VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

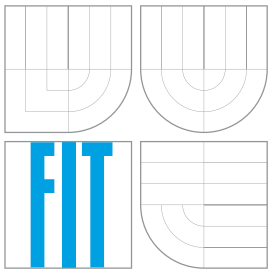
VÝPOČET NUMERICKÉ INTEGRACE NA PLATFORMĚ FITKIT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN MAREK
AUTHOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VÝPOČET NUMERICKÉ INTEGRACE NA PLATFORMĚ FITKIT

NUMERICAL INTEGRATION IN FITKIT PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN MAREK
AUTHOR

VEDOUCÍ PRÁCE — Ing. VÁCLAV ŠÁTEK
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce pojednává o uplatnění numerické metody řešení diferenciálních rovnic pomocí Taylorovy řady. Tento postup byl implementován pomocí specializovaných elementárních numerických procesorů v hradlovém poli FPGA obsaženém ve FITkitu. Bylo tedy nutné vyvinout prostředí, které by dokázalo přijaté hodnoty interpretovat a zobrazit do grafu. Dále bylo pro dobré pochopení postupu výpočtu bylo nutné vyvinout aplikace, které dokázaly prvky zobrazit v podobě, která je vhodná pro neznalého pozorovatele. Je zde také experimentováno s různými úpravami tohoto postupu, aby byla zvýšena rychlost výpočtu.

Abstract

This work discusses the application of numerical methods for solving differential equations using the Taylor series. This procedure was implemented using elementary numerical specialized processors in FPGA contained in FITkit. It was necessary to develop an environment that would be able to interpret the received values and display the graph. It was for a good understanding of the calculation procedure was necessary to develop applications that are able to view the elements in the form appropriate for the uninformed observer. It also experimented with various modifications of this procedure in order to increase the speed of calculation.

Klíčová slova

Taylorova řad, FITkit, elementární procesor, zobrazení prvků, tuhé systémy, RC obvod.

Keywords

Taylor series, FITkit, elementary processor, display element, stiffs systems, RC circuit

Citace

Martin Marek: Výpočet numerické integrace na platformě FITkit, bakalářská práce, Brno, FIT VUT v Brně, 2011

Výpočet numerické integrace na platformě FITkit

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Václava Šátka

.....
Martin Marek
11. května 2011

Poděkování

Tímto bych chtěl poděkovat Ing. Václavu Šátkovi za odborné vedení a také Ing. Michalu Krausovi za konzultace týkající se implementace numerické integrace do FITkitu.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Numerická integrace je v dnešní době využívána především v simulacích reálných systémů (využito například pro řešení elektických obvodů). Jedním z důvodů je jednoduchá algoritmizovatelnost oproti analytickému řešení diferenciálních rovnic. Skutečnost, že numerickou integrací dosáhneme jen jednoho řešení, nás nijak neomezuje, protože obvykle provádíme simulaci pro určité počáteční podmínky a nepotřebujeme tudíž znát obecné řešení rovnic, které definují námi simulovaný systém.

Numerická integrace pro výpočet diferenciálních rovnic je implementována v mnoha aplikacích. Jako příklad můžeme uvést Maple, Matlab a TKSL. V této práci je v mnoha případech využito dat, která jsou výstupem výpočetního systému TKSL.

Numerickou integraci můžeme řešit mnoha metodami. Základní rozčlenění je na jednokrokové a vícekrokové metody. Jednokrokové metody, mezi které patří i metoda s využitím Taylorovy řady, vyčíslují hodnotu vždy jen z aktuálního stavu systému. Z toho důvodu jim pro spuštění výpočtu stačí nastavit jen počáteční podmínky, jako výchozí stav systému.

Oproti tomu metody vícekrokové využívají i historii stavů systému. U těchto metod je tedy nutné začít vypočet pomocí jedné z jednokrokových metod, abychom si vytvořili zmiňovanou historii stavů, ze které může vícekroková metoda vypočítat následující stav systému.

Tato práce se zabývá pedevším jednokrokovou metodou využívající k výpočtu Taylorovu řadu. Tato metoda se vyznačuje svoji velkou přesností díky využití vysokých řádů derivace. Jelikož je výopčet dervivací pro procesory výkonnostně velmi náročné je v systému TKSL i v této práci využito moderní Taylorovy řady [6].

Kapitola 2 ve stručnosti vysvětluje, jakým způsobem se počítají diferenciální rovnice pomoci moderní Taylorovy řady. A také jakým způsobem lze tento výpočet algoritmizovat.

V kapitole 3 jsou prováděny experimenty s tuhým systémem, na kterých bylo zkoumáno, zda je možné výpočet pomocí Taylorovy řady nějakým způsobem urychlit. V takovýchto systémech nastává situace, že jedna funkce je vyčíslena s požadovanou přesností pomocí několika málo prvků Taylorovy řady, ale jiná funkce potřebuje pro vyčíslení s požadovanou přesností několikanásobně více prvků řady. Pokud by byla simulace tohoto systému prováděna na paralelním systému, tak by část, počítající jednoduší funkce, nebyla po jejich vyčíslení využita.

Jelikož Taylorova řada umožňuje vysokou míru paralelního výpočtu, bylo využito přípravku FITkit[1], kteý obsahuje programovatelné pole FPGA, k testům numerických integrátorů. Tyto integrátory jsou jednoúčelové jednotky, které lze vzájemně propojovat a vytvářet tak složitější struktury, reprezentující soustavy rovnic. V tomto užití moderní Taylorovy řady je nejvíce patrný paralelismus výpočtu, kdy každá jednotka počítá jinou proměn-

nou nezávisle na ostatních. Tento fakt je velkou výhodou při využití ve spojitých simulacích.

Při realizaci této práce bylo navrhnuto několik aplikací, pro různé účely. Mezi ně patří například aplikace, která je schopna zobrazovat prvky Taylorovy řady, které byly získány z programu TKSL, na lineární i logaritmické ose.

Na závěr této práce jsou shrnuty poznatky o moderní Taylorově řadě, jakožto prostředku k výpočtu diferenciálních rovnic.

Kapitola 2

Známé postupy řešení diferenciálních rovnic

2.1 Numerické řešení diferenciálních rovnic

V dnešní době je již známo několik numerických metod [3] pro řešení diferenciálních rovnic. Tyto metody se od sebe odlišují například složitostí výpočtu a stabilitou. Jejich nespornou výhodou oproti analytickému řešení je poměrně jednoduchá algoritmizovatelnost a tedy jejich nasazení v univerzálních výpočetních systémech.

2.1.1 Taylorova řada

Taylorova řada, je jednou ze základních jednokrokových metod a lze ji vyjádřit zápisem:

$$y_{n+1} = y_n + hy'_n + \frac{h^2}{2!}y_n^{(2)} + \frac{h^3}{3!}y_n^{(3)} + \cdots + \frac{h^p}{p!}y_n^{(p)} \tag{2.1}$$

Kde h je integrační krok.

Podle dostupné literatury je hlavním problémem použití vyšších derivací. I z tohoto důvodu není Taylorova řada v tomto formátu jednoduše algoritmizovatelná.

2.1.2 Eulerova metoda

Jedná se o jednokrokovou numerickou metodu. Tato metoda má mnoho variant, její základní vzorec je zapsán rovnicí 2.2. V tomto případě se další hodnota vypočítá jen z aktuální hodnoty.

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{2.2}$$

První modifikace 2.3 této metody, nejprve spočítá hodnotu pomocný bod P v bodě $x_n + frach2$. A následně spočítá hodnotu v bodě $x_{[n+1]}$ tak, že vyjde z bodu x_n podle směrnice v bodu P do vzdálenosti kroku h.

$$\begin{aligned} k_1 &= f(x_n, y_n) \\ k_2 &= f(x_n + \frac{h}{2}, y_n + \frac{1}{2}hk_1) \\ y_{n+1} &= y_n + hk_2 \end{aligned} \tag{2.3}$$

Druhá modifikace 2.4 vypočítá také pomocného bodu P, ovšem v bodě x_n+h, a následně vyjde z bodu x_n do bodu x_{n+1} se směrnicí, která je v bodě P, čímž získáme bod P_1. Hodnota funkce y_{n+1} leží ve středu přímky $|PP_1|$.

$$\begin{aligned} k_1 &= f(x_n, y_n) \\ k_2 &= f(x_n + h, y_n + hk_1) \\ y_{n+1} &= y_n + \frac{1}{2}\{k_1 + k_2\} \end{aligned} \tag{2.4}$$

Eulerova metoda v základním tvaru v podstatě reprezentuje první dva prvky Taylorovy řady a tudíž jde s dostatečně malým krokem dosáhnou i touto metodou velmi přesných výsledků.

2.1.3 Metoda Runge-Kutta

Jedná se opět o jednokrokovou metodu, jejíž obecný tvar lze zapsat jako 2.5

$$\begin{aligned} k_1 &= f(x_n, y_n) \\ k_i &= f(x_n + \alpha_i h, y_n + h\sum_{j=1}^{i-1} \beta_{ij} k_j), \quad i = 2, \ldots, s \\ y_{n+1} &= y_n + h(w_1 k_1 + \cdots + w_s k_s) \end{aligned} \tag{2.5}$$

Ovšem v tomto případě je nutné volit konstanty α_i, β_{ij} a w_i tak, aby měla metoda požadovaný řád.

2.1.4 Vícekrokové metody

Nejznámější vícekrokovou metodou je Adams – Bashforth 2.6, která počítá následující hodnotu ze čtyř předchozích. Zde ovšem nastává problém se startem metody. Ze začátku je tedy nutné využít některou z jednokrokových metod pro alespoň tolik prvků, s kolika počítá námi zvolená vícekroková metoda.

$$y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) \tag{2.6}$$

Pro zpřesnění výsledku je možné využít metod typu prediktor/korektor. Mezi tyto metody patří například Adams-Bashforth-Moulton 2.7.

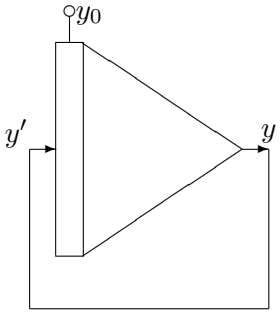
$$y_{n+1} = y_n + \frac{h}{24}(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}) \tag{2.7}$$

2.2 Moderní metoda Taylorovy řady

Tato nová specializovaná metoda Taylorovy řady[6] pro výpočet obyčejných diferenciálních rovnic byla vytvořena na základě analýz různých lineárních rovnic prvního řádu s konstantními koeficienty. Jednou z těchto rovnic je například

$$y' = y, \quad y(0) = y_0 \tag{2.8}$$

Tuto rovnici lze zkreslit i do blokového schématu, zobrazenému na obrázku 2.1. Z tohoto schématu je i patrné, že tuto rovnici lze spočítat pomocí jednoho integrátoru.



