VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NUMERICKÝ DĚLÍCÍ INTEGRÁTOR SSI

SSI DIVIDING NUMERICAL INTEGRATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Suntcov Roman
AUTHOR

VEDOUCÍ PRÁCE — Ing. Václav Šátek Ph.D.
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Suntcov Roman**
Obor: Informační technologie
Téma: **Numerický dělící integrátor SSI**
SSI Dividing Numerical Integrator
Kategorie: Modelování a simulace

Pokyny:

1. Seznamte se s paralelním numerickým řešením obyčejných diferenciálních rovnic.
2. Seznamte se s numerickou integrací a principem dělícího numerického integrátoru pro přímé využití Taylorovy řady.
3. Seznamte se s variantami paralelních, sériových a sériově-paralelních integrátorů (PPI, SPI, SSI).
4. Navrhněte a implementujte v provedení FPGA řídící systém SSI integrátoru pro zadaný řád integrační metody, pro zvolenou délku slova a pro zadaný krok výpočtu.
5. Vytvořte simulátor řídícího systému.

Literatura:

- Kunovský, J.: Modern Taylor series method, Habilitation work, VUT Brno, 1994.
- M. Kubíček, M. Dubcová, D. Janovská: Numerické metody a algoritmy, VŠCHT Praha, 2005.
- M. Kraus: Paralelní výpočetní architektury založené na numerické integraci, disertační práce FIT VUT, 2013.

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

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

Vedoucí: **Šátek Václav, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Práce se zabývá numerickou integrací a operací dělení v hardware. Čtenář je seznámen s numerickým řešením diferenciálních rovnic pomocí několika různých metod, z nichž lze zmínit například Taylorovu řadu. Dále je probrána operace dělení v hardware a způsob jejího provedení v FPGA. Následně je navržen paralelně-paralelní a sériově-paralelní integrátor. Praktickým cílem práce je návrh a implementace sériově-sériového dělícího integrátoru a vytvoření simulátoru pro něj.

Abstract

The thesis deals with numerical integration and hardware division operations. The reader is familiar with the numerical solution of differential equations through several different methods, for example Taylor's series. Furthermore, it is discussed the operation of division in the hardware and the method of its implementation in the FPGA. Subsequently, a parallel-parallel and serial-parallel integrator is designed. The practical aim of the thesis is to design and implement a serial-serial dividing integrator and create a simulator for it.

Klíčová slova

Obyčejná diferenciální rovnice, Taylorova řada, SRT algoritmus, pevná řádová čárka, dělící integrátor, FPGA, Verilog.

Keywords

Differential equations, Taylor series, the SRT algorithm, fixed point, dividing integrator, FPGA, Verilog.

Citace

Suntcov Roman: Numerický dělící integrátor SSI, bakalářská práce, Brno, FIT VUT v Brně, rok 2018

Numerický dělící integrátor SSI

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pána Ing. Václava Šátka Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Roman Suntcov
15. května 2018

Poděkování

Chtěl bych poděkovat mému vedoucímu Ing. Václavu Šátku Ph.D. za odborné vedení práce a za cenné rady během její tvorby. Dále bych chtěl poděkovat celé své rodině, přítelkyni Marině a všem svým kamarádům, kteří mě podporovali po celou dobu studia a byli tu pro mě. Bez nich tato práce nemohla vzniknout.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Tématem mojí práce je numerický sériově-sériový dělící integrátor. Numerický integrátor je komponenta, pomocí které můžeme rychle a jednoduše řešit složité diferenciální rovnice. Moje práce se zaměřuje na operaci dělení a hlavním cílem je navrhnout a implementovat sériově-sériový dělicí integrátor (SSDI) pro výpočet diferenciálních rovnic s operací dělení na FPGA. FPGA (Field Programmable Gate Arrays – programovatelné hradlové pole) je elektronická součástka, která je používána pro vytváření dynamických číslicových obvodů.

Důležitou součástí této práce je operace numerické integrace. V kapitole 2 jsou rozebrány numerické řešení diferenciálních rovnic a stručně popsány základní pojmy numerické integrace a některé jednokrokové a vícekrokové metody. Dále se seznámíme s Taylorovou řadou, Eulerovou metodou a metodou Runge-Kutta. Na konci této kapitoly je ukázáno řešení diferenciální rovnice s operací dělení pomocí Taylorovy řady.

Třetí kapitola se zaměřuje na provedení operace dělení v integrátorech. Protože je tato operace velmi časově náročná a složitá pro vypočet, existuje několik metod pro její zjednodušení. V této kapitole budou rozebrány některé z těchto metod.

Další kapitola je zaměřena na různé druhy integrátorů, z nichž jsou zmíněny tři – paralelní, sériový a sériově-paralelní typ integrátoru. Kapitola obsahuje návrh a implementaci sériově-sériového integrátoru pro vypočet diferenciálních rovnic s operací dělení v obvodu FPGA.

V páté kapitole je popsán návrh a implementace SSDI. Implementace sériově-sériového dělícího integrátoru je nejsložitější a nejnáročnější ze všech typů. V této kapitole je popsán proces návrhu a jeho implementace. Kapitola se zabývá tím, jak byl integrátor upraven, jaké nové komponenty byly zavedeny do systému, podle kterého algoritmu pracuje kontrolér, a ve kterém jazyce a proč je program implementován.

Šestá kapitola se zabývá grafickým simulátorem SSDI naprogramovaným v jazyce Java. Aplikace slouží k názornému zobrazení algoritmu sériově-sériového dělícího integrátoru.

2 Numerická integrace

V současné době se pro výpočet diferenciálních rovnic už téměř nepoužívají analytické metody, a to z několika důvodů – metody jsou příliš složité nebo je není možné použít. Navíc rozvoj počítačových technologií velmi usnadnil vypočet diferenciálních rovnic pomocí numerických metod.

Existuje velké množství různých numerických metod pro řešení diferenciálních rovnic, které se liší ve dvou hlavních kritériích – *přesnost* a *rychlost*. Jedno je vždy potlačeno na úkor druhého. Například zvolením menšího kroku nebo zvýšením řádu numerické metody dosáhneme větší přesnosti, ale zároveň ztratíme řešením úlohy více času. Důležitou podmínkou je zvolit tyto hodnoty optimálně.

Přesnost výpočtu je navíc ovlivněna i dalšími faktory. Prvním z nich je *chyba zaokrouhlení* (omezená přesností výpočetní techniky). Druhým faktorem je *chyba numerické metody* (čím má metoda vyšší řád, tím je přesnější) [1], [2], [4].

Numerické metody se rozlišují na jednokrokové a vícekrokové. Jednokrokové metody používají hodnotu z předcházejícího kroku k výpočtu dalšího. Ve vícekrokových metodách je pro výpočet hodnoty dalšího kroku použito několik předcházejících mezivýsledků. Jedna z nejznámějších vícekrokových metod je metoda Adams-Bashforth, která byla vyvinuta již v 19. století a používá se do dnes. V následujících podkapitolách se zaměříme na jednokrokové numerické metody (Taylorova řada, Eulerova metoda, metoda Runge-Kutta).

2.1 Počáteční úloha

Počáteční úloha (*Cauchyho úloha* nebo *problém počáteční hodnoty*) je v matematice v oboru diferenciálních rovnic hledání takového řešení obyčejné diferenciální rovnice, které vyhovuje *počáteční podmínce* [5]. Počáteční podmínka stanovuje, jakých hodnot musí nabývat funkce (případně i její derivace) v určitém bodě svého definičního oboru.

Počáteční úloha je zadána diferenciální rovnicí:

$$y'(t) = f(t, y) \tag{2.1.1}$$

Kde $f : \Omega \subset \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ pro otevřenou množinu Ω, která je otevřenou podmnožinou $\mathbb{R} \times \mathbb{R}^n$ a bodem (*počáteční podmínkou*) v definičním oboru funkce:

$$(t_0 , y_0) \in \Omega \tag{2.1.2}$$

Řešením počáteční úlohy je taková funkce y, která je řešením diferenciální rovnice a vyhovuje podmínce:

$$y(t_0) = y_0 \tag{2.1.3}$$

2.2 Taylorova řada

Taylorova řada je jedna ze základních jednokrokových numerických metod a je vyjádřena následujícím zápisem:

$$y_{i+1} = y_i + hy_i' + \frac{h^2}{2!}y_i'' + \frac{h^3}{3!}y_i''' + \frac{h^4}{4!}y_i'''' + \cdots + \frac{h^n}{n!}y_i^{(n)} \tag{2.2}$$

Proměnná y_i je funkcí v čase $y(t_i)$, h je integrační krok funkce. Jednou z důležitých vlastností této metody je určení přesnosti výpočtu. Výpočet konkrétní hodnoty y_{i+1} je iteračně prováděn až do dosažení zadané přesnosti. To znamená, že výpočet skončí, až když je rozdíl dvou po sobě následujících vypočítaných hodnot menší než zvolená přesnost. Hlavním problémem při vypočtu pomocí Taylorovy řady je nutnost použití vyšších stupňů derivací.

2.3 Eulerova metoda

Eulerova metoda je nejjednodušší jednokroková metoda, která využívá právě dva první členy Taylorovy řady:

$$y_{i+1} = y_i + hy_i' \tag{2.3}$$

y_{i+1} je následující hodnota, která je vypočtena z aktuální hodnoty. Tato metoda se používá pro velmi malé hodnoty h, díky čemuž je výsledek dostatečně přesný. Výhodou Eulerovy metody je rychlost, protože pro vypočet potřebujeme spočítat pouze jednu derivaci.

