Plasma

```
/*conv.c*/
int putchar(int value);
int puts(const char *string);

#define MemoryRead(A) (*(volatile unsigned int*)(A))
#define MemoryWrite(A,V) (*(volatile unsigned int*)(A))=(V)

#define fx_space  0x60000010
#define fp_space  0x60000020
#define op1_space 0x60000100
#define op2_space 0x60000110
#define register  0x60000120

#define op3_space 0x60000130
#define op4_space 0x60000140
#define register2 0x60000150

char *itoa10s(unsigned long num)
{
   static char buf[12];
   int i;
   buf[10] = 0;
   if ((num & 0x80000000) == 0x80000000)
   {
     num = -num;
     puts("-");
   }
   //else puts(" ");
   for (i = 9; i >= 0; --i)
   {
      buf[i] = (char)((num % 10) + '0');
      num /= 10;
```

```
   }
   return buf;
}

int main()
{
  unsigned long i=0;
  unsigned long p;
  for(i = 0; i<3; i++)
  {
    MemoryWrite(fx_space, 10+2*i); //zapsani cisla A pro prevod FX na FP
    MemoryWrite(op1_space,  10+2*i); //zapsani cisla A
    MemoryWrite(register, 1); //zapsani do registru hodnota run
    while (1) // cekani na vysledek
    {
      if ((MemoryRead(register) & 2 )== 2)
      break;
    }
    p = MemoryRead(op2_space);
    MemoryWrite(op3_space, p); //prevod na zpet
    MemoryWrite(register2, 1); //zapsani do registru
    while (1) // cekani na vysledek
    {
      if ((MemoryRead(register2) & 2 )== 2)
      break;
    }
    puts(itoa10s(MemoryRead(op1_space)));
    puts(" -> ");
    puts(itoa10s(p));
    puts("(prevod FX na FP)");
    puts(" -> ");
    puts(itoa10s(MemoryRead(op4_space)));
    puts("(prevod FP na FX)");
    putchar ('\n');
  }
}
```