Obrázek 2.1: Schéma rovnice 2.8

Takto jednoduchou rovnici je samozřejmě snadné spočítat i pomocí metod jako jsou Runge-Kutta či eulerova metoda. Pokud zvolíme metodu Runge-Kuttay 4. řádu 2.1.3 musíme sestrojit rovnice 2.9.

$$\begin{aligned} k_1 &= h \cdot y_0 \\ k_2 &= h \cdot (y_0 + \frac{k_1}{2}) \\ k_3 &= h \cdot (y_0 + \frac{k_2}{2}) \\ k_4 &= h \cdot (y_0 + \frac{k_4}{2}) \\ y_1 &= y_0 + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \end{aligned} \tag{2.9}$$

Pro Eulerovu metodu 2.1.2 získáme rovnici 2.10, kterou jsme zapsali ve tvaru, který nám už částečně připomíná Taylorovu řadu.

$$\begin{aligned} y_1 &= y_0 + DY1_0 \\ DY1_0 &= h \cdot y_0 \end{aligned} \tag{2.10}$$

Obdobným způsobem jde upravit i rovnici, kterou jsme získali metodou Runge-Kutta 2.9 a to na tvar

$$y_1 = y_0 \cdot (1 + h + \frac{h^2}{2!} + \frac{h^3}{3!} + \frac{h^4}{4!}) \tag{2.11}$$

Z rovnic 2.10 a 2.11 lze jednoduchým způsobem odvodit, že se jedná o charakteristický rozvoj funkce. A to exponenciální řadu $e^t = 1 + \frac{t}{1!} + \frac{t^2}{2!} + \frac{t^3}{3!} + \cdot$. Tento výsledek je shodný s analytickým řešením, které je pro rovnici 2.8 známé

$$y = e^t \tag{2.12}$$

V tomto případě platí

$$y_1 = y_1' = y_2'' = y_3'' = e^t \tag{2.13}$$

zároveň jsou i splněny předpoklady pro platnost Taylorovy řady a to, že funkce $f(t)$a její derivace musejí být jednoznačné, konečné a spojité mezi t a $t+h$. Následně převedeme tento

zápis 2.14 do Taylorovy řady.

$$\begin{aligned} y_1 &= y_0 + h \cdot y_0 + \frac{h^2}{2!} y_0 + \frac{h^3}{3!} y_0 + \dots \\ y_1 &= y_0 \cdot (1 + h + \frac{h^2}{2!} + \frac{h^3}{3!} + \dots) \end{aligned} \tag{2.14}$$

A následně lze provést značení 2.15, ze kterého je viditelná lehká algoritmizovatelnost této řady, což je vhodné pro numerickou integraci zpracovávanou univerzálním nebo speciálním procesorem.

$$\begin{aligned} DY1_0 &= h \cdot y_0 \\ DY2_0 &= \frac{h}{2} DY1_0 \\ DY3_0 &= \frac{h}{3} DY2_0 \\ DY4_0 &= \frac{h}{4} DY3_0 \\ &\vdots \\ DYn_0 &= \frac{h}{n} DY(n-1)_0 \end{aligned} \tag{2.15}$$

Kapitola 3

Modifikace výpočtu pomocí Taylorovy řady

3.1 Úvod do problému

Výpočet numerické integrace pomocí Taylorovy řady je ve většině případů časově velmi náročný, protože námi požadovaná přesnost si vyžaduje výpočet buď s velkým počtem prvků této řady a nebo naopak s extrémně malým integračním krokem. Tyto dva fakty vedly k níže uvedeným experimentům, které se snaží tyto problémy určitými způsoby kompenzovat.

Veškeré následující experimenty budou prováděny na soustavě rovnic 3.1, definující systém, který se řadí mezi tzv. *tuhé systémy*. V těchto systémech rovnici dochází v počátku výpočtu k *přechodnému ději*. Tento děj probíhá přesně do času $\frac{1}{a}$, který je pro nás zajímavý z několika hledisek.

$$\begin{aligned} u' &= -z & u_0 &= 1 \\ z' &= u & z_0 &= 0 \\ y' &= a \cdot z - a \cdot y & y_0 &= 0 \end{aligned} \tag{3.1}$$

Jedním z nich je například fakt, že po čase $\frac{1}{a}$ se funkce y začne shodovat s funkcí $sin(t-\frac{1}{a})$ čehož se dá využít při výpočtu. Z tohoto poznatku je patrné, že zvětšením hodnoty a se začne funkce y přibližovat k funkci z, což je $sin(t)$. V tomto případě lze tedy při dostatečně velké hodnotě a prohlásit funkci y a z za shodné. Z tohoto zjištení lze také odvodit vztah $y_t = z_{(t-\frac{1}{a})}$, který je však platný až od času $\frac{1}{a}$.

Pro kontrolu navrhovaných úprav, bylo spočítáno i analytické řešení soustavy rovnic 3.3. Tohoto řešení bylo využíváno pro porovnávání výsledků v aktuálním kroku, aby byla jednoduše zjistitelná odchylka od tohoto řešení.

$$\begin{aligned} u &= cos(t) \\ z &= sint(t) \\ y &= cos(\alpha) \cdot sin(t-\alpha) \\ \alpha &= arctg\left(\frac{1}{a}\right) \end{aligned} \tag{3.2}$$

3.2 Společné vlastnosti jednotlivých postupů

U všech postupů je nutné zvolit integrační krok, to je hodnota, která udává, v jak velkých časových intervalech budou vyčíslovány hodnoty funkcí, které budeme počítat. Jako další

je nutné zadat maximální řád Taylorovy řady a samozřejmě čas, do kterého se mají funkce počítat. Tyto hodnoty jsou udávány jako maximální, je tedy možné, že hodnoty funkcí s danou přesností se podaří spočítat i s nižším řádem Taylorovy řady. Problém nastává, pokud se výpočet na danou přesnost nezdaří s udaným řádem. V takovém případě je krok metody snížen na poloviční hodnotu a takto upravený krok, bude využit až dokonce výpočtu. Z důvodu snižování kroku na polovinu je možné úpravou vstupních parametrů výpočtu, jako krok a řád metody, dosáhnout lepších výsledků, než kdybychom ponechali určení vhodného kroku na aplikaci samotné.

3.3 Standardní výpočet

Výpočet touto metodou probíhá přesně podle rovnic uvedených v 2.2. V takovém případě jsou hodnoty vyčíslovány v každém kroku na požadovanou přesnost a v případě neúspěchu je půlen integrační krok a výpočet aktuální hodnoty se restartuje. Využívá se tedy rovnic 3.3, ve kterých jsou hodnoty proměnných DYs_{ORD} definovány v 3.4, ale jejich výpočet probíhá podle rovnic 2.15 uvedených dříve.

$$\begin{aligned} y_1 &= DY1_0 + DY1_1 + DY1_2 + \cdots + DY1_{maxORD} \\ y_2 &= DY2_0 + DY2_1 + DY2_2 + \cdots + DY2_{maxORD} \\ &\vdots \\ y_s &= DYs_0 + DYs_1 + DYs_2 + \cdots + DYs_{maxORD} \end{aligned} \tag{3.3}$$

$$DYs_{ord} = y_{s-1}\frac{h^{ord}}{ord!} \tag{3.4}$$

3.4 Metoda nahrazení funkce

Tato metoda využívá faktu, že funkce y ze soustavy rovnic 3.1 je možné považovat od času $\frac{1}{a}$ za funkci z posunutou o časovou konstantu $\frac{1}{a}$. I v tomto případě je však nutné pamatovat na přechodný děj, který je nutné vypočítat standardním způsobem. Výpočet tedy probíhá následovně.

1. Výpočet standardní metodou a to až do času $\frac{1}{a}$, ve kterém končí přechodný děj.
2. Od tohoto času probíhá výpočet pomocí Taylorovy řady pro všechny funkce kromě y.
3. Vyčíslíme tedy hodnoty všech funkcí a zapamatujeme si hodnotu funkce z a čas t_{temp}, ve kterém tato hodnota nastala.
4. Jakmile nastane čas, který je roven času $t_{temp} + \frac{1}{a}$, tak prohlásíme $y(t) = z(t_{temp})$.
5. Pokud však nastane stav $t + h > t_{temp} + \frac{1}{a}$, je aktuální krok upraven tak, aby $t + h_{upraven} = t_{temp} + \frac{1}{a}$, čímž zajistíme správné přiřazení hodnoty. Pokračujeme krokem 2

Z výše uvedeného postupu vyplývá, že touto metodou je možné nastavit maximální krok na $\frac{1}{a}$. Tento fakt je tedy velkým omezením této metody.

I přes udávané omezení je tato metoda násobně rychlejší než výpočet běžnou metodou. Je to způsobenou především kopírováním hodnoty. Tento postup tedy nepotřebuje od konce

přechodného děje tak vysoký řád Taylorovy řady, protože se nepočítá náročná funkce, která je nahrazena funkcí jinou.

V konečné fázi dostáváme výsledky rychleji ovšem ne s takovou přesností. Určité nepřesnosti jsou způsobeny námi udávanou aproximací, protože funkce y není přesnou funkcí z, ale je násobena jinou funkcí způsobující její oscilaci. Pravdou ovšem zůstává, že zde udávaná chyba výsledku se s postupem času nijak nezvětšuje a tudíž je možné celý výpočet provádět s maximální odchylkou, která se projeví již v první periodě funkce.

3.5 Metoda s velkým krokem a nahrazením funkce

Tato metoda vychází z metody předchozí. Její nespornou výhodou však je možnost nastavit krok násobně větší než udávané $\frac{1}{a}$. Tohoto je docíleno následným postupem.

1. Výpočet standardní metodou s krokem vhodným pro výpočet a to až do času $\frac{1}{a}$, ve kterém končí přechodný děj.

2. Od tohoto času probíhá výpočet pomocí Taylorovy řady pro všechny funkce vyjma y s krokem definovaným uživatelem.

3. Vyčíslíme tedy hodnoty všech funkcí a zapamatujeme si hodnotu funkce z.

4. Nyní provedeme výpočet s krokem $\frac{1}{a}$, ve kterém opět vypočítáme hodnoty všech funkcí jako v předchozí metodě a do hodnoty funkce y uložíme hodnotu, kterou jsme si zapamatovali. Pokračujeme krokem 2.

3.6 Metoda s velkým krokem a dopočítáním

V této metodě je využito znalosti času, po který trvá přechodný děj. Po uplynutí tohoto času, se vždy funkce y ustálí na hodnotě, kterou lze označit jako platnou v čase ukončení přechodného děje. Při tomto způsobu výpočtu je možné vyčíslit základní funkce s velkým krokem. Po tohoto kroku však musíme vyčíslovat s krokem násobně menším, se kterým jsme schopni pomocí Taylorovy řady s maximálním řádem , který jsme určili při spuštění výpočtu, vyčíslit správné hodnoty. Všechny výsledky, které spočítáme až do času $t + \frac{1}{a}$ musíme prohlásit za neplatné. A až první výsledek po tomto čase můžeme prohlásit za platný. V tomto způsobu výpočtu dosahujeme velmi vysoké přesnosti. Tento výpočet probíhá následujícím postupem

1. Výpočet standardní metodou s krokem vhodným pro výpočet a to až do času $\frac{1}{a}$, ve kterém končí přechodný děj.

2. Od tohoto času probíhá výpočet pomocí Taylorovy řady pro všechny funkce vyjma y s krokem definovaným uživatelem. A hodnotu funkce y zkopírujeme z předchozího kroku.