2.4 Metody Runge-Kutta

Metody Runge-Kutta také vychází z Taylorovy řady, ale na rozdíl od Eulerovy metody jsou přesnější a rychlejší. Mezi známé modifikace těchto metod patří Runge-Kutta 2., 4. a 8. řádu.
Obecný tvar metody Runge-Kutta 2. řádu neboli lichoběžníkové metody:

$$y_{i+1} = y_i + h\left(\frac{1}{2}k_1 + \frac{1}{2}k_2\right) \tag{2.4.1}$$

Z definice počáteční úlohy (2.1.1) vyplývá:

$$k_1 = f(t_i, y_i) \tag{2.4.2}$$
$$k_2 = f(t_i + h\,, y_i + hk_1) \tag{2.4.3}$$

Runge-Kutta 4. řadu je nejznámější a nejpoužívanější ze všech tří modifikací. Její obecný tvar je:

$$y_{i+1} = y_i + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.4.4}$$
$$k_1 = f(t_i, y_i) \tag{2.4.5}$$
$$k_2 = f(t_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_1) \tag{2.4.6}$$
$$k_3 = f(t_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_2) \tag{2.4.7}$$
$$k_4 = f(t_i + h, y_i + hk_3) \tag{2.4.8}$$

2.5 Příklad řešení diferenciální rovnice

V této podkapitole je rozebrán způsob řešení diferenciální rovnice pomocí všech výše uvedených metod. Zvolíme si jednoduchou diferenciální rovnici (počáteční úlohu):

$$y' = y \quad , \quad y(0) = y_0 \tag{2.5.1}$$

Při aplikaci jednotlivých metod na rovnici (2.5.1) dostaneme:

- Eulerova metoda:

$$y_1 = y_0 + DY1_0 \tag{2.5.2}$$
$$DY1_0 = hy_0 \tag{2.5.3}$$

- Metoda Runge-Kutta 4.řadu:

$$y_1 = y_0 + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \tag{2.5.4}$$
$$k_1 = hy_0 \tag{2.5.5}$$
$$k_2 = h(y_0 + \frac{k_1}{2}) \tag{2.5.6}$$
$$k_3 = h(y_0 + \frac{k_2}{2}) \tag{2.5.7}$$
$$k_4 = h(y_0 + \frac{k_3}{2}) \tag{2.5.8}$$

- Metoda Taylorovy řady:
  Z diferenciální rovnice vyplývá, že:

$$y' = y \qquad y' = y'' = y''' = \cdots = y^{(n)} \tag{2.5.9}$$

Tato rovnice potvrzuje podmínku platnosti Taylorovy řady (funkce $f(t)$ a její derivace musí být jednoznačná, konečná a spojitá v intervalu mezi t a $t + h$). Po dosazení výchozí rovnice do Taylorovy řady dostaneme:

$$y_1 = y_0 + hy_0 + \frac{h^2}{2!}y_0 + \frac{h^3}{3!}y_0 + \frac{h^4}{4!}y_0 + \cdots + \frac{h^n}{n!}y_0 \tag{2.5.10}$$

Výše uvedenou rovnici přepíšeme na:

$$y_1 = y_0 + DY1_0 + DY2_0 + DY3_0 + DY4_0 + \cdots + DYN \tag{2.5.11}$$

Pak platí:

$$DY1_0 = hy_0 \tag{2.5.12}$$
$$DY2_0 = \frac{h}{2} DY1_i \tag{2.5.13}$$
$$DY3_0 = \frac{h}{3} DY2_i \tag{2.5.14}$$
$$\vdots$$
$$DYN_0 = \frac{h^n}{n!} y_0 = \frac{h}{n} DYN \tag{2.5.15}$$

Rovnice (2.5.12)-(2.5.15) můžeme používat jako iterační numerický algoritmus pro výpočet diferenciální rovnice metodou Taylorovy řady. Tento způsob je velmi jednoduchý, potřebujeme zvolit jenom počáteční podmínku, integrační krok a řád metody (počet členů Taylorovy řady).

2.6 Řešení diferenciální rovnice s operací dělení

Ve své práci se budu zabývat řešením diferenciálních rovnic s operací dělení. Výpočet jednoduché rovnice numerickou metodou je znázorněn níže.
Zvolíme jednoduchou rovnici:

$$y' = \frac{u}{v} \tag{2.6.1}$$

Její následující derivace je:

$$y'' = \frac{u'v - uv'}{v^2} = \frac{1}{v}(u' - y'v') \tag{2.6.2}$$
$$y''' = \left(\frac{1}{v}(u' - y'v')\right)' = \frac{1}{v}(u'' - 2y''v' - y'v'') \tag{2.6.3}$$
$$\vdots$$

Algoritmus výpočtu je stejný jako u předchozího příkladu, ale místo jednoho členu DYN_i máme členy dva DUN_i a DVN_i:

$$DU1_i = hu_i \qquad DV1_i = hv_i \tag{2.6.4}$$
$$DU2_i = \frac{h}{2} DU1_i \qquad DV2_i = \frac{h}{2} DV1_i \tag{2.6.5}$$
$$DU3_i = \frac{h}{3} DU2_i \qquad DV3_i = \frac{h}{3} DV2_i \tag{2.6.6}$$
$$\vdots \qquad\qquad \vdots$$
$$DUN_i = \frac{h}{n} DUN_i \qquad DVN_i = \frac{h}{n} DVN_i \tag{2.6.7}$$

Následně dosadíme členy DVN_i a DUN_i do zadané rovnice:

$$\frac{DY1_i}{h} = \frac{1}{v_i} u_i \tag{2.6.8}$$

$$\frac{DY2_i}{\frac{h^2}{2!}} = \frac{1}{v}\left(\frac{DU1_i}{h} - \frac{DY1_i}{h} * \frac{DV1_i}{h}\right) \tag{2.6.9}$$

$$\frac{DY3_i}{\frac{h^3}{3!}} = \frac{1}{v}\left(\frac{DU2_i}{\frac{h^2}{2!}} - 2\frac{DY2_i}{\frac{h^2}{2!}} * \frac{DV1_i}{h} - \frac{DV2_i}{\frac{h^2}{2!}} * \frac{DY1_i}{h}\right) \tag{2.6.10}$$

$$\vdots$$

Tyto rovnice upravíme a vyjádříme jednotlivé členy:

$$DY1_i = \frac{hu}{v} \tag{2.6.11}$$

$$DY2_i = \frac{1}{2v}(DU1_i * h - DY1_i * DV1_i) \tag{2.6.12}$$

$$DY3_i = \frac{1}{3v}(DU2_i * h - 2DY2_i * DV1_i - DY1_i * DV2_i) \tag{2.6.13}$$

$$\vdots$$

Výchozí vzorce tvoří tzv. Pascalův trojúhelník, který budeme potřebovat při návrhu a implementaci integrátoru.

2.7 Řešení soustavy diferenciálních rovnic

Výše uvedený způsob můžeme použít pro řešení další skupiny diferenciálních rovnic – pro soustavu nehomogenních lineárních diferenciálních rovnic.
Zvolíme si rovnici:

$$y' = \frac{sin(t)}{e^{at}} \qquad y(0) = \frac{1}{a^2+1} \tag{2.7.1}$$

Výše uvedené rovnice převedeme na ekvivalentní systém diferenciálních rovnic:

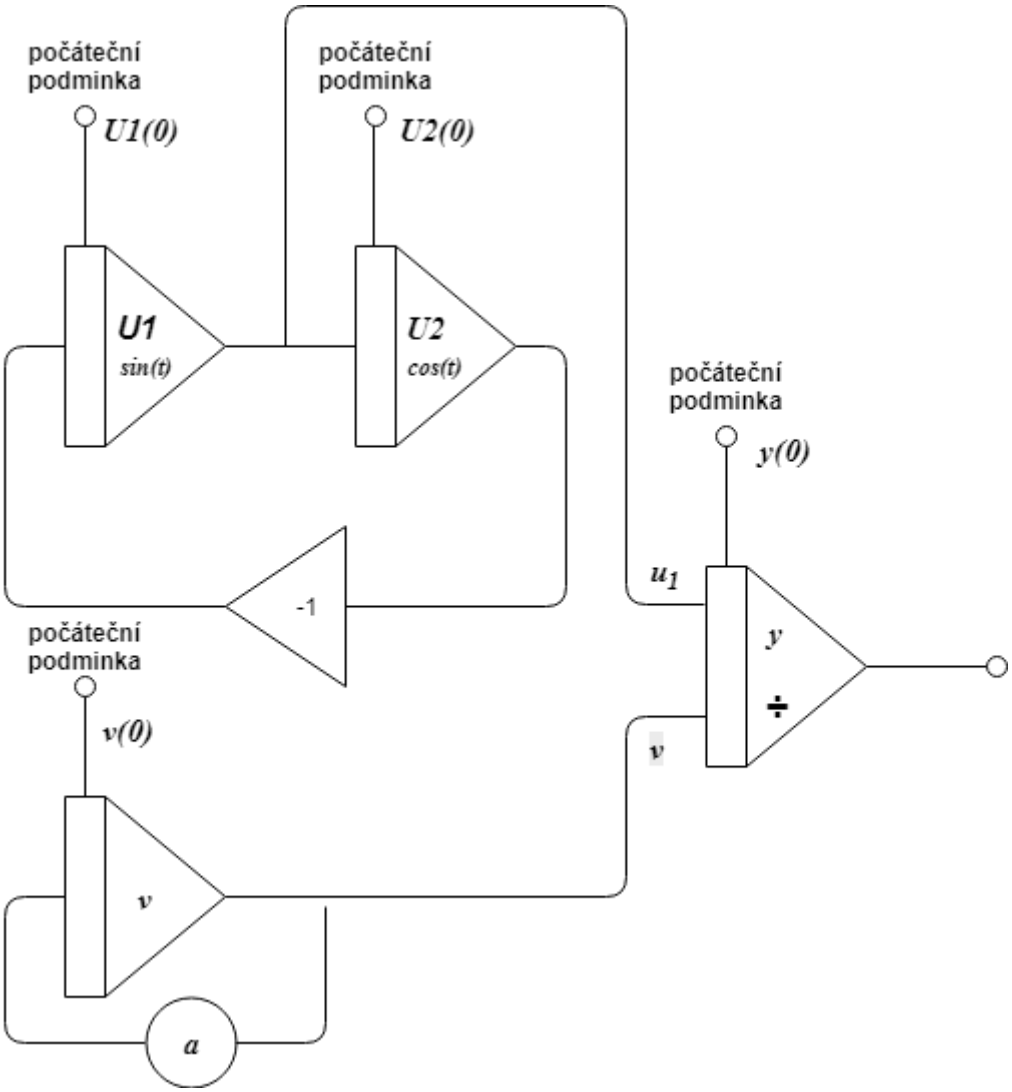
$$y' = \frac{u_1}{v} \qquad y(0) = \frac{1}{a^2+1} \tag{2.7.2}$$

$$u_1' = u_2 \qquad u_1(0) = 0 = \sin(t) \tag{2.7.3}$$

$$u_2' = -u_1 \qquad u_2(0) = 1 = \cos(t) \tag{2.7.4}$$

$$v' = av \qquad v(0) = 1 = e^0 \tag{2.7.5}$$

V dalším kroku překreslíme rovnice na blokové schéma obsahující 3 integrátory (obrázek 2.1).