3. Nyní provádíme výpočet s krokem, který je vhodný pro výpočet přechodného děje, tento krok je definován jako $\frac{1}{a*10}$ (zjištěno experimentálně). Tento výpočet provádíme až do kroku ve kterém platí $t_{act} + h > t_{lls} + \frac{1}{a}$. Kde t_{lls} je čas, do kterého byl vyčíslen výsledek, kde nebyla vyčíslována hodnota y. Pokračujeme krokem 2.

3.7 Dosažené poznatky

Při provádění experimentů bylo zjištěno možné urychlení výpočtu. Toto urychlení spočívá v rozpoznání typu soustavy rovnic. Pokud se jedná o podobný typ jako byl prezentován zde, je možné uplatnit i uvedené metody. V jiném typu soustavy rovnic je jistě možné najít vlastnoti, díky kterým je možné výpočet urychlit. Případné nasazení metod výpočtu tohoto typu je velmi vhodné v aplikacích, kde je kladen důraz na rychlost výsledku. Je však nutné brát v potaz i skutečnost určité nepřesnosti udávaných výpočtů. Mohou však existovat i aplikace, ve kterých tato nepřesnost není nijak zásadní.

Nejjednodušším urychlením výpočtu, který zde ovšem nebyl prezentová, je vyčíslit každou proměnnou jen tolika prvky Taylorovy řady, které stačí pro požadovanou přesnost a v dalším výpočtu ostatních proměnných v daném čase považovat tyto vypočítané hodnoty za konečné. Tímto je možné ušetřit výpočetní výkon díky nevyčíslování *zbytečných* členů Taylorovy řady, které se nepromítnou do námi zvolené přesnosti. Tento princip je uplatnitelný na všechny typy soustav rovnic, ovšem nedosahuje tak velkého urychlení výpočtu.

Kapitola 4

Programy pro podporu výuky

Pro podporu výuky předmětů ITO a IPR byly vytvořeny jednoduché aplikace a prezentace, na kterých lze názorně předvést jakým způsobem funguje výpočet pomocí Taylorovy řady a jak se mění velikost členu řady v závislosti na zvoleném integračním kroku.

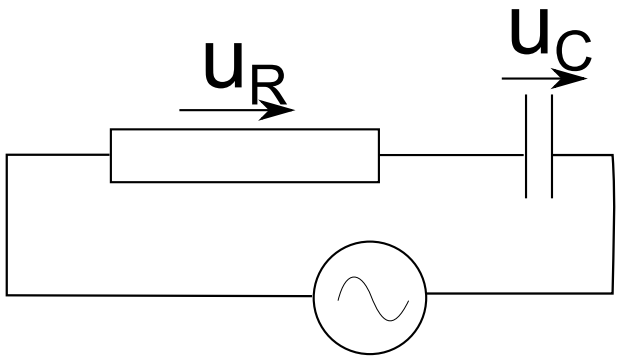
4.1 Zobrazení průběhu napětí v RC článku

Byla vytvořena jednoduchá aplikace, která pomocí grafu zobrazuje průběh napětí na kondenzátoru a rezistoru v obvodu 4.1 se střídavým proudem. Graf je zobrazen po zadání veškerých hodnot definujících uváděný obvod. V tomto grafu je zobrazen fázový posun napětí na kondenzátoru od napětí na rezistoru.

4.1.1 Popis obvodu

Aplikace zobrazuje fázový posun napětí v RC obvodu, ve kterém je rezistor a kondenzátor zapojen v sérii. Tento obvod je napájen střídavým napětím. Napětí na kondenzátoru ve střídavém poli má posunutou fázi vůči napětí na zdroji. Derivaci napětí na kondenzátoru můžeme zapsat jako

$$u'_C = \frac{1}{C} \cdot i_C \quad \Rightarrow \quad u'_C = \frac{1}{C} \cdot \frac{u_R - u_C}{R} \tag{4.1}$$



Obrázek 4.1: RC obvod

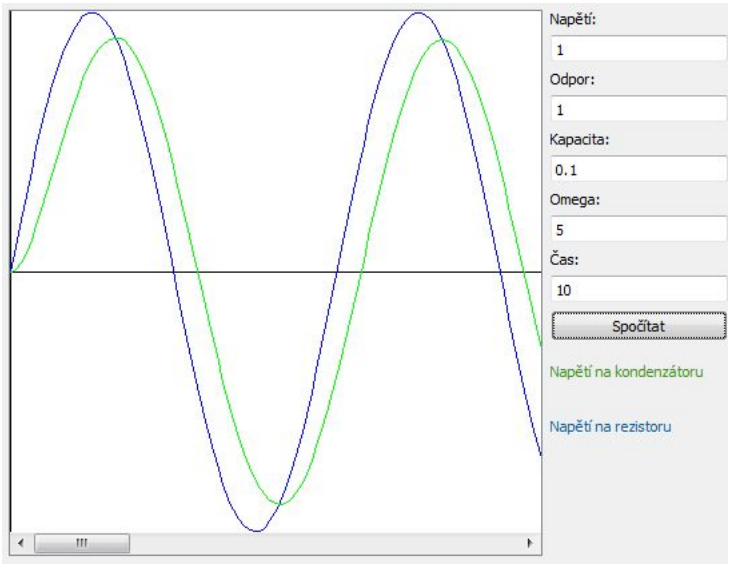
Analytické řešení rovnice je

$$\begin{aligned} uC &= K_1 \cdot e^{\frac{t}{\tau}} + K_4 \cdot sin(\omega \cdot t + \phi) \\ \tau &= R \cdot C \\ K_1 &= \tau * \omega \\ K_2 &= \frac{uR}{\tau^2 \cdot \omega + 1} \\ K_3 &= K_1 \\ K_4 &= \frac{K_2}{cos(\phi)} \\ \phi &= arctan\left(-\frac{K_3}{K_2}\right) \end{aligned}$$

Kde je počáteční podmínka $uC(0) = 0$.

4.1.2 Popis programu

Program je koncipován jako jediné okno bez jakékoliv nabídky. Důvodů k tomuto rozhodnutí je několik, mezi nimi je například nepotřebnost jakýchkoliv nabídek a maximální jednoduchost ovládání. Při spuštění není zobrazen žádný graf, protože aplikace nemá zadány veškeré parametry obvodu. Uživatel tedy musí zadat veškeré vyžadované parametry obvodu, dle kterých dokáže aplikace spočítat dané vlastnosti. Jakákoliv chybová hlášení jsou zobrazována pod textboxy pro zadávání hodnot.



Obrázek 4.2: Náhled programu pro RC článek

V zobrazovaném grafu je zobrazeno napětí na rezistoru a napětí na kondenzátoru. Z daného grafu je patrné, že napětí na kondenzátoru je vždy opožděno o hodnotu ϕ za napětím na rezistoru.

Do grafu jsou zanášena jen napětí na rezistoru a na kondenzátoru, protože tato aplikace byla vyvinuta za účelem zobrazení fázového posunu mezi nepětím na kondenzátoru a rezistoru v RC článku.

4.1.3 Závěr

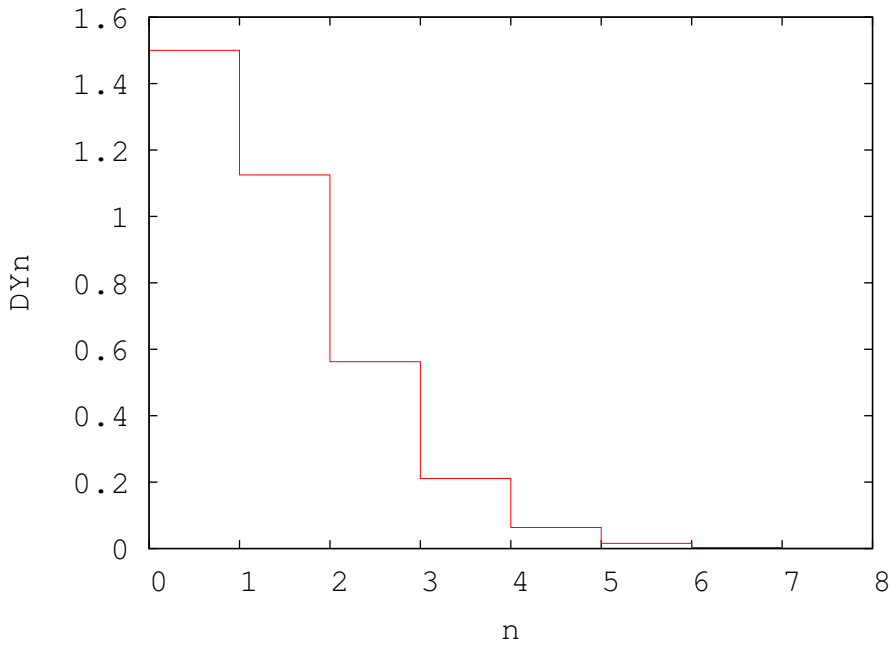
Ze získaných dat je možné vysledovat určité závislosti mezi hodnotami určujícími parametry obvodu. Jako příklad můžeme uvést, že s klesající kapacitou v obvodu je fázový posun a rozdíl amplitud nižší. Fázový posun se také snižuje snížením frekvence a snížením odporu.

4.2 Prezentace prvků Taylorovy řady ve 3D

Pro podporu výuky v ITO a IPR byly vytvořeny také dvě prezentace, zobrazující průběh výpočtu pomocí Taylorovy řady. V těchto prezentacích je využito funkcí Microsoft Office PowerPoint 2007.

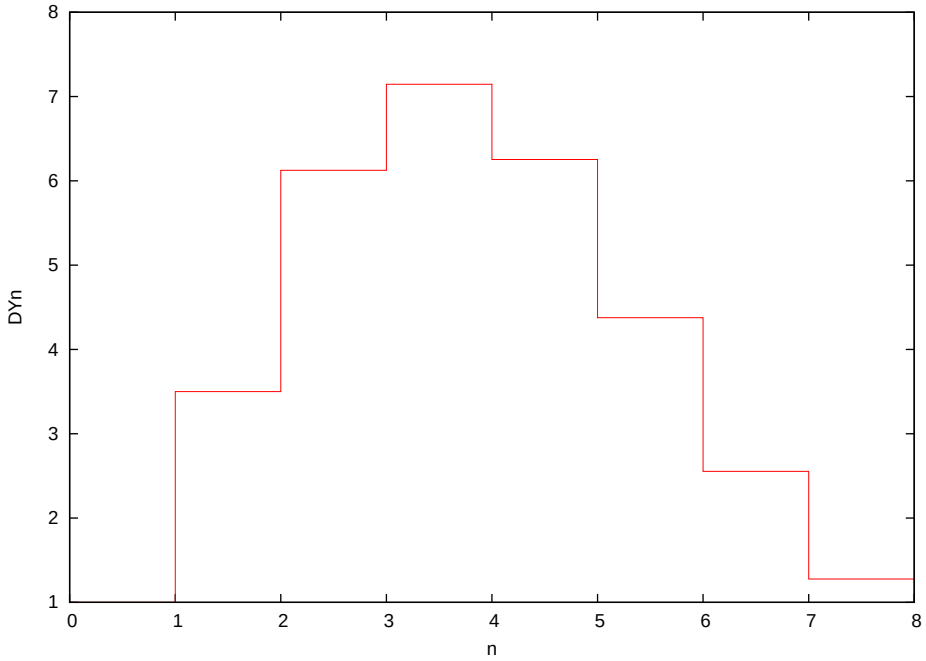
4.2.1 Základní poznatky

Výpočet diferenciálních rovnic pomocí Taylorovy řady je využíváno v nejednom programu. Tento postup využívají i programy TKLS_386 a TKSL_C, se kterými jsem pracoval nejen při vypracování této práce. Ve verzi TKSL_C je možné si nechat u každé vypočtené hodnoty vypsat i jednotlivé prvky Taylorovy řady, které byly pro vyčíslení dané hodnoty v aktuálním čase zapotřebí. Této možnosti jsem využil pro získání dat, které je možné zobrazit v grafech. Tyto grafy se dynamicky proměňují s každým dalším prvkem řady. Pomocí této animace je docíleno větší názornosti výpočtu. Při zobrazování těchto dat jsem se setkal s dvěmi základními variantami Taylorovy řady, které se ve výpočtu objevují.



Obrázek 4.3: Kladné rozložení prvků Taylorovy řady

První z nich je řada s čistě kladnými prvky, zobrazena v grafu 4.3, které postupně konvergují k nule. Na těchto prvcích můžeme pozorovat i vhodnost námi zvolených parametrů Taylorovy řady, a to především na rychlosti konvergence. Obecně lze říci, že zvolením příliš



Obrázek 4.4: Kladné rozložení prvků Taylorovy řady s nevhodně zvoleným integračním krokem

velkého integračního kroku klesá rychlost konvergence a v některých případech může nastat situace, že hodnota jednotlivých prvků začne z počátku výpočtu růst (graf 4.4) a až po několikátém členu se začíná snižovat.

Druhou variantou je řada, ve které se střídají kladné a záporné prvky. Tato řada je zobrazena na grafu 4.5. V této řadě lze sledovat obdobné vlastnosti jako v čistě kladné řadě, s rozdílem střídání hodnot. Pokud bychom brali v úvahu jen absolutní hodnoty prvků, dostali bychom dvě řady s totožnými vlastnostmi.

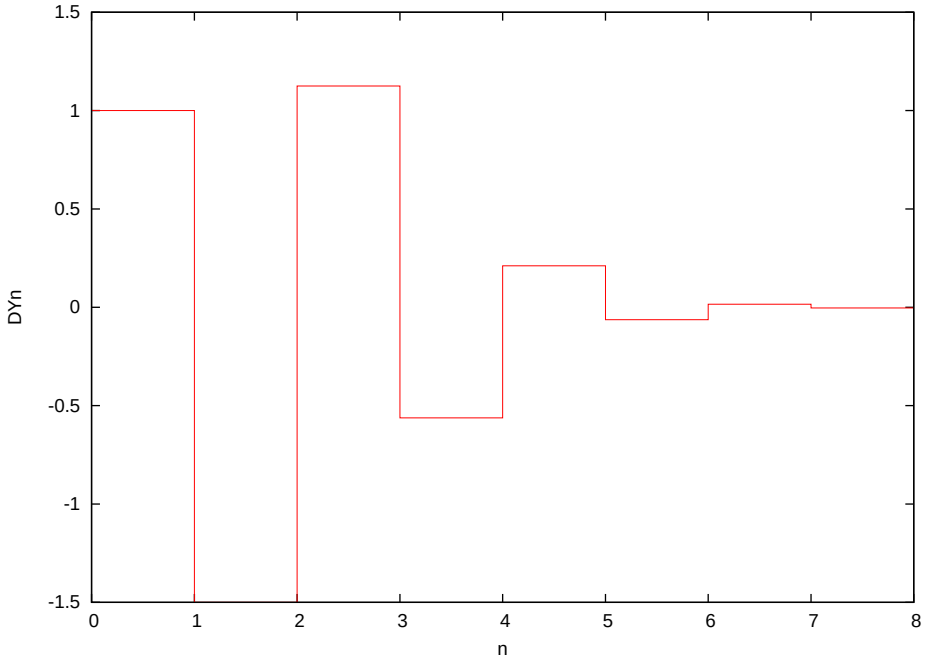
4.2.2 Popis prezentace

Jak již bylo řečeno dříve, je prezentace vytvořena v nástroji MS Office PowerPoint 2007. Každý snímek prezentace je vytvořen pro jiný integrační krok. Mezi snímky jsou zřetelné rozdíly. Z těchto rozdílů lze vysledovat, jak velký vliv má na výpočet zvolený integrační krok.

V každém snímku jsou obsaženy dva grafy. Náhled snímku je na obrázku 4.6. Grafy jsou pomocí časování synchronizovány, aby bylo žřejmé, jak probíhá výpočet.

Graf vlevo znázorňuje hodnotu funkce v daném čase po výpočtu n prvků. Zde je s každým krokem přičítán člen Taylorovy řady, který je aktuálně spočítán. Toto se provádí do té doby, než je absolutní hodnota členu menší než požadovaná přesnost.

Graf v pravo naopak znázorňuje samotné prvky Taylorovy řady. To znamená, že do grafu postupně přibývají prvky, které se zároveň započítávají do grafu vlevo. Tyto prvky se řadí za sebe, aby byla možnost pozorovat, jaký průběh má Taylorova řada s daným nastavením. Na tomto grafu lze nejlépe pozorovat, zda zvolený krok výpočtu funkce byl vhodně zvolen či nikoliv, protože pokud byl zvolen příliš velký krok, tak se to v grafu projeví tak, že ze začátku bude hodnota prvků růst a až po určitém členu začne teprve klesat k nule. Je tedy



Obrázek 4.5: Střídání kladných a záporných prvků Taylorovy řady

potřeba násobně větší množství prvků, než při menším integračním kroku.

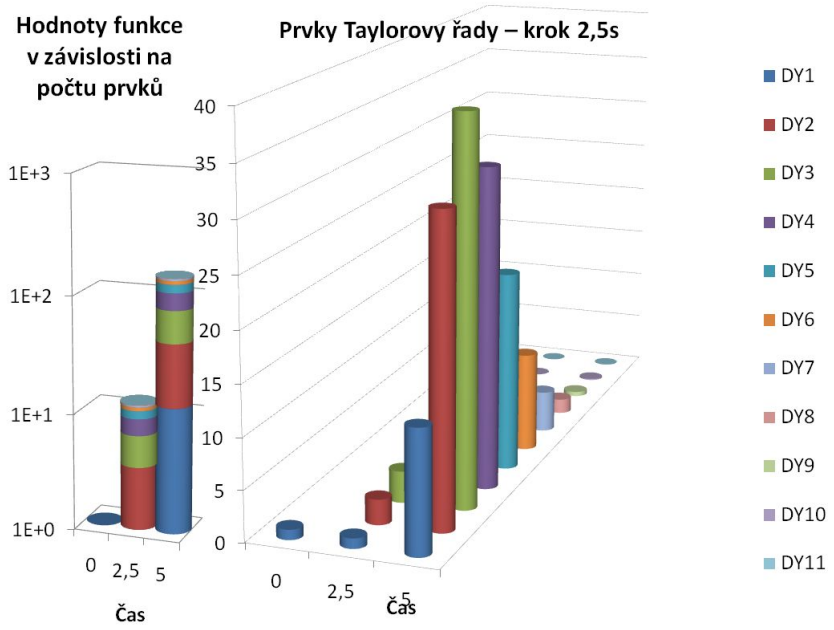
Je však možné, že s delším krokem, který potřebuje více prvků Taylorovy řady, dosáhneme rychlejšího výpočtu. Tento jev může být způsoben požadováním velké přesnosti a tudíž i pro menší krok je potřeba jen o několik méně prvků.

4.3 Zobrazování prvků Taylorovy řady

Pro rychlé zobrazování prvků Taylorovy řady, je třeba aplikace, která je schopna vykreslovat grafy. Takových je dostupných mnoho, ať už placených či volně dostupných. Ovšem potíž nastává, pokud chceme zobrazit na logaritmické ose i prvky se zápornou hodnotou. Standardní nástroje zobrazí jen kladné hodnota a to z důvodu, že logaritmická osa se dělí na segmenty, které jsou značeny jako 10^{-2}, 10^{0}, 10^{2} atd. Přičemž osa x se zobrazí v místě, kde je na ose y hodnota $10^{0} = 1$, a tudíž se hodnoty menší než 1 zdají být záporné. Proto byla vytvořena aplikace, která dokáže zobrazit logaritmickou osu i v záporných hodnotách i když implementovaný návrh přesně nesouhlasí s matematickým pojetím. Názornost takovéhoto grafu je v tomto případě hlavním argumentem proti matematické správnosti grafu.

4.3.1 Základní návrh

Standardní zobrazování grafu na logaritmickou osu je takové, že osa x je zobrazena v hodnotě 1. A hodnoty, které jsou menší než 1, jsou zobrazeny jako záporné. Je to způsobeno rozdělením hodnot do segmentů. Z každé hodnoty je třeba spočítat její umístění. Jako první vypočítáme do kterého segmentu daná hodnota patří a to následujícím způsobem $segment = log_{zaklad}(hodnota)$, kde číslo segmentu určuje celá část čísla $segment$ a hodnota $zaklad$ určuje v jaké soustavě chceme graf zobrazovat.



Obrázek 4.6: Náhled prezentace výpočtu

Z tohoto je patrné, že číslo pro číslo $0,01$ je výpočet segmentu následující $log_{10}(0,01) = -2$ a tento leží v záporných hodnotách, při standardním rozložení grafu s logaritmickou osou.

Pro výpočet pozice v daném segmentu se využívá desetinné části čísla *segment* a to následujícím způsobem $pozice = segment_{desCast} \cdot velikostSegementu$. Tímto rozložíme hodnoty rovnoměrně po celém segmentu, protože desetinná část nabývá hodnot $< 0; 1)$.