Obr. 2.1: Schéma zapojení integrátorů

Výsledné schéma vytvoříme pomocí tří jednoduchých integrátorů, jednoho invertoru, jednoho dvouvstupového dělicího integrátoru. Integrátor $U1$ představuje funkci *sin(t)*, $U2$ představuje funkci *cos(t)*. Zapojením těchto dvou integrátorů a invertoru získáme *goniometrickou funkci sin*. Integrátor v reprezentuje funkci e^{at}. Zapojením výsledku *sin* a integrátoru v získáme funkci $\frac{sin(t)}{e^{at}}$. Výstupem dělícího integrátoru y je hodnota rovnice (2.7.1) v čase $i + 1$.

Stejným způsobem můžeme řešit libovolnou diferenciální rovnici, resp. soustavu diferenciálních rovnic. Tento způsob je typický pro analogové počítače, kde se podle blokového schématu provede zapojení jednotlivých prvků (integrátory, invertory a další), které počítají výsledek.

3 Operace dělení

Dělení je aritmetická binární operace mezi dvěma čísly, která je opačná (inverzní) k operaci násobení. Dělení nulou není definováno, proto je nutné ošetřit tento stav nějakou chybou.

Dělení představuje nejkomplikovanější elementární matematickou operaci při realizaci v digitálním obvodu. Při realizaci matematických struktur je upřednostňováno sčítání a odčítání, často i násobení. Významnou roli hraje fakt, že součet, rozdíl nebo součin dvou celých čísel je opět celé číslo, což v případě dělení neplatí. Z těchto důvodů není dělička definována v FPGA (na rozdíl od násobiček nebo sčítaček), proto si ji musíme definovat sami.

Existují velké množství algoritmů dělení, které obvyklé rozdělujeme do dvou skupin – *sekvenční* a *iterační*. Sekvenční algoritmy vypočítají během jedné iterace jedno číslo podílu. Iterační algoritmy vypočítají za jednu iteraci všechny bity výsledku. Oba typy algoritmů pracují do doby stanovené ukončovací podmínkou, která reprezentuje požadovanou *přesnost* a často se vyznačuje jako *epsilon* nebo ε [6]. Do první skupiny patří algoritmy bez návratu (restaurací) k nezápornému zbytku, algoritmy s návratem (restaurací) k nezápornému zbytku, SRT algoritmy a další. Newton-Raphson, Goldshmid a další, jsou příkladem iteračních algoritmů.

Hlavní rovnice pro operaci dělení je:

$$D = Q * d + R, \qquad 0 \leq |R| < d \tag{3.1}$$

Jednotlivé symboly představují:

D – dělenec
d – dělitel
Q – podíl
R – zbytek po dělení

V dalších podkapitolách budou podrobněji popsány jednotlivé sekvenční algoritmy.

3.1 Dělení s návratem k nezápornému zbytku

Algoritmus pracuje takovým způsobem, že odečítá dělitele od průběžného zbytku, a zjišťuje každý bit výsledku. Pokud je vypočítaný zbytek kladný (neboli jeho nejvyšší bit se rovná log. 0), pak $Q_{n-1} = 1$ a $R_{i+1} = 2R_i$, pokud je vypočítaný zbytek záporný (neboli jeho nejvyšší bit se rovná log. 1), pak $Q_{n-1} = 0$ a $R_i = 2R_i - d$, což není správně a je nutné provést restauraci přičítáním dělitele – návrat k nezápornému zbytku. Průběžný zbytek se pak posune o jeden bit doleva a znovu se odečítá dělitel.

3.2 Dělení bez návratu k nezápornému zbytku

Postup algoritmu je podobný předchozímu. Hodnotu jednotlivých bitů výsledku počítáme pomocí nejvyššího bitu předchozího zbytku. Rozdíl spočívá v počítání hodnoty následujícího průběžného zbytku, což je ukázáno níže:

$$pokud\ Q_{n-1} = 1, \qquad tak\ R_{i+1} = 2R_i - d$$
$$pokud\ Q_{n-1} = 0, \qquad tak\ R_{i+1} = 2R_i + d$$

Restaurace se provede až na konci výpočtu. Pokud je průběžný zbytek menší než nula, je přičten k děliteli.

Oba z těchto algoritmů se opakují *n*-krát, kde n je počet bitů dané architektury.

3.3 Příklad dělení bez návratu k nezápornému zbytku

Zvolíme si příklad $21 : 4 = 5\ zb.1$. Nejprve musíme převést čísla do dvojkové soustavy, tedy:

$$21_{10} = 00010101_2, 4_{10} = 0100_2, -4_{10} = 1100_2$$

Pak:

00010101	vyšší bit je 0, => číslo je větší než 0 (> 0), => musíme odečíst dělitele (-d)
+1100	-d
11010101	< 0, => **C4 = 0** (první bit výsledku je 0); +d
1010101x	posuv doleva
+0100	+d
1110101x	< 0, => **C3 = 0**
110101xx	posuv
+0100	+d
000101xx	> 0, => **C2 = 1**
00101xxx	posuv
+1100	-d
11101xxx	< 0, => **C1 = 0**
1101xxxx	posuv
+0100	+d
0001xxxx	zbytek se rovná **1**, což je > 0, => **C0 = 1**, nepotřebujeme provést korekci

Výsledkem je číslo $\mathbf{10100_2 = 5_{10}}$, zbytek je roven **1**, což je správným řešením tohoto příkladu.

3.4 Algoritmus SRT

Název algoritmu je poskládán z iniciál jmen jeho autorů (Sweeney, Robertson a Tocher). Algoritmus SRT patří do skupiny algoritmů bez návratu k nezápornému zbytku. Využívá speciální SRT tabulku, na jejímž základě počítáme hodnoty jednotlivých bitů výsledku dělení. Existují rozšíření algoritmu, které jsou schopné redukovat počet operací při dělení zpracováním více bitů najednou a taky umožňují pracovat s čísly se znaménkem [6]. Při dělení čísel se znaménkem požadujeme, aby byl zbytek kladný, jinak musíme provést korekci.

Nejjednodušší variantou algoritmu je SRT s *radixem 2*, která pracuje po dvou bitech a určuje následující operaci podle tří nejvyšších bitů (včetně znaménkového). Algoritmus navíc používá relativní soustavu čísel, která obsahuje tři číslice – $0,1,\overline{1}$. Takovéto překódování se realizuje pomocí Boothového algoritmu a používá převodovou tabulku 3.1. V praxi se často používají algoritmy s vyšším radixem. Při použití tří bitů se jedná o SRT s *radixem 4*, při použití čtyř bitů je to SRT s *radixem 8*.

Překódované číslo	Sousední pravý bit	Boothuv kód
0	0	0
0	1	1
1	0	$\overline{1}$
1	1	0

Tabulka 3.1: Boothové překódování s radixem 2.

Hlavním cílem SRT algoritmu bylo zvýšení rychlosti algoritmu dělení bez návratu k nezápornému zbytku. Z tohoto důvodu bylo provedeno několik úprav tohoto algoritmu. SRT s radixem 2 můžeme zapsat jako:

$$q_i = \begin{cases} 1 & tak \quad 2r_{i-1} \geq D \\ 0 & tak \quad -D \leq 2r_{i-1} < D \\ -1 & tak \quad 2r_{i-1} < D \end{cases}$$

Obr. 3.1: SRT s radixem 2

V tomto vzorci už máme přidanou podmínku pro hodnotu 0, tato podmínka odstraní neustálé odečítání a přičítání.