Pokud tedy rovnice shrneme, lze je zapsat následovně.

$$\begin{aligned} segment &= log_{zaklad}(hodnota) \\ cisloSegmentu &= segment - (segment\ mod\ zaklad) \\ poziceVSegmentu &= (segment\ mod\ zaklad) \cdot velikostSegementu \\ poziceVGrafu &= cisloSegmentu \cdot velikostSegmentu + poziceVSegmentu \end{aligned}$$

Abychom vyřešili problém se zápornými čísli, je první možností brát v úvahu absolutní hodnoty funkce. Následně u prvků se zápornou hodnotou je třeba jejich spočítanou pozici převrátit podle osy x. Tímto ovšem získáme graf, kde se budou míchat hodnoty kladné a záporné. Tento celek tedy nedává žádný smysl.

Jako další je tedy nutné tyto dva grafy od sebe *oddělit*, ale přitom musí zůstat propojení mezi jednotlivými body. Tohoto docílíme ukládáním hodnot do dvou různých seznamů a ke každé hodnotě si uložíme i její pozici v ose x. Udávaného oddělení dosáhneme posunutím hodnot o hodnotu, kterou spočítáme jako největší zápornou pozici v grafech. Tímto docílíme toho, že osa x bude v bodě, který je nejmenší pro oba grafy. Nyní můžeme ke kladným prvkům přičíst tuto absolutní hodnotu a následně je možné seznam se zápornými prvky převrátit podle osy x a odečíst od nich danou absolutní hodnotu.

Nyní jsou prvky v grafu umístěny tak, jak jsme požadovali a je možné z nich vytvořit graf jednoduchým postupem. Vybíráme z obou seznamů prvek s nejnižší hodnotou x a ten vkládáme postupně do grafu, aby se nám vytvořila spojnice mezi prvky ve správném pořadí.

4.3.2 Vlastní implementace

Implementace je provedena ve framevorku Qt, který obsahuje mnoho tříd, usnadňující nám návrh prostředí i samotnou práci s daty.

Začneme rozdělením prvků, které chceme zobrazit, do dvou samostatných seznamů. Jeden pro kladné a druhý pro záporné hodnoty.

V těchto seznamech máme uloženy body, kterými by procházely grafy kladných a záporných hodnot, pokud bychom brali jejich absolutní hodnotu. Seznamy si také samy udržují maximální zápornou hodnotu, díky čemuž se můžeme následně dotázat na posunutí grafů kladným či záporným směrem. A samozřejmě jsou seřazeny vzestupně podle hodnoty na ose x.

Nyní můžeme začít vkládat body do jednoho společného seznamu. Výběr následující hodnoty je velice jednoduchý. To, ze kterého seznamu bude další prvek, určíme podle hodnoty x v prvním prvku seznamu, tam kde je menší, to je náš aktuální seznam. Pokud je vyžadováno posunutí kladným či záporným směrem je toto posunutí provedeno při vkládání do společného seznamu.

Po sloučení těchto seznamů do jednoho, ve kterém je provedeno i posunutí grafů, můžeme přistoupit k samotnému vykreslování. Pro toto vykreslování je využito třídy, která je zděděná od základní třídy QGraphicsView. V této třídě je doimplementováno zvýrazňování prvků Taylorovy řady a možný posun mezi nimi. Abychom mohli do této třídy vložit graf, musíme jej vytvořit pomocí QPainterPath, do které postupně vkládáme body. Do samotné scény na dané body vkládáme i vlastní grafický prvek, který se nezobrazuje, ale hraje velkou roli při kliku do grafu, kdy se dotazujeme na náš uživatelský typ a v jeho místě vykreslíme svislou čerchovanou linku, která zvýrazní člen Taylorovy řady. Tento také vyšle signál s hodnotami tohoto prvku, které jsou následně zobrazeny v uživatelském rozhraní.

Pokud chceme zobrazit křivky konvergence, tak vytvoříme další dva grafy pomocí QPainterPath, kde každý graf bude reprezentovat jeden z původních seznamů, ve kterých jsou uloženy jen kladné nebo jen záporné hodnoty. Tato funkčnost je vhodná především pro řady, kde se vyskytují jak kladné tak záporné hodnoty.

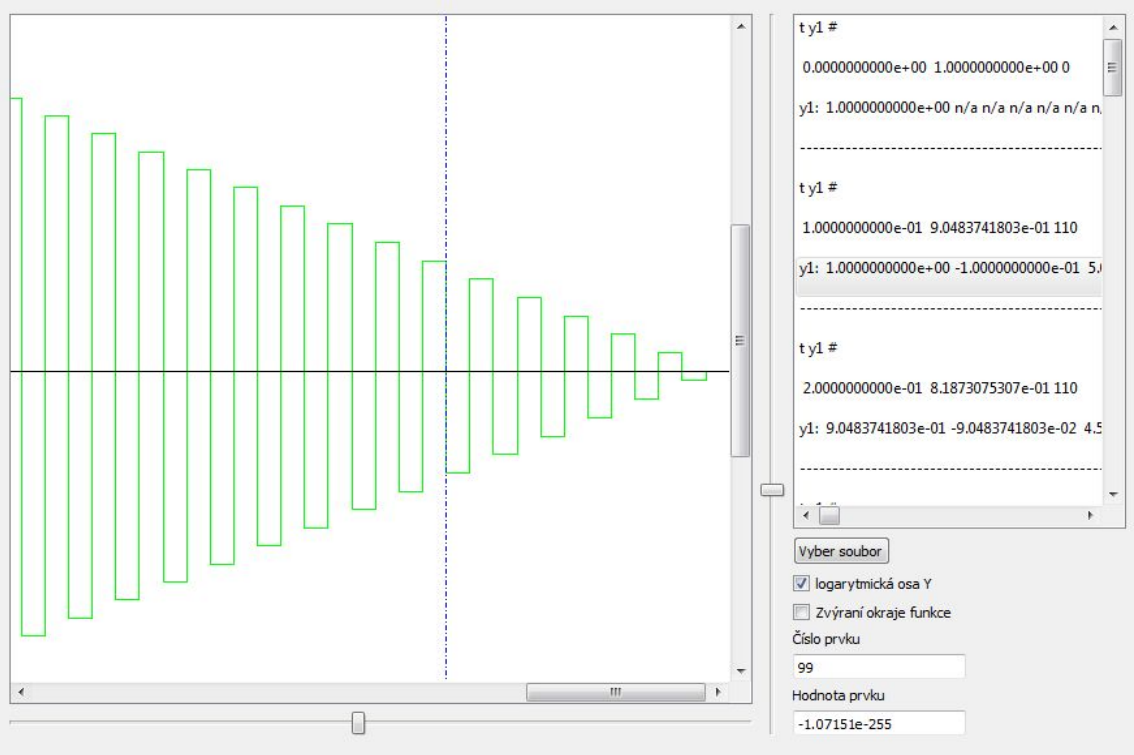
Po vykreslení grafů, spočítáme poměr mezi velikostí grafu a velikostí zobrazovacího widgetu a nastavíme transformační matici tak, aby graf zaplnil co nelépe plochu poskytovanou widgetem. Tuto transformační matici můžeme následně ovlivnit pomocí posuvníků pod a vedle widgetu pro zobrazení grafu.

4.3.3 Ovládání programu

Základní ovládání programu spočívá v načtení souboru s daty pro vykreslení a následné vybrání řádku, který vyžadujeme vykreslit. Tímto máme graf zobrazen v základním rozložení bez logaritmické osy a bez zobrazení funkcí zvýrazňující aproximaci.

Vlevo je widget, do kterého se vykresluje graf. Pod tímto widgetem a vpravo od něj jsou slidery, kterými lze měnit zvětšení v ose x nebo y. Jako vstupní soubor zde slouží textový soubor, který lze získat z programu TKSL_C tak, že si vyžádáme vytištění prvků Taylorovy řady, pro výpočet hodnoty funkce v daném čase. Tento soubor si tedy načteme do programu přes tlačítko *vybrat soubor* a ten se nám rozčlení na jednotlivé řádky do widgetu vpravo. Následně je na nás, abychom si vybrali řádek, který chceme zobrazit v grafu.

Tato koncepce byla vytvořena, aby co nejvíce vyhovovala formátování souboru, který dodržuje TKSL_C. To znamená, že na každám řádku bude informace o celém systému v daném čase a případně, pokud bylo vyžádáno, na řádku dalším vyčíslení všech prvků Taylorovy řady.



Obrázek 4.7: Náhled programu pro vykreslování prvků Taylorovy řady

Většina takto vypočítaných řad dosahuje velmi rychle hodnot nižších než 1, které následně konvergují pomaleji. Je tedy velice vhodné zapnout si zobrazení na logaritmické osy y. Tuto funkci můžeme zapnout zatržením checkboxu vpravo pod widgetem se seznamem řádků.

Jako další funkčnost je implementováno zobrazení funkcí, které ohraničují členy Taylorovy řady. Toto je vhodné především ve velkém zvětšení, kdy již není patrné zda řada konverguje k nule a také v případě, kdy Taylorova řada obsahuje kladné i záporné členy.

Velikost grafu je možné nastavovat v osách x i y, pomocí dvou posuvníků umístěných okolo vykresleného grafu. Toto zvětšení tedy neprobíhá se stejným krokem v obou osách, ale každou je možno nastavit samostatně.

Kapitola 5

Numerická integrace na FITkitu

5.1 FITkit

FITkit[1] je samostatný hardware, který obsahuje výkonný mikrokontrolér s nízkým příkonem, hradlové pole FPGA (anglicky Field Programmable Gate Array) a řadu periferií. Důležitým aspektem je využití pokročilého reprogramovatelného hardwaru na bázi hradlových polí FPGA jenž lze, podobně jako software na počítači, neomezeně modifikovat pro různé účely dle potřeby – uživatel tedy nemusí vytvářet nový hardware pro každou aplikaci znovu.

Při návrhu aplikací se využívá skutečnosti, že vlastnosti hardwaru se v dnešní době převážně popisují vhodným programovacím jazykem (např. VHDL), díky čemuž se návrh softwaru a hardwaru provádí do značné míry obdobně. Student se tedy nemusí hardwaru bát, naopak, může jej s výhodou ve svých aplikacích využít. Generování programovacích dat pro FPGA z popisu v jazyce VHDL probíhá zcela automaticky pomocí profesionálních návrhových systémů.

K tomuto hardwaru je také dostupný scriptovatelný terminál QDevKit. Do této aplikace je možné připojovat naše vlastní pluginy, čehož bylo v této práci využito pro zobrazování dat, přijatých od specializovaného procesoru navrženého pro hradlové pole FPGA ve FITkitu.