3.5 Příklad dělení pomocí algoritmu SRT s radixem 2

Tento příklad jsem převzal ze slajdu k předmětu INP [6]. Našim úkolem je spočítat výsledek následujícího příkladu:

$$-39 : -6 = 7 \text{ (zbytek 3)}.$$

Nejdříve musíme převést čísla do doplňkového kódu dvojkové soustavy:

$$-39 = 11011001_2, 6 = 0110_2, -6 = 1010_2$$

Pro řešení budeme používat tabulku 3.2, podle které se provede přičtení nebo odečtení dělitele a určí se hodnota bitu podílu:

	d > 0		d < 0	
Ri	bit podílu	operace	bit podílu	operace
000 111	0	žádná	0	žádná
001 010 011	1	-d	-1	+d
101 110 100	-1	+d	1	-d

Tabulka 3.2 Dělení pomocí algoritmu SRT

1	**110**11001	hodnota bitu podle tabulky se rovná **C5 = 1**
	0110	-d
	00111001	
-1	**011**1001x	posuv doleva, **C4 = -1**
	1010	+d
	0001001x	
-1	**001**001xx	posuv, **C3 = -1**
	1010	+d
	110001xx	
1	**100**01xxx	posuv, **C2 = 1**
	0110	-d

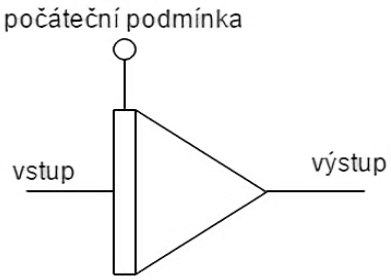
	11101xxx	
1	**<u>110</u>**1xxxx	posuv, **C1 = 1**
	<u>0110</u>	<u>-d</u>
	0011	zbytek 3

Výsledek je $Q = 1 - 1 - 1\,1\,1 = 16 - 8 - 4 + 2 + 1 =$ **7 *zb*. 3.**

4 Numerické integrátory

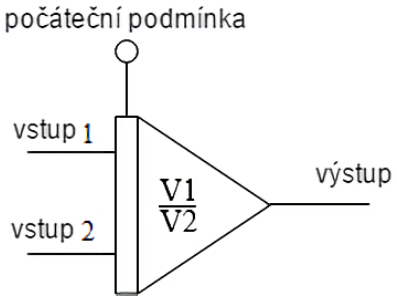
Integrátor je elektrotechnická součástka, která v obvodu provádí matematickou funkci integrování. Napětí (signál) na výstupu je integrál vstupního napětí podle času. Pro výpočet je nutné nastavit počáteční podmínku a integrační krok. Ve své práci se budu zabývat implementací numerického dělicího integrátoru (obrázek 4.2), který má na rozdíl od jednoduchého integrátoru (obrázek 4.1) dva vstupy.

Blokové znázornění jednoduchého integrátoru vypadá následovně:



Obr. 4.1: Jednoduchý integrátor

Blokové znázornění dvouvstupového dělicího integrátoru je:



Obr. 4.2: Dělicí integrátor

V následujících kapitolách jsou uvedeny tři varianty numerických integrátorů:

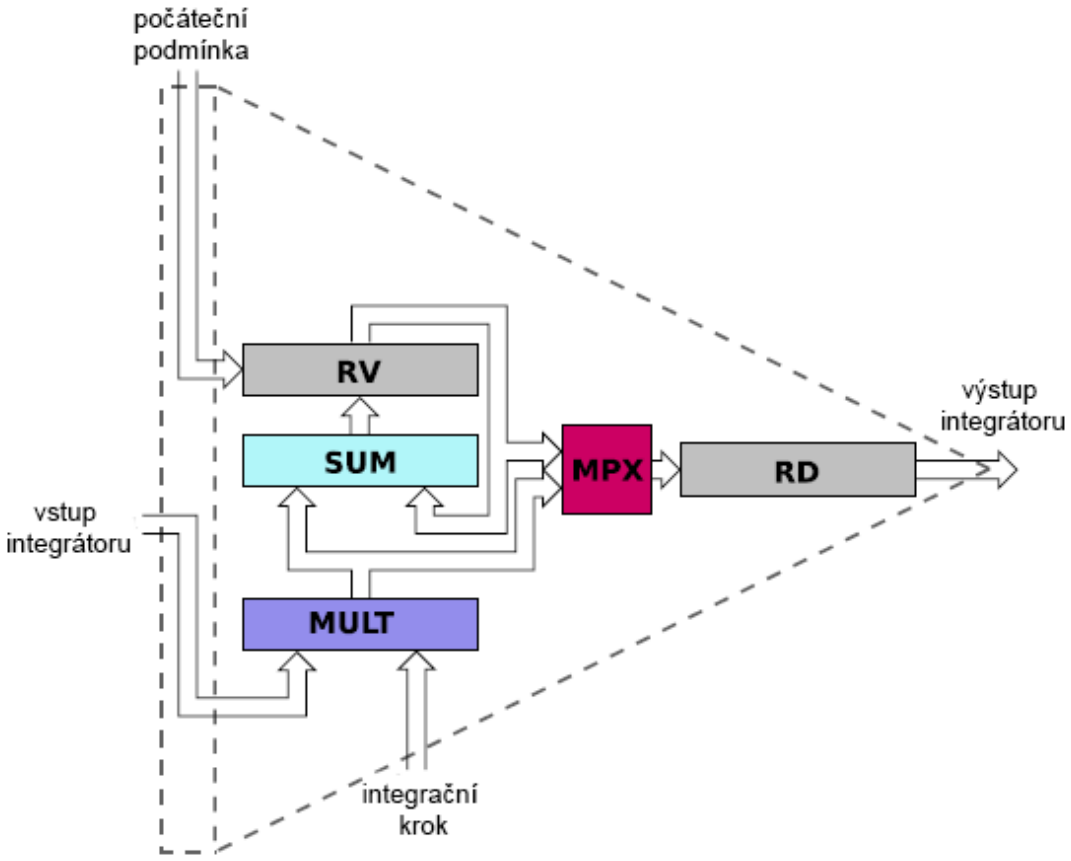
- Paralelně-paralelní integrátor (PPI)
- Sériově-paralelní integrátor (SPI)
- Sériově-sériový integrátor (SSI)

Integrátory se liší podle způsobu výpočtu a komunikace (více [3]). V dalších podkapitolách jsou tyto varianty integrátorů podrobněji rozebrány.

4.1 Paralelně-paralelní integrátor

Jedná se o jednu z jednodušších variant integrátorů. Hlavním principem je paralelní komunikace a výpočty. Na rozdíl od sériově-sériového (SSI) a sériově-paralelního integrátorů (SPI) všechny bity přechází mezi jednotlivými bloky náraz. Stejný princip platí i u násobičky a sčítačky. Operace se provádí najednou se všemi bity jednotlivých čísel.

Níže je ukázka jednoduchého PPI, který jsem převzal z disertační práce pana Krause [3]:



Obr.4.3 Schéma Paralelně-paralelního integrátoru

Jak je vidět, schéma obsahuje 2 registry: registr výsledku (RV) a registr součinu (RD). Dále pak paralelní sčítačku (SUM), paralelní násobičku (MULT) a multiplexor (MPX).

Výpočet začíná tím, že do registrů RD a RV se uloží hodnota y_i a nastaví se počáteční podmínky: integrační krok (h), vstupní hodnota funkce $f(y_i)$. Tyto hodnoty se poté vynásobí pomocí násobičky (MULT), výsledek násobení dvou vstupů se zapíše do registrů RD a současně se přečte k registru RV pomocí sčítačky (SUM). Tímto dosáhneme stavu, že je v RV hodnota Taylorovy řady $DY1$ a v RV je mezisoučet $y_i + DY1$. Cyklus se následně opakuje, ale na vstupu je hodnota $f(DY1)$ a hodnota integračního kroku je $h/2$. Opět proběhne výpočet násobení (výsledkem je hodnota $DY2$) a součet (výsledkem je $y_i + DY1 + DY2$). Hodnoty se uloží do registrů RD a RV a je spuštěná nová iterace algoritmu (s novými vstupními hodnotami $DY2$ a $h/3$).

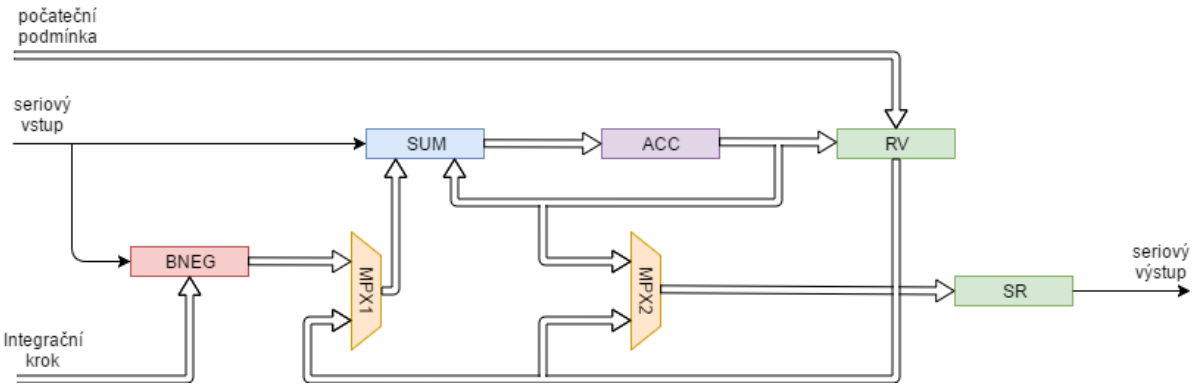
Integrátor pracuje tak dlouho, dokud nedosáhne požadované přesnosti výpočtu. Na konci cyklu je v registru výsledku (RV):

$$RV = y_i + DY1_i + DY2_i + DY3_i + DY4_i + \cdots + DYN_i$$

Což je řešení diferenciální rovnice spočítané pomocí Taylorovy řady.

4.2 Sériově-paralelní integrátor

Tato varianta se liší od předchozí tím, že se násobení provádí sekvenční metodou (po jednom bitu), založenou na Boothově algoritmu násobení. Data vstupují a vystupují sériově. Konečný automat tohoto integrátoru je složitější, protože je potřeba řídit přenos vstupních a výstupních dat po jednom bitu. Další informace o tomto typu integrátorů jsou v [7].



Obr. 4.4: Sériově-paralelní integrátor

Schéma se liší od předchozího tím, že byl přidán blok BNEG, který provádí negaci vstupní hodnoty. Jednoduchý registr RD nahradíme posuvným registrem SR. Navíc datový tok vstupu a výstupu je bitový (vyjádřeno na obrázku jednoduchými šipkami).

Cyklus začíná zápisem vstupní hodnoty y_i do registrů RV a SR. Pak se provede nastavení počáteční hodnoty integračního kroku h. Multiplexor je přepnut na tok od BNEG. Následující kroky algoritmu jsou:

1. Nulování ACC
2. Na vstupu ACC je nejméně významový bit vstupní hodnoty $f(y_i)$
3. Na základě toho bitu se provede jedna ze tří činností:
   3.1. Činitel z registru RN se přečte do ACC
   3.2. Činitel z registru RN se odečte od ACC
   3.3. ACC se vynuluje
4. Výsledek ze SUM se zapíše do ACC
5. Akumulátor a posuvný registr SR se posune o jeden bit doprava

Algoritmus se opakuje do zpracování posledního bitu z $f(y_i)$. Výsledkem je $h * f(y_i)$. Následující operace jsou podobné jako v paralelně-paralelním integrátoru. Operace $h * f(y_i)$ se uloží do registru SR. MPX1 se přepne na tok od RV. Hodnota akumulátoru se sečte s hodnotou uloženou v RV ($y_i + DY1$). Výsledek se uloží do ACC a pak RV.

Potom se celý algoritmus opakuje s nově vypočítanými vstupními hodnotami. Provedení cyklu skončí při dosažení určité přesnosti výsledku.

4.3 Sériově-sériový integrátor

Sériově-sériový integrátor je nejsložitějším a nejpomalejším integrátorem ze všech uvedených typů. Celá komunikace a jednotlivý výpočty (násobení a sčítaní) se provádějí sekvenčně po jednom bitu. Navíc se operace násobení počítá pomocí Boothova algoritmu a operace sčítání.

Navržené schéma sériově-sériového dělícího integrátoru s popisem jeho chování a jednotlivými stavy je probráno podrobněji v následující kapitole.

5 Návrh a implementace SSDI

V této kapitole je popsán návrh a implementace sériově-sériového dělícího integrátoru. Tento integrátor je implementován v jazyce Verilog. Důvodem pro použití toho programovacího jazyka je jeho jednoduchost a podobnost s jazykem C/C++, navíc existuje velké množství odborné literatury a příkladů v ruském jazyce.

Verilog je programovací jazyk pro popis digitálních obvodů, jejich komponent a hardwaru. Byl vytvořen v roce 1984 Philem Morbym a Prahbu Goelem v Automated Integrated Design Systems. Stejně jako VHDL byl původně určen pro modelování digitálních systémů a od roku 1987 se začal používat jako způsob popisu syntetizovaných projektů.

Jazyk Verilog je navržen tak, aby podporoval všechny úrovně abstrakce používané pro návrh těchto obvodů – umožňuje popsat obvod na hradlové i algoritmické úrovni. Je použitelný i pro návrh analogových obvodů. Programovací jazyk Verilog má prostředky pro popis paralelismu, konektivity a explicitního vyjádření času. Verilog umožňuje efektivně provádět popis, simulaci a syntézu digitálních obvodů pomocí vestavěných primitiv (built-in primitives), uživatelsky definovaných primitiv (user-defined primitives), časových kontrol (timing checks), simulovat zpoždění šíření signálů od vstupu do výstupu (pin-to-pin delay simulation), schopností specifikovat externí testovací signály (externí stimul) [9]. Tento jazyk byl zvolen kvůli systému FPGA.

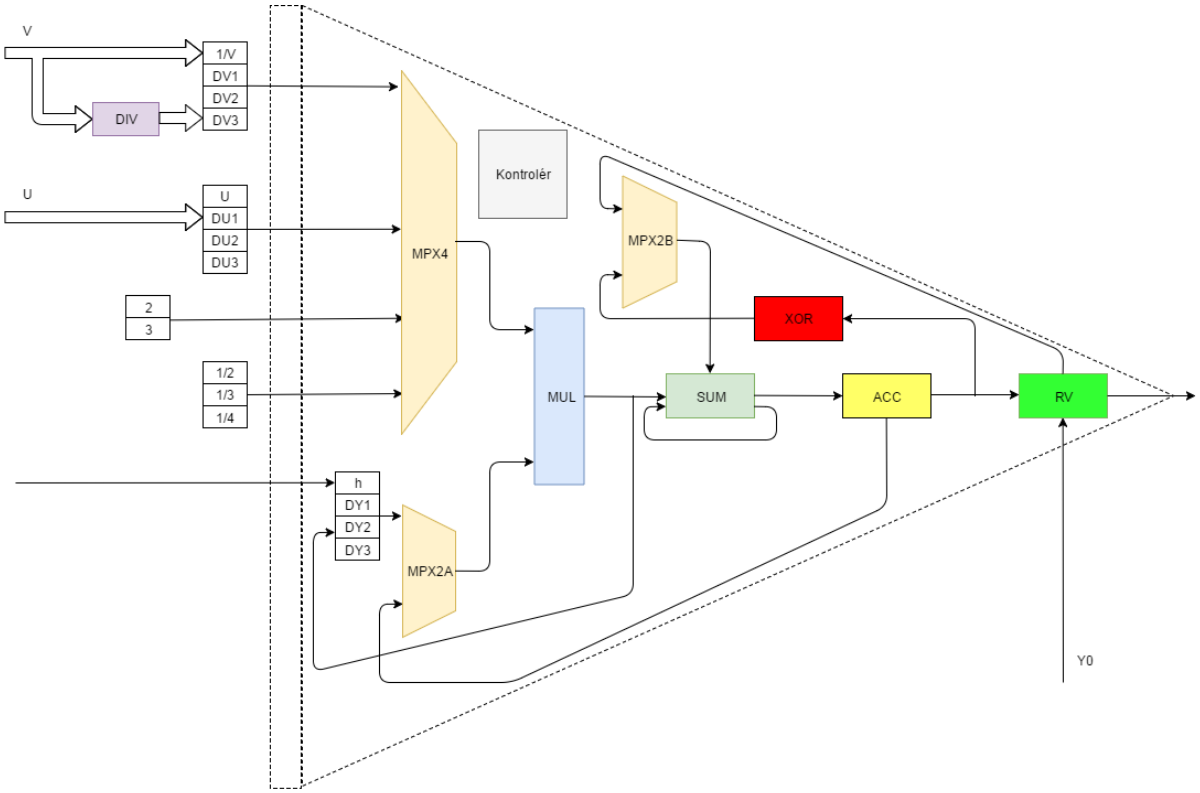
FPGA (Field Programmable Gate Arrays – programovatelné hradlové pole) je elektronická součástka, která se používá pro vytváření dynamických číslicových obvodů.

5.1 Návrh SSDI

V této podkapitole se probírá sériově-sériový dělící integrátor jako na systém navzájem propojených komponent a je znázorněno, jakým způsobem byly jednotlivé komponenty upraveny a rozšířeny, jak jednotlivé komponenty komunikují mezi sebou a jak funguje celý systém.

5.1.1 Popis systému

Základem pro návrh a implementaci sériově-sériového dělícího integrátoru je paralelně-paralelní dělicí integrátor z bakalářské práce [10] a sériově-sériový integrátor z disertační práce pana Krause [3].
Navržený integrátor umožňuje pracovat i se zápornými čísly. Komponenty jako sčítačka a násobička jsou upravené tak, aby pracovaly v doplňkovém kódování. Navíc byl pro převod čísla do doplňkového dvoubitového kódování navržen prvek XOR, který provádí negaci všech bitů vstupní hodnoty a přičte k výsledku 1. Výsledek převodu se pošle na vstup sčítačky, která místo operace sčítaní provede operaci odečítání.
Níže je zobrazeno schéma SSDI systému a popis jednotlivých komponent:



Obr. 5.1: Sériově-sériový dělící integrátor

Význam jednotlivých bloku:

DIV	dělička SRT
MPX4	multiplexor
MPX2A	multiplexor
MPX2B	multiplexor
MUL	násobička
SUM	sčítačka
ACC	akumulátor
XOR	komponenta provádějící negaci vstupního čísla
RV	registr výsledku