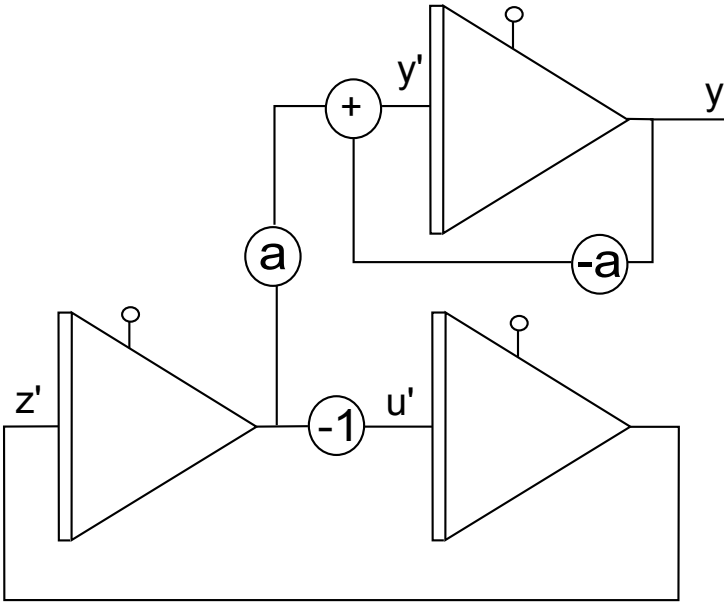
5.2 Motivace k vytvoření specializovaného procesoru

Univerzální procesory běžných počítačů jsou postaveny na architektuře, která je primárně určena pro operace s celými čísly a pro práci s desetinnými čísly se v moderních procesorech

využívá specializovaná jednotka FPU.

Není tedy důvod, aby pro specializované systémy nebyla zkonstruována specializovaná jednotka pro výpočet diferenciálních rovnic, kde by jedna jednotka reprezentovala jednu derivaci funkce a propojením několika takovýchto jednotek by vznikl systém reprezentující danou soustavu rovnic.

Aby byl takto sestavený systém univerzální alespoň v rámci diferenciálních rovnic, je třeba zajistit dynamické propojování jednotek. Toto nám poskytuje hradlové pole FPGA, ve kterém je možné naimplementovat i samotné výpočetní jednotky.



Obrázek 5.1: Schéma systému zapsaného rovnicí 3.1

Pro ukázku můžeme předvést propojení jednotek na rovnicích 3.1. Tato soustava rovnic je shématicky nakreslena jako 5.1.

5.3 Typy výpočetních jednotek

Výpočetní jednotky([5], [4]) pro tuto činnost můžeme rozdělit na několik typů. Především dle vnitřních a vnějších sběrnic. Jejich činnost jako taková však zůstává stále stejná a jedinný rozdíl nastává v provedení daného kroku. Tento rozdíl můžeme pozorovat například na sekvenční a kombinační sčítačce. Kombinační sčítačka má k dispozici veškeré aktuální hodoty bitů a díky tomu může provést součet všech bitů rychleji, toto je však závislé a implementaci sčítačky. Pokud máme sčítačku sekvenční přicházejí bity postupně a je tedy nutné alespoň tolik taktů, kolik mají vstupní čísla bitů.

Jako další velký rozdíl je násobení. Protože pro sekvenční násobení je využito boothova algoritmu, zatímco pro paralelní výpočet je využito kombinační násobičky.

5.3.1 Základní poznatky o výpočetní jednotce

Ve výpočetní jendotce je využíváno dvou matematický operací a to *násobení*, kterého je využito pro výpočet DYn_s, a sčítání, využito pro přičítání jednotlivých prvků ke konečnému

výsledku y_{n+1}.

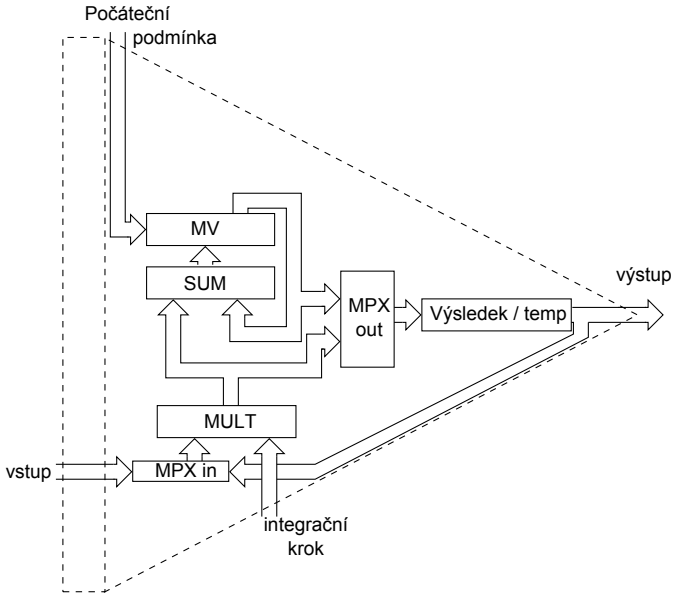
Podle typů komponent použitých pro tyto operace a typu komunikace mezi jednotlivými jednotkami je lze rozdělit na:

- Paralelně – Paralelní
- Sério – Paralelní
- Sério – Sériová

5.3.2 Paralelně – Paralelní

Tato jednotka využívá čistě paralelního přístupu. Vstup, výstup i vnitřní sběrnice jsou tedy čistě paralelní. Využívá se kombinačních prvků (sčítačka, násobička ...), jejichž možné návrhy jsou např. v [2].

Takto vytvořený integrátor zabírá v čipu FPGA nejvíce prostoru, což je způsobeno využitím paralelních sbernic. Je však v 32bitové šířce dat nejrychlejší.



Obrázek 5.2: Paralelně – Paralelní integrátor

Popis jednotlivých prvků:

- **MV:** Mezivýsledek. Zde je uložena aktuální hodnota, která ještě není platná.
- **SUM:** Kombinační sčítačka.
- **MPXout:** Výstupní multiplexor. Řízen bitem určujícím konec výpočtu. Dokud probíhá výpočet je do výstupního registru kopírována hodnota z násobičky.
- **MULT:** Kombinační násobička.
- **MPXin:** Vstupní multiplexor, řízen bitem určujícím konec výpočtu. Dokud probíhá výpočet je do násobičky přiváděn minulý člen DYs_n z registru temp.

- **Výsledek/temp:** Registr, obsahující kompletní výsledek, nebo poslední vyčíslený člen Taylorovy řady. Skutečnost, že se tento výsledek nepromítne v jiném integrátoru je způsobena současným řízením všech integrátorů a tedy nastavením vstupního multiplexoru.

Princip integrátoru:

Do integrátoru 5.2 je nejprve vložena počáteční podmínka do registru **MV**, tato hodnota je následně zkopírována i do výstupního registru. Samotný výpočet jednotlivých hodnot lze popsat jako

1. Vynulování registru **MV**
2. Přes **MPXin** je načtena hodnota vstupu, ze které se bude počítat hodnota následující. Tato hodnota je uložena do registrů **MV** a **temp**. V tomto kroku je hodnota přiváděna skrze vodiče a integrační krok je roven jedné.
3. Přes **MPXin** je načtena hodnota registru **temp**, která je vynásobena hodnotou integračního kroku přivedenou přes příslušnou sběrnici.
4. Výsledek z násobičky **MULT** je přičten k registru **MV** a zároveň zkopírován do registru **temp** pro použití v dalším cyklu.
5. Pokud jsme provedli požadovaný počet iterací, zkopíruj hodnotu **MV** do výstupního registru a pokračuj na 1 jinak 3.

Tato jednotka vyčíslí jeden člen Taylorovy řady za čas roven součtu zpoždění na násobičce, sčítačce a případné zpoždění na sběrnicích.

$$\tau = \tau_{MULT} + \tau_{SUM} + \tau_{sb}$$

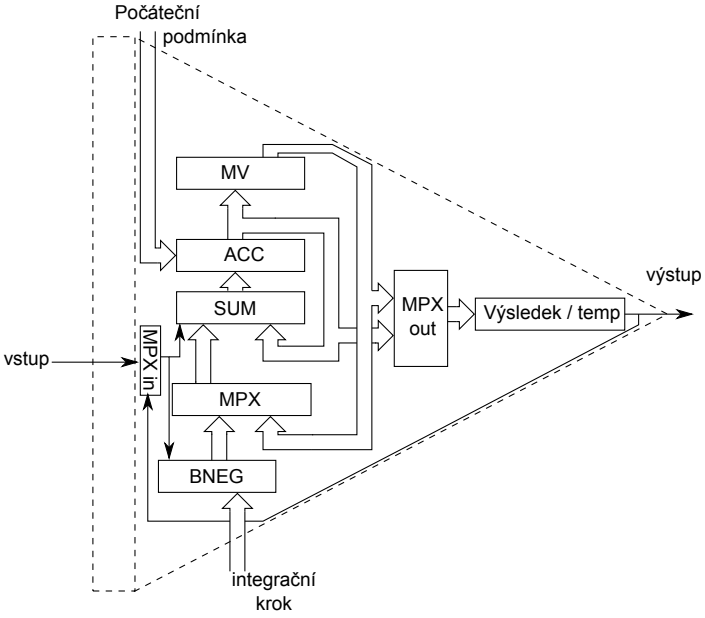
Tento čas je dosažen díky kombinačním prvkům. Je ale vykoupen složitým zapojením, které zabírá velkou plochu na čipu.

5.3.3 Sério – Paralelní

Jednotka 5.3 této koncepce využívá pro vstup a výstup sériových sběrnic, a pro vnitřní propojení je využito sběrnic paralelních. Tento fakt přispívá velkou měrou ke snížení prostoru, který zabírají paralelní propojovací sítě.

Popis jednotlivých prvků:

- **MV:** Mezivýsledek. Zde je uložena aktuální hodnota, která ještě není platná.
- **SUM:** Kombinační sčítačka. Zda se integrační krok přičte či nikoliv, je řízeno dle boothova alogoritmu 5.4.
- **MPXout:** Výstupní multiplexor. Řízen bitem určujícím konec výpočtu. Dokud probíhá výpočet je do výstupního registru kopírována hodnota z **ACC**.
- **MPX:** Multiplexor, který je řízen stavem výpočtu. Dokud probíhá násobení je do sčítačky přiváděna hodnota integračního kroku, jakmile je násobení ukončeno je do sčítačky přivedena hodnota **MV** a tím přičten aktuální člen DYs_n do **MV**.



Obrázek 5.3: Sério – Paralelní integrátor

- **BNEG:**Obvod řízené negace. Tento obvod je řízen logikou, která je popsána v 5.4.
- **MPXin:** Vstupní multiplexor, řízen bitem určujícím konec výpočtu. Dokud probíhá výpočet je do násobičky přiváděn minulý člen DYs_n z registru temp.
- **Výsledek/temp:** Posuvný registr, obsahující kompletní výsledek, nebo poslední vyčíslený člen Taylorovy řady. Skutečnost, že se tento výsledek nepromítne v jiném integrátoru je způsobena současným řízením všech integrátorů a tedy nastavením vstupního multiplexoru.

Princip integrátoru

Do integrátoru 5.3 je nejprve vložena počáteční podmínka do registru **MV**, tato hodnota je následně zkopírována i do výstupního registru. Samotný výpočet jednotlivých hodnot lze popsat jako

1. Vynulování registru **ACC**
2. Přes **MPXin** je načten bit vstupu, určující následující krok výpočtu. Tento krok ovlivní, zda bude k aktuální hodnotě **ACC** přičtena nebo odečtena hodnota integračního kroku, případně nebude přičteno nic. Následně je hodnota v **ACC** posunuta o jednu pozici vpravo. Toto se řídí algorytmem 5.4.
3. Dokud není provedeno celé násobení vrať se k 2. Pokud je násobení kompletní pokračuj.
4. Hodnota v **ACC** je přičtena k registru **MV** a zároveň zkopírována do registru **temp** pro použití v dalším cyklu.
5. Přes **MPXin** je načten bit registru **temp** a provádí se výpočet jako v 2-4 s rozdílem, že přes vstupní multiplexor se přivádí hodnota registru **temp**.

6. Pokud jsme provedli požadovaný počet iterací, zkopíruj hodnotu **MV** do výstupního registru a pokračuj na 1 jinak 5.

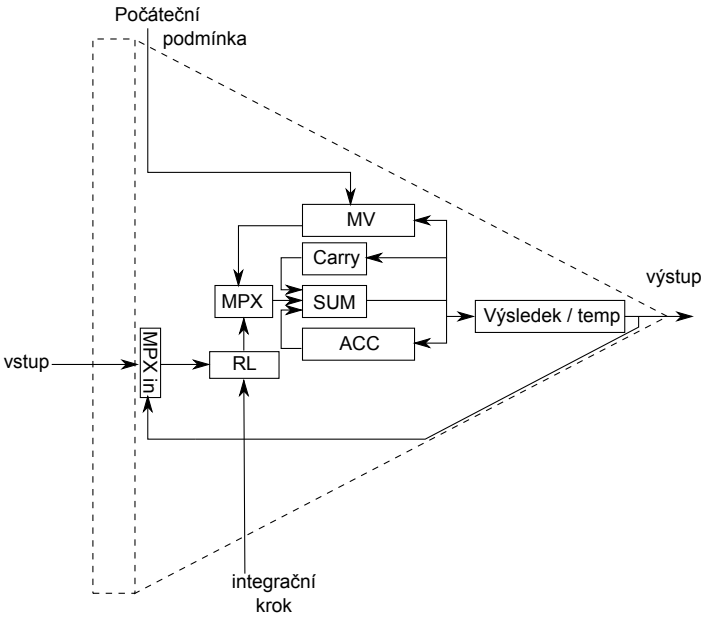
Sério – Paralelní jednotka vyčíslí jeden člen Taylorovy řady za čas roven součtu zpoždění na násobičce, sčítačce a případné zpoždění na sběrnicích vynásobeno bitovou šířkou. A přičtením zpoždění na sčítačce, pro přičtení vyčísleného členu do registru MV.

$$\tau = N \cdot (\tau_{MULT} + \tau_{SUM} + \tau_{sb}) + \tau_{SUM}$$

I když této jednotce trvá vyčíslení jednoho členu Taylorovy řady delší dobu. Lze považovat sériové spojení jednotek jako její hlavní výhodu, což nám ušetří mnoho místa na čipu.

5.3.4 Sério – Sériový

V tomto integrátoru 5.4 je využito čistě sériových sběrnic a sekvenčních prvků. Díky tomu je na čipu zabráno nejméně místa, ale pro čísla s malou bitovu šířkou je tento koncept nejpomalejší. Jeho velká přednost se projeví až při výpočtu s čísly, které mají velkou bitovou šířku. A to především z důvodu, že sběrnice zůstávají stále sériové a zvětšují se jen registry, aby do nich bylo možné uložit číslo o dané bitové šířce.



Obrázek 5.4: Sério – Sériový integrátor

Popis jednotlivých prvků:

- **MV:** Mezivýsledek. Zde je uložena aktuální hodnota, která ještě není platná.
- **SUM:** Sekvenční sčítačka.
- **MPX:** Multiplexor, který je řízen stavem výpočtu. Dokud probíhá násobení je do sčítačky přiváděn bit integračního kroku, jakmile je násobení ukončeno je do sčítačky přivedena hodnota **MV** a tím přičten aktuální člen DYs_n do **MV**.
- **RL:** Obvod implementující rozhodování na základě algoritmu 5.4 a tedy určitou transformaci bitů podle aktuálního vstupního bitu.

- **MPXin:** Vstupní multiplexor, řízen bitem určujícím konec výpočtu. Dokud probíhá výpočet je do násobičky přiváděn minulý člen DYs_n z registru temp.
- **Výsledek/temp:** Posuvný registr, obsahující kompletní výsledek, nebo poslední vyčíslený člen Taylorovy řady. Skutečnost, že se tento výsledek nepromítne v jiném integrátoru je způsobena současným řízením všech integrátorů a tedy nastavením vstupního multiplexoru.
- **Carry:** Klopný obvod uchovávající carry bit.

Princip integrátoru

Do integrátoru 5.4 je nejprve vložena počáteční podmínka do registrů **RV** a **temp**.

1. Vynulování registru **ACC** a klopného obvodu **Carry**.
2. Podle stavu výpočtu je přes **MPXin** přiveden nejméně významný vstupní nebo temp bit do **RL** a také nejnižší bit integračního kroku, který je dle vstupního bitu náležitě upraven.
3. Přes MPX je přiveden do sčítačky, kde je přiveden i nejnižší bit z **ACC** a provede se sečtení, jehož výsledek se vrátí zpět do **ACC** na nejvyšší pozici, přičemž se celý registr posune vpravo. Následně se provede stejná operace s dalším bitem hodnoty integračního kroku a celá akce se opakuje až do doby, než je tato operace provedena pro všechny bity. Po provedení tohoto kroku pokračujeme krokem 2 a to až do doby, než je operace provedena pro všechny bity vstupu.
4. Následně pokud není proveden požadovaný počet kroků, se zkopíruje hodnota **ACC** do registru **temp** a do **MV** se přičte hodnota **ACC** a pokračuje se krokem 2. Pokud je požadovaný počet kroků proveden je následně zkopírována hodnota **MV** do výstupního registru a pokračuje se krokem 1.

Tento integrátor je v zapojení ze všech nejjednodušší a zabírá tedy nejmenší plochu na čipu. Kvůli sériovým komponentům zde nastává rapidně větší zpoždění, protože veškeré výpočty se provádějí sekvenčně. V tomto čase je tedy zahrnut čas, který je nutný pro výpočet pomocí boothova algoritmu, a také čas pro sečtení aktuálního členu DYn_s s hodnotou v daném čase.

5.3.5 Integrační krok

Integrační krok je počítán mimo integrátor, protože je přiváděn do více jednotek současně. Je tedy zbytečné jej počítat v každém integrátoru zvlášť a zabírat tímto obvodem místo na čipu.

5.3.6 Přesnost výpočtu

V jednotce není implementován komparátor, který by určoval, že jednotka už dosáhla požadované přesnosti. Z tohoto důvodu jednotka počítá vždy stejný počet prvků Taylorovy řady, který definujeme mimo jednotku. V současné verzi integrátoru je implementován pevný krok, který je do FPGA vepsán jako konstanta.

Bity násobitele		Provedená operace	
bit b_i	bit b_{i-1}	Boothův kód	Popis operace
0	0	0	Přičtení nuly
0	1	1	Přičtení násobence
1	0	-1	Odečtení násobence
1	1	0	Přičtení nuly

Tabulka 5.1: Tabulka boothova překódování

5.4 Boothův algoritmus násobení

Boothův algoritmus [2] násobení je využíván v sériových verzích násobiček. V tomto případě nám tedy stačí jen určitá rozhodovací logika a sekvenční nebo kombinační sčítačka. V našem případě je tento princip používán z důvodu, že v jeden okamžik neznáme kompletní hodnotu členu DYn_{s-1}, ale jsme schopni z předchozí a aktuální hodnoty bitu určit, zda se bude násobenec přičítat či odečítat od výsledku.

V obvodu je však nutné zavést jeden klopný obvod, například typu D, který bude uchovávat předchozí hodnotu a na začátku výpočtu bude nastaven na hodnotu nula. Po jednom výpočtu je tento bit vždy přepsán aktuální hodnotou na vstupu a poté je možné na vstup přivést hodnotu dalšího bitu.

Jelikož se násobenec přičítá i odečítá a my máme k dispozici pouze sčítačku, je nutné mít k dispozici jeho jak kladou tak zápornou variantu. Toho lze dosáhnout například obvodem řízené negace, který provede negaci a tím, že do úplné sčítačky na vstup carry přivedem hodnotu jedna získáme dvojkový doplněk, což je číslo opačné.

Pokud je v tabulce uvedeno, že se má k aktuálnímu stavu přičíst hodnota nula. Jen se zakáže zápis do ACC a tím se dosáhne přičtení nulové hodnoty.

Námi využívané násobení pomocí boothova algoritmu se označuje jako *Boothovo překódování s radixem 2*, zobrazeno v tabulce 5.1.

5.5 Návrh propojení s PC

Pro hradlové pole FPGA, které je součástí FITkitu, byl vytvořen předpis ve VHDL. Tento předpis reprezentuje numerický integrátor v různých verzích. Tyto integrátory je možné skládat do soustav, které představují diferenciální rovnice. FITkit neobsahuje žádné primární zobrazovací zařízení, a tak jsou výsledky, které vypočítá, dostupná přes rozhranní, ze kterého je tyto hodnoty schopen číst mikroprocesor, který je také obsažen na FITkitu. V první verzi takto získaná data byla zasílána do terminálu, který je zobrazoval čistě textově v hexadecimální podobě.

Jako další bylo tedy nutné vytvořit komunikační protokol, skrze který by bylo možné zadávat počáteční podmínky a počet kroků integrace do pole FPGA a následně získaná data prezentovat v grafu. Možnosti pro toto byli dvě. První by znamenala vytvořit samostatnou aplikaci, která by obstarávala kompletní komunikaci s FITkitem a následně získaná data zobrazovala. Jako druhá byla možnost vytvořit plugin pro skriptovatelný terminál QDevKit [1]. V tomto pluginu už není nutné implementovat propojení s FITkitem, protože toto obstarává samotná aplikace, pro kterou je tento plugi napsán.

Byla zvolena varianta pluginu. A to z výše uvedených důvodů, z nichž hlavní byl fakt, že není nutné implementovat spojení s FITkitem.