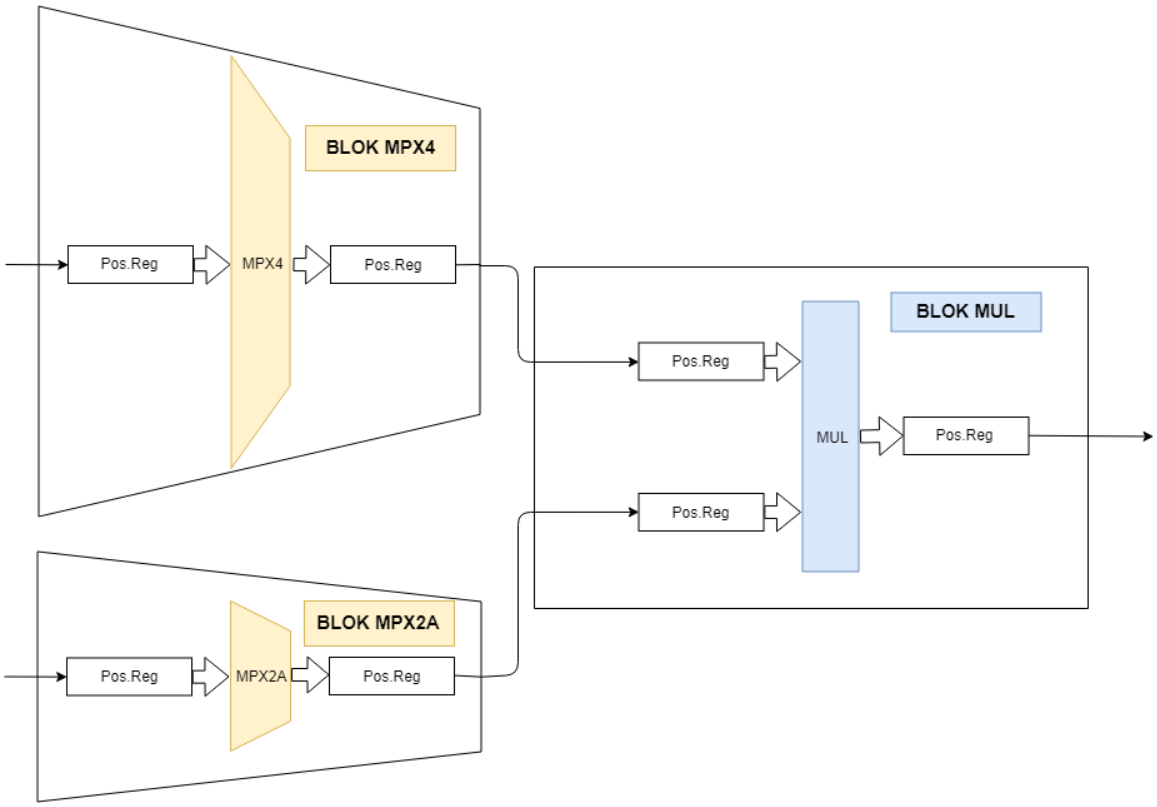
Blok DIV značí SRT děličku, kterou potřebujeme pro výpočet $\frac{1}{v}$. Více informací o této části obvodu si lze dohledat v knize [8]. V další části schématu jsou 3 multiplexory – dva dvouvstupové a jeden čtyřvstupový. Násobička je komponenta provádějící násobení dvou vstupních čísel a pracuje jen s kladnými čísly, která mohou být upravená podle znaménkových bitů na záporné číslo. Vstupní hodnoty jsou přenášeny do násobičky po jednom bitu. Jakmile přenos skončí, komponenta MUL provede násobení. Výsledek se opět přenese sekvenčně po jednom bitu. Z důvodu, že cílem mé práce je navrhnout a implementovat operaci děleni, blok MUL pracuje jako paralelní násobička. Implementace této operace v sériově-sériovém dělícím integrátoru by byla příliš náročná. SUM je jednobitovou sčítačkou, která má 3 jednobitové vstupy. Na dva vstupy přichází jednotlivé bity obou čísel, na poslední vstup je připojen tzv. Carry bit. Komponenta XOR provádí negaci vstupní hodnoty a funguje stejně jak násobička – přijímá a odesílá čísla po jednom bitu, ale samotná operace negace se

provede v jednom taktu. Všechny registry uvedené na schématu jsou posuvné, protože se data posílají po jednom bitu.

Vstupní hodnoty U a V přicházejí na vstup systému paralelně. Na základě těchto hodnot si na začátku provádění program nastaví správné hodnoty registrů 1/V, DV1-DV3, U, DU1-DU3.

5.1.2 Popis rozšíření

Navržený systém je v podstatě PPDI, jehož jednotlivé bloky jsou rozšířeny takovým způsobem, aby pracovaly sériově. Proto bylo nutné ke každé komponentě doplnit posuvný registr, který přenáší čísla od jedné komponenty do druhé po jednom bitu.



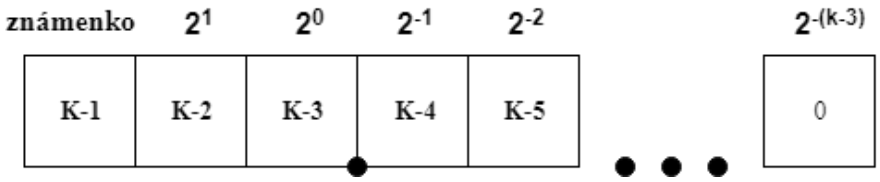
Obr. 5.2: Rozšíření MPX a MUL

Jak je vidět na obrázku 5.2, k blokům MPX4, MPX2A a MUL je přidělena speciální komponenta Pos.Reg (posuvný registr). Když kontrolér pošle řídicí signál "MPX4->MUL, MPX2A->MUL", aktivuje se přechod čísel z posuvných registrů MPX4 a MPX2A do bloku MUL. V každém taktu se přenese jeden bit z posuvného registru MPX4 a jeden bit z posuvného registru MPX2A do bloku MUL. Když budou oba registry prázdné (obě čísla jsou v MUL a připraveny pro násobení), do kontroléru se pošle speciální signál, který oznámí konec operace přenosu a vyvolá příští krok algoritmu – operaci násobení.

Podrobnější popis jednotlivých kroků algoritmu a práce kontroléru je uveden v níže v podkapitole Kontrolér SSDI.

5.1.3 Normalizace čísel

Integrátor popsaný v této práci pracuje s čísly v *pevné řadové čárce*, proto je nutné normalizovat všechna čísla, se kterými tento integrátor pracuje. V programu se používají celá konstantní čísla 1 a 2, která jsou při převodu do binární soustavy $(1)_{10} = (01)_2$ a $(2)_{10} = (10)_2$ a tedy potřebujeme pro jejich reprezentaci ve dvojkové soustavě dva bity. Další (nejvyšší) bit reprezentuje znaménko původního čísla. Schéma 5.3 reprezentuje Fixed point aritmetiku, která byla použita v SSDI.



Obr. 5.3: Fixed point aritmetika

Normalizace čísla je jednou z nutných podmínek pro jeho využití v algoritmu SRT, který byl popsán v podkapitole 3.4. SRT dělička, kterou reprezentuje blok DIV na obrázku 5.1, není součástí implementovaného sériově-sériového dělícího integrátoru. Hodnota registru $1/V$ je předem spočítaná a nastavená.

5.2 Krátký popis chování

Na začátku algoritmu se do registrů zapíší hodnoty DV1, DV2, DV3, které byly vypočítány na základě vzorců 2.6.4. a hodnoty u, DU1, DU2 a DU3, které byly vypočítány podle 2.6.5. Dál se podle SRT algoritmu dělení spočítá hodnota $1/v$. Hodnota iteračního kroku h se zapíše do příslušného registru h. Konstanty 2, 3, 1/2, 1/3, 1/4 se nastaví v konstantních registrech. Y0 se zapíše do posuvného registru RV. Provede se vynulování čítačů a akumulátoru.

Multiplexor MPX4 se přepne na tok dat od registru U, MPX2A je přepnut na tok dat z registru h a MPX2B je přepnut na tok dat od registru RV. Data z U se postupně nahrávají do MPX4 a z h do MPX2A. Pak se data z těchto dvou multiplexorů posílají do násobičky. Pokud budou všechny bity obou čísel přeposlány do násobičky, provede se jejich násobení a výsledek se pošle do akumulátoru.

V následujícím kroku se nastaví MPX4 na tok od 1/V a MPX2A na tok od ACC. Postupně se přepošle obsah registru a akumulátoru na odpovídající multiplexory. Data z multiplexorů jsou přenášena do násobičky a provede se násobení dvou čísel. V podstatě se v tomto kroku provádí operace dělení $\frac{h*u}{v}$ a vypočítá se první člen Taylorovy řady – $DY1$. Výsledek se zapíše do odpovídajícího registru DY1. V příštím kroku se zapíše obsah registru RV do MPX2B. Následně jsou jednotlivé bity dvou čísel poskládány pomocí bloku SUM. A to tak, že jsou poslány jednotlivé bity z registrů RV a násobičky MUL do sčítačky. Na závěr je spočítán jejich součet, který je poslán do akumulátoru a dále do posuvného registru výsledku.

Tímto způsobem spočítáme součet prvního členu Taylorovy řady s počáteční hodnotou Y0. Jinými slovy v RV je $y_1 = y_0 + DY1_i$. Podrobnější popis chování integrátoru je popsán v následující kapitole.

5.3 Kontrolér SSDI

Hlavní a složitější část této práce je návrh a implementace kontroléru. Kontrolér SSDI, kromě jednotlivých výpočtů a komunikace mezi komponentami celého systému, má řídit i přenos každého jednotlivého čísla do dalších bloků po jednotlivých bitech. Jak už bylo uvedeno v podkapitole 5.1.2 a na obrázku 5.2, každý blok je rozšířen o dva posuvné registry – jeden pro příjem a druhý pro odesílání čísel.

Způsob řízení kontrolérem:

1. Ve chvíli, kdy potřebujeme přesunout číslo, kontrolér vyvolá řídicí signál **<název zdrojového bloku>_pop = 1** a **<název cílového bloku>_push = 1**, tím se přenos spustí.
2. Kontrolér čeká na odpověď, která je signalizována jako **<název bloku>_ack = 1**. Výjimkou je signál **MUL_ack**, který signalizuje násobení dvou čísel (blok MUL reaguje signálem **MUL_pop_ack**).
3. Kontrolér ukončí příjem a odesílání tím, že pošle signály **<název zdrojového bloku>_pop = 0** a **<název cílového bloku>_push = 0** do odpovídajících komponent.

Kontrolér obsahuje konečný automat, který řídí celý systém. Přepínání do následujících stavů je řízené tak, že za jednu periodu hodinového signálů CLK sériově-sériový dělící integrátor buď vyvolá nový stav nebo přenos dalšího bitu. Systém umožňuje pracovat pouze s 32 bitovými čísly, což znamená, že pro přenos jednoho (někdy dvou čísel zároveň) je potřeba 34 taktů hodinového signálu. Dva pro řídící signály začátku a konce, dalších 32 taktů pro přenos 32 bitů čísla (někdy čísel).

Koncový automat obsahuje celkem 134 stavů a 31 řídicích signálů po jejichž provedení dostaneme výsledek výpočtu rovnice z 2.7.1 Taylorovou řadou 3. řádu.

Níže je uvedena zjednodušená tabulka stavů a popis jednotlivých operací. Z důvodu přehlednosti algoritmu celého systému jsou v této tabulce operace přenosu a přijetí sjednocené do jediné operace přenosu a některé kroky algoritmu jsou sjednocené do jednoho (třeba nastavení výběrových signálů *sel*).