V programu pro FITkit byla upravena část, která je určena pro mikrokontrolér. Hlavní změna proběhla v implementaci zasílaných zpráv, aby byl vytvořen jistý protokol, ze kterého lze jednoduše vyčíst data. Jako další úprava tohoto programu bylo autonomní zasílání dalších příkazů do FPGA bez nutnosti čekat na další povel od terminálu. Toho bylo docíleno implementací zprávy v protokolu, který definuje počet integračních kroků.

5.6 Protokol spojení

5.6.1 Příkazy pro FITkit

Příkazy, které jsou zasílány FITkitu jsou v textové podobě a tak je nutné veškeré hodnoty převést na vhodný formát.

- **NUMH [počet]** – Tímto příkazem je do FITkitu zaslána informace o požadovaném počtu integračních kroků.
- **DATAY [y0]** – Zaslání počáteční podmínky do FPGA. Hodnota y0 musí být v 32 bitové reprezentaci s pevnou řádovou čárkou, kde pro desetinnou část je určeno všech 32 bitů. Toto číslo musí být v hexadecimální reprezentaci a zasláno jako string. Pro příklad číslo 0,25 je v tomto formátu zapsáno jako 4000 0000. Jakmile zadám tento příkaz tak FITkit začne okamžitě vyčíslovat hodnoty a zasílat je zpět do počítače.

5.6.2 Data zasílaná FITkitem

Aby bylo možné data z FITkitu reprezentovat je nutné tyto data zaslat zpět do terminálu v podobě, která bude lehce převeditelná na formát, který je srozimetelný počítači. Je tedy využito jiného principu převodu než pro zaslání dat do FITkitu.

- **Vysledek: xxxx** – V tomto formátu zasílá FITkit výsledek výpočtu zpět do terminálu. Je zde využito faktu, že datový typ *unsigned char* má standardně velikost 8 bitů a tudíž je možné v jednou znaku zakódovat dvě hexadecimální číslice. Jde tedy pomocí bitové masky postupně jednoduše vymaskovat jednotlivé bity a k reálné hodnotě v desetinné soustavě vždy přičíst váhu danného bitu.
- **END** – Tato zpráva oznamuje počítači, že již byly vyčísleny všechny požadované hodnoty. Aplikace v tomto okamžiku vykreslí kompletní graf.

5.7 Převod výsledku na formu vhodnou pro zobrazení

Program v FPGA počítá s čísly s pevnou desetinnou čárkou a to jen v rozsahu <0;1). Jedno číslo je dáno 32 bity, kde všech 32 bitů určuje desetinnou část. Takže kupříkladu číslo 8000 0000 hexadecimálně označuje číslo 0,5 decimálně. Ovšem ve FITkitu je číslo, které je zasíláno zpět do PC reprezentováno pouze čtveřicí znaků. A to z důvodu, že hexadecimální číslice zabírá 4 bity, ale jeden znak zabírá 8 bitů. Je tedy zbytečné každou číslici zasílat jako samostatný znak, když můžeme ušetřit 50 % datové šířky. Samotný převod je reprezentován jednoduchým algoritmem 1, který pomocí bitové masky testuje bity a podle toho přičítá

dekadickou hodnotu příslušného bitu k výsledku.

Algoritmus 1: Převod bitové reprezentace na double.

```
Algoritmus:    int index = 1;
for (int i = 0 ; i < NUM_OF_DIGIT ; i++ ) {
    mask = 0x80;
    for (int a = 0 ; a < 8 ; a++) {
        index *= 2;
        data = vstup[i];
        if (data & mask) {
            vystupniHodnota += 1/index;
        }
        mask = mask >>1;
    }
}
```

Z kódu 1 je vidět, že není závislý na určité bitové šířce desetinného čísla, ani na počtu bitů, určených pro desetinnou část. Protože pokud budeme chtít, aby se jako desetinná část z 32 bitů bralo jen 30 bitů, stačí hodnotu *index* nastavit na hodnotu $\frac{1}{4}$ čímž docílíme toho, aby 32. bit měl hodnotu dva a 31. bit hodnotu jedna.

Samotný algoritmus při každém cyklu testuje, zda se z dat vymaskoval bit s hodnotou jedna. Pokud ano, je k výsledku přičtena hodnota vymaskovaného bitu. Po tomto testu je posunut maskovací bit v mace o jednu pozici vpravo, čímž budeme v dalším kroku vymaskovávat bit s hodnotou 2^{a-1}. Hodnota bitu je tedy vždy určena jako $\frac{1}{index^{i \cdot a}}$.

5.8 Zobrazování grafu

Do grafu jsou hodnoty zobrazovány postupně, jak přichází od FITkitu. Jelikož je v FPGA nastaven pevný krok na hodnotu 0.125, který nelze v aktuální době programově měnit a jedinná možnost jak jej upravit je jeho přepis v kódu VHDL a nové naprogramování FITkitu, je stejná hodnota využita i pro zobrazování grafu. Protože FITkit nezasílá čas, pro který byla daná hodnota spočítána, je nutné si ji pamatovat v programu, který vykresluje graf, a s každou další příchozí hodnotou je nutné i tuto vnitřní proměnnou incrementovat o hodnotu kroku. Po získání všech výsledků z FITkitu jsou do grafu dokresleny osy x, y.

Aby bylo možné z grafu jednoduše vyčíst hodnoty v daném bodě aniž bychom musely složitě odměřovat podle stupnice na osách, je implementováno i prohlížení do grafu, které funguje v obou směrech. A to jak ve směru z grafu do seznamu hodnot, tak ve směru opačném. (Implementace zvýraznění bodu je popsána níže.) Při výběru bodu se námi vybraný bod v grafu zvýrazní svislou čerchovanou čarou, se kterou je možné posunovat v grafu z bodu na bod. Tato svislice se nám zobrazí ve dvou případech. První z těchto případů je, když klikneme do blízkého okolí vloženého bodu. Druhý způsob je kliknout na hodnotu v seznamu hodnot bodů, která se nachází vlevo od zobrazeného grafu.

Implementace zvýraznění bodu

Zvraznění bodu je dosaženo vkládáním neviditelných prvků do grafu. Tyto prvky mají vždy nastaveny parametry, které udávají jejich umístění v grafu a také jejich skutečnou hodnotu. Skutečnou hodnotu tento prvek vždy vyšle po aktivaci do seznamu hodnot a v tomto seznamu je následně hodnota zvýrazněna. Aby bylo možné rozpoznat tento prvek je přepsána

virtuální funkce *type*, tak aby vracela mnou definovanou hodnotu, která je rozdílná od všech standardních grafických prvků.

Následného procházení prvků stiskem navigačních kláves je dosaženo, uložením ukazatelů na všechny tyto neviditelné prvky do seznamu, ze kterého je následně vybírána následná nebo předešlá hodnota. Podobně funguje i vybírání ze seznamu hodnot. Jednoduše se z tohoto seznamu vybere prvek, který hodnotami odpovídá hodnotám ze seznamu.

5.9 Chyby výsledku

FPGA pracuje s formátem čísel, který umožňuje uchovat čísla jen v rozsahu <0;1). může tedy nastat případ, že tento rozsah nebude dostatečný. V tomto případě dochází k nestabilitě výpočtu. Tato nestabilita se projevuje například v místě, kde by výsledek měl přesáhnout hodnotu jedna, a to náhlým propadem hodnoty funkce, hodnota funkce po tomto okamžiku je přibližně rovna hodnotě $akt - 1$.

Pokud k tomuto stavu dojde, je třeba provést přepočet počátečních podmínek a celý výpočet spustit znovu s novými hodnotami. Docílíme tím toho, že vypočtená funkce bude n-násobně menší než hodnoty funkce, kterou bychom získaly při výpočtu s originálními vstupy. Jedná se tedy o validní výstupy, které stačí přepočítat pomocí opačné transformace a tím získáme hodnoty shodné jako originální funkce.

U rovnice implementované ve FITkitu se jedná o jednoduché vydělení počátečních podmínek určitým číslem T a po získání hodnot jejich následné vynásobení opět číslem T.

Tato funkčnost mohla být doimplementována i do pluginu pro zobrazení dat. Touto implementací by se však vytratila nezávislost pluginu na rovnici, která je implementována v FPGA. Složitější rovnice mohou vyžadovat také složitější přepočet vstupních hodnota a nelze tedy počáteční podmínky čistě podělit určitou konstantou.

Jako možnost se jevý specializovaný software, který by autonomně obstarával programování FITkitu a podle vložené rovnice by se sám podle určitých pravidel staral o přepočet počátečních podmínek, které by zasílal do FITkitu.

Kapitola 6

Závěr

V této práci bylo ukázáno několik postupů jak lze pracovat s moderní metodou Taylorovy řady. V prvé řadě byly prováděny experimenty s tuhým systémem, který je pro výpočet velice náročný. Bylo však dokázáno, že úpravou výpočtu je možné tento výpočet urychlit. Tato úprava je však platná pouze pro udaný systém a systému jemu podobné. Pokud by byla požadována automatizace úpravy výpočtu, je třeba zavést určité typy systémů, pro které by se dali používat určitá pravidla. Automatické rozpoznání systému by tedy bylo závislé na definici, kterou bychom byli schopni jednotlivé systémy definovat.

Jako další součást této práce byla spolupráce na vývoji numerických integrátorů, které je možné implementovat do FITkitu a to především návrhu rozhraní s počítačem pro zobrazení dat získaných z FITkitu. Jako další rozšíření tohoto softwaru je navržení rozhranní, které by sestavovalo předpis rovnic pro FPGA a tento předpis následně nahrálo do FITkitu. Tímto by bylo možné využívat hradlové pole z FITkitu pro výpočet diferenciálních rovnic bez nutnosti seznámení se s popisovacím jazykem VHDL.

V neposlední řadě bylo v průběhu práce vyvinuto několik aplikací pro podporu výuky v předmětech ITO a IPR. Tyto aplikace prezentují průběh výpočtu pomocí Taylorovy řady a také zobrazení těchto prvků ve formátu vhodném pro studium. Je totiž častým jevem velmi rychlé přiblížení hodnoty k nule. Jelikož však vyžadujeme vysokou přesnost i do řádu 10^{-30} je nutné tyto hodnoty zobrazovat v logaritmické ose, což je problém u řad, které obsahují záporné hodnoty. Byl tedy vyvinut postup, který dokáže zobrazit i takovéto řady.

Literatura

[1] WWW stránky FITkitu. Naposledy navštíveno 28. 3. 2011.
URL http://merlin.fit.vutbr.cz/FITkit/

[2] Drábek, V.: Studijní opora předmětu INP. 2006.

[3] Fajmon, B.; I., R.: *Matematika 3*. Scripta INM.

[4] Hegar, A.: Diplomová práce, FEI, VUT v Brně, 2001.

[5] Kraus, M.: *Elementární procesor specializovaného paralního systému*. Diplomová práce, FIT, VUT v Brně, 2006.

[6] Kunovský, J.: *Modern Taylor Series Method*. Dizertační práce, FEI-VUT Brno, 1994.