№	Stavy	Použité signály	Popis funkce
1	S0-S5	MPX4_sel, MPX2A_sel, MPX4_pop, MPX2A_pop, MUL_push, MUL_en	$u * h$
2	S6-S9	MUL_pop, SUM_push, SUM_pop, ACC_push	MUL →SUM, SUM → ACC
3	S10-S11	ACC_pop, MPX2A_push, MPX2A_sel	ACC → MPX2A
4	S12-S19	MPX4_pop, MPX2A_pop, MUL_push, MUL_en, MPX2A_sel, MUL_pop, MPX2A_push	$\frac{1}{v} * u * h$, MUL → DY1
5	S20-S26	MPX2B_sel, RV_pop, MUL_pop, SUM_push, SUM_pop, ACC_push, ACC_pop, RV_push	y_0 + DY1, SUM → ACC, ACC → RV
6	S27-S31	MPX4_sel, MPX2A_sel, MPX4_pop, MPX2A_pop, MUL_push, MUL_en	DV1 * DY1

7	S32-S38	MPX2B_sel, MUL_pop, SUM_push, SUM_pop, ACC_push, ACC_pop, XOR_push	DV1 ∗ DY1 + 0, SUM → ACC, ACC → XOR, operace XOR
8	S39-S43	MPX4_sel, MPX2A_sel, MPX2B_sel, MPX4_pop, MPX2A_pop, MUL_push, MUL_en	DU1 ∗ h
9	S44-S49	XOR_pop, MUL_pop, SUM_push, SUM_pop, ACC_push, ACC_pop, MPX2A_sel, MPX2A_push, MPX4_sel	DU1 ∗ h − DV1 ∗ DY1, SUM → ACC, ACC → MPX2A
10	S50-S55	MPX2A_pop, MPX4_pop, MUL_push, MUL_en, MPX2B_sel, MUL_pop, SUM_push	const $\frac{1}{2}$ ∗ ACC(9) MUL+ 0
11	S56-S59	SUM_pop, ACC_push, MPX2A_sel, ACC_pop, MPX2A_push, MPX4_sel	SUM → ACC, ACC → MPX2A
12	S60-S65	MPX4_pop, MPX2A_pop, MUL_push, MUL_en, MPX2A_sel MUL_pop, MPX2A_push, MPX2B_sel	$\frac{1}{v}$ ∗ ACC(11) MUL → DY2
13	S64-S71	RV_pop, MUL_pop, SUM_push, SUM_pop, ACC_push, ACC_pop, RV_push	y_0 + DY1 + DY2, SUM → ACC, ACC → RV
14	S72-S76	MPX4_sel, MPX2A_sel, MPX4_pop, MPX2A_pop, MUL_push, MUL_en, MPX2B_sel	DV2 ∗ DY1
15	S77-S82	MUL_pop, SUM_push, SUM_pop, ACC_push, ACC_pop, XOR_push, MPX2B_sel, MPX2A_sel, MPX4_sel	DV2 ∗ DY1 + 0, SUM → ACC, ACC →XOR, operace XOR
16	S82-S86	MPX2A_pop, MPX4_pop, MUL_push, MUL_en, MPX2B_sel	DY2 ∗ const2
17	S86-S92	MUL_pop, SUM_push, SUM_pop, ACC_push, MPX2A_sel, ACC_pop, MPX2A_push, MPX4_sel	MUL → SUM, SUM+0, SUM → ACC, ACC →MPX2A
18	S92-S96	MPX2A_pop, MPX4_pop, MUL_push, MUL_en, MPX2B_sel	DV1 ∗ DY2 ∗ const2
19	S96-S102	MUL_pop, XOR_pop, SUM_push, SUM_pop, ACC_push, ACC_pop, XOR_push, MPX4_sel, MPX2A_sel	MUL(22) + XOR(15), SUM → ACC, ACC → XOR operace XOR
20	S103-S106	MPX4_pop, MPX2A_pop, MUL_push, MUL_en, MPX2B_sel	DU2 ∗ h

21	S106-S112	MUL_pop, XOR_pop, SUM_push, SUM_pop, ACC_push, MPX2A_sel, ACC_POP, MPX2A_push, MPX4_sel	MUL(20) + XOR(19), SUM → ACC, ACC → MPX2A
22	S112-S118	MPX2A_pop, MPX4_pop, MUL_push, MUL_en, MPX2B_sel, MUL_pop, SUM_push	const $\frac{1}{3}$ ∗MPX2A(21), MUL → SUM, MUL+0
23	S119-S122	SUM_pop, ACC_push, MPX2A_sel, ACC_pop, MPX2A_push, MPX4_sel	SUM(22) → ACC, ACC → MPX2A
24	S123-S128	MPX4_pop, MPX2A_pop, MUL_push, MUL_en, MPX2A_sel, MUL_pop,MPX2A_push, MPX2B_sel	$\frac{1}{v}$ ∗ ACC (23) MUL → DY3
25	S128-S130	RV_pop, MUL_pop, SUM_push	y_0 + DY1 + DY2 + DY3
26	S131-S134	SUM_pop, ACC_push, ACC_pop, RV_push	SUM → ACC, ACC → RV

Tabulka 5.1 Stavy kontroléru SSDI

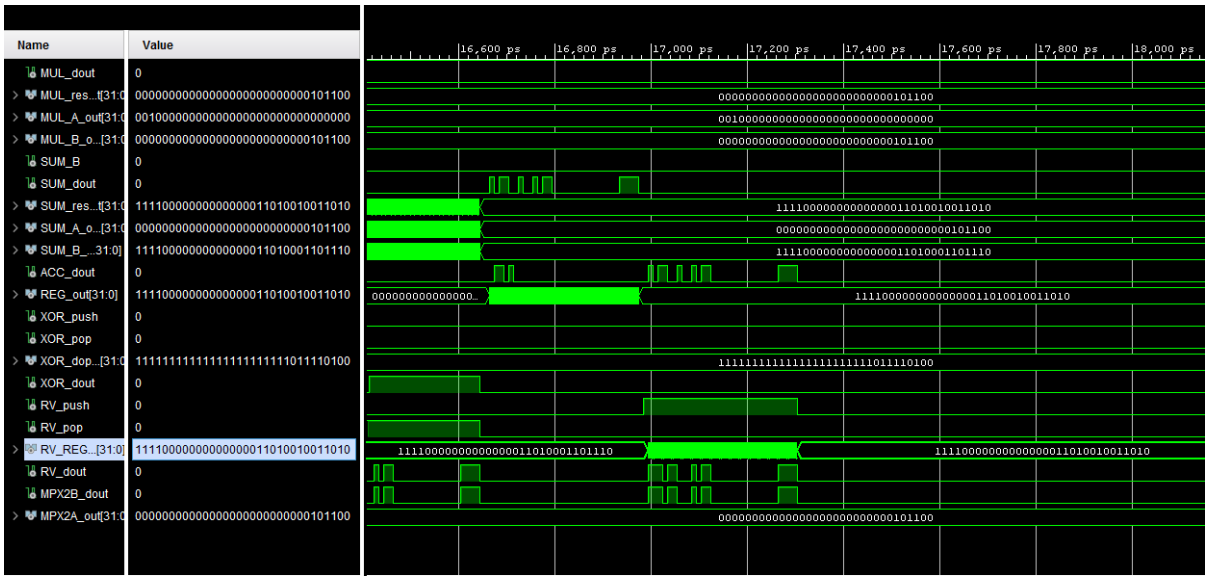
5.4 Simulace

Implementace, testování a simulace sériově-sériového dělícího integrátoru byly provedeny v aplikaci Vivado ve verzi 2018.1. Během simulace bylo zjištěno, že při použití Taylorovy řady 3. řádu jsou výsledky relativně přesné. Na obrázku 5.4 je zobrazen průběh výpočtu rovnice (2.7.1) na 32 bitech.

Výsledky výpočtu na 32 bitech:

$$(11110000000000000110100000101001)_2 = (-0.49995033333\ldots)_{10}$$
$$(11110000000000000011010010011010)_2 = (-0.49994983524\ldots)_{10} \quad (5.4)$$

Pro tuto práci, pod vedením pana Ing. Václava Šátka Ph.D, byla vytvořena aplikace, která počítá rovnice (2.7.1). Tato aplikace byla naprogramována a provedena v interaktivním programovém prostředí MATLAB s využitím skriptovácího jazyka Matlab. Při porovnání výsledku výpočtu v MATLABu $(-0.49995033333\ldots)_{10}$ s výsledkem ze simulace navrženého SSDI je vidět, že se liší až na pátem desetinném místě, což je velmi přesné s ohledem na malý řád výpočtu.



Obr. 5.4.: Simulace v programu Vivado 2018.1

6 Simulátor sériově-sériového dělicího integrátoru

Posledním krokem je vytvoření simulační aplikace, která znázorní algoritmus provádění operace dělení v dříve navrženém SSDI. Pro implementaci simulátoru byl použit nástroj Netbeans IDE a objektově orientovaný programovací jazyk Java. Důvodem zvolení Netbeans je jeho editor a knihovna *Swing*, která umožňuje rychle a snadno vytvářet grafické uživatelské rozhraní a jeho jednotlivé části (okna, dialogy, panely, textová pole, tlačítka a další komponenty). V jazyce Java byla popsána jak funkcionalita jednotlivých komponent, tak i celého systému.

6.1 Popis simulátoru

Po spuštění programu se otevře dialogové okno uvedené na obrázku 6.1 Součástí tohoto okna je zjednodušené schéma sériově-sériového dělícího integrátoru 5.1 z kapitoly 5.

Schéma je upravené tak, že místo sériových komunikačních šipek jsou šipky paralelní (přenos čísel po jednom bitu je příliš dlouhá operace pro znázornění v simulátoru), byly odstraněny bloky DIV a XOR, blok SUM nekomunikuje sám se sebou (operace složení je paralelní, nepotřebujeme kontrolovat přetečení). Navíc nejsou na schématu znázorněny posuvné registry z obrázku 5.2.

Na levé straně okna se nachází čtyři panely s textovými poli, které představují jednotlivé registry a jejich hodnoty. Hodnoty jsou nastaveny takovým způsobem, aby řešily diferenciální rovnici (2.7.1) s integračním krokem 0,1. Počáteční hodnoty mohu být rovněž nastaveny uživatelem, proto jsou hodnoty 1/V, DV1-DV3, U, DU1-DU3, h a RV editovatelné. Pole odpovídající konstantním hodnotám 1/2, 1/3, 2, 3 obsahuje konstantní hodnoty, které byly nastaveny předem a nikdy se nemění. Do registru DY1-DY2, násobičky MUL, sčítačky SUM a akumulátoru ACC se postupně zapisují hodnoty, které budou vypočítány během provedení programu. V registru RV je na začátku hodnota y0, ale na konci provedení programu se tam objeví konečný výsledek.

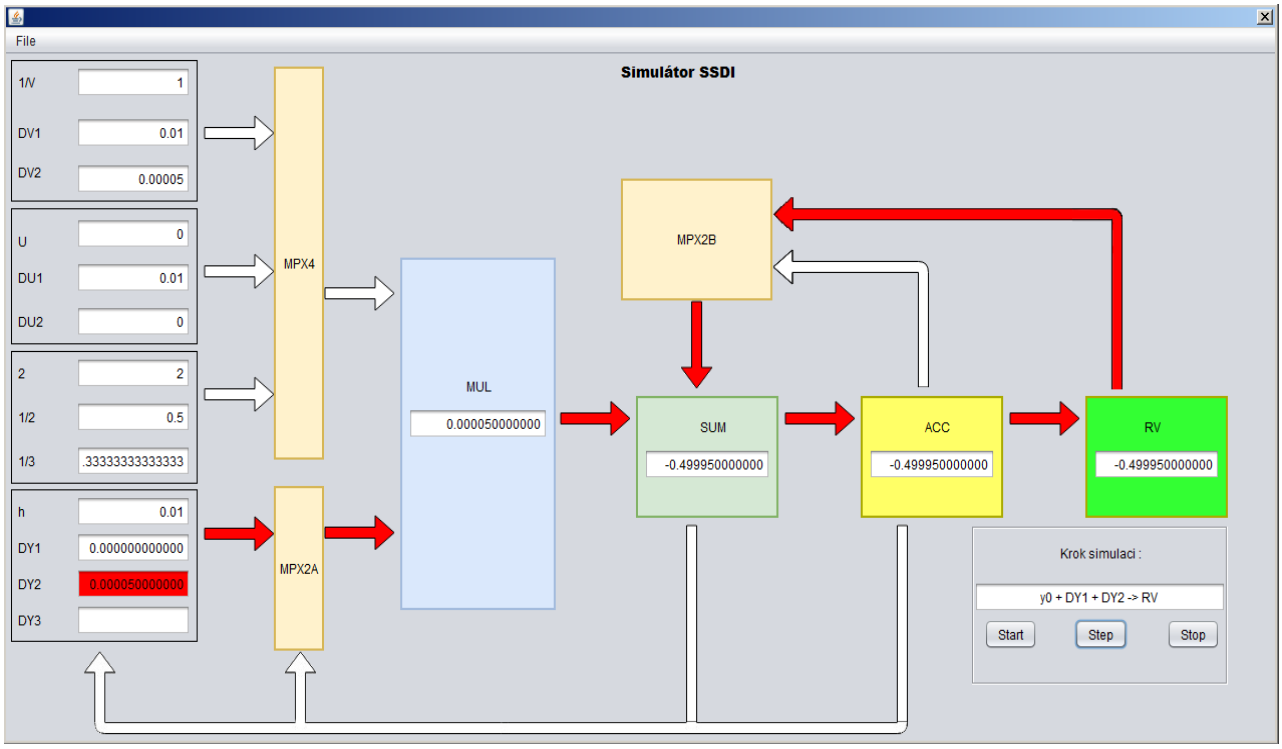
V pravém dolním rohu se nachází ovládací panel. Na něm jsou umístněny tlačítka Start, Step a Stop a textové pole Krok simulace, které informuje o aktuálním stavu simulace. Tlačítko Start spustí simulaci, která se přepíná mezi stavy v časových intervalech 500ms. Tlačítko Step posunuje simulaci do dalšího kroku. Tlačítko Stop simulaci zastaví.

Nahoře se nachází lišta s menu, v níž je položka File. Po otevření File se zobrazí dvě další možnosti – Reset a Exit. Tlačítko Reset nastaví celou simulaci do počátečního stavu, resetuje hodnoty jednotlivých polí a deaktivuje šipky. Tlačítko Exit ukončí provedení aplikace.

6.2 Popis chování

Po stisknutí tlačítka Start nebo Krok, se simulace nastartuje a načte hodnoty jednotlivých registrů. Program pak postupně projde krok po kroku všechny stavy podle tabulky 5.1 se stavem konečného automatu uvedeného v kapitole 5.3. V každém stavu jsou červenou barvou označeny registry a hodnoty, které jsou použity při výpočtu. Červené šipky označují komunikaci mezi jednotlivými bloky. Takové

označení můžete vidět na obrázku 6.1. Vypočítané hodnoty DY1, DY2 se uloží do odpovídacích registrů a v tomto okamžiku změní jejich barvu na zelenou. Po provedení celé simulace se do okna Krok zapíše zpráva “FINISH” a v registru RV se objeví výsledek.



Obr. 6.1: Simulátor SSDI

Závěr

V bakalářské práci byla probrána obyčejná numerická integrace a byly podrobněji rozebrány různé numerické metody: Taylorova řada, Eulerova metoda, metoda Runge-Kutta 2. a 4. řádu. Následně bylo ukázáno, jak lze jednotlivé metody aplikovat na řešení konkrétních rovnic. Dále byla podrobně probrána aplikace Taylorovy řady pro řešení jednoduché rovnice s operací dělení a bylo ukázáno řešení soustavy diterciálních rovnic pomoci blokového schématu.

V práci byla probrána problematika operace dělení v FPGA. Podívali jsme se na různé sekvenční metody – dělení s návratem / bez návratu k nezápornému zbytku a SRT algoritmus. Dělení pomocí SRT algoritmu s radixem 2 bylo znázorněno na příkladu.

Teoretickým základem pro hlavní část práce bylo seznámení se s integrátory SPI a PPI v provedení pro pevnou řádovou čárku. Bylo rozebráno, jak tyto integrátory mohou řešit diferenciální rovnice a následně bylo popsáno, z jakých komponent jsou složeny a jak mezi sebou komunikují.

Po analýze všech uvedených informací byl navržen a implementován sériově-sériový dělící integrátor pro použití v FPGA, který používá algoritmus výpočtu Taylorovy řády 3. řádu. Tento integrátor obsahuje tři multiplexory, násobičku, sčítačku, akumulátor, komponentu XOR a jeden registr RV. Každý prvek z tohoto bloku obsahuje uvnitř 2 posuvné registry, které odpovídají za přenos jednotlivých čísel po jednom bitu mezi komponentami celého systému. Integrátor byl navíc upraven tak, aby bylo možné pracovat s čísly v pevné řadové čárce.

Na závěr byl vytvořen softwarový simulátor, který zobrazuje průběh práce sériově-sériového dělícího integrátoru. Simulátor lze využít pro svou názornost a jednoduchost i pro výukové účely k demonstraci nebo pochopení algoritmu.

V práci by bylo možné pokračovat tak, že by se porovnala časová a paměťová efektivita sériově-sériového dělícího integrátoru s SPDI a PPDI. Další možností je implementace SRT děličky. Zajímavým tématem by bylo rozšířit integrátor o čísla s pohyblivou řádovou čárkou.

Literatura

[1] J. Kunovský : *Modern Taylor series method*, FEI-VUT Brno, 1994, 115 s., Habilitation work.

[2] M. Kubíček, M. Dubcová, D. Janovská: *Numerické metody a algoritmy*. VŠCHT Praha, 2005, ISBN 80-7080-558-7.

[3] M. Kraus: *Paralelní výpočetní architektury založené na numerické integraci*. FIT VUT Brno, 2013, disertační práce.

[4] P. Peringer: Modelování a simulace IMS. FIT VUT v Brně, 2012-12-17, studijní opora.

[5] A. Polyanin, V. Zaitsev: *Handbook of exact solutions for ordinary differential equations*. 2. vyd. Boca Raton, FL: Chapman & Hall/CRC, 2003, ISBN 1-58488-297-2.

[6] L. Sekanina: operace dělení. 2017, slajdy k předmětu INP – Návrh počítačových systémů.

[7] J. Opálka: *Automatické řízené výpočty*. FIT VUT v Brně, Brno, 2014, Bakalářská práce.

[8] Jean Pierre Deschamps, Gustavo D. Sutter, Enrique Canty: *Guide to FPGA Implementation of Arithmetic Functions*. Springer, 2012, ISBN 978-94-007-2986-5.

[9] В. В Соловьев*: Основы языка проектирования цифровой аппаратуры Verilog*. М.: Горячая линия — Телеком, 2014, ISBN 978-5-9912-0353-1.

[10] F. Matečný*: Simulátor procesora s operaciou delenia*. FIT VUT v Brně, 2016, ISBN 978-5-9912-0353-1.

Seznam příloh

Obsah SD

Přiložené SD obsahuje:

- Text bakalářské práce ve formátu PDF.
- Zdrojový soubor práce ve formátu docx.
- Obrázky použité v této práci ve formátu PNG.
- Zdrojové soubory SSDI v jazyce Verilog.
- Zdrojové soubory simulátoru v jazyce Java.
- Spustitelný soubor – Simulator.jav