VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

SIMULACE CMOS VLSI OBVODŮ

CMOS VLSI CIRCUITS SIMULATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. HILDA ŠŤASTNÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VÁCLAV ŠÁTEK, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Šťastná Hilda, Bc.**
Obor: Matematické metody v informačních technologiích
Téma: **Simulace CMOS VLSI obvodů**
CMOS VLSI Circuits Simulation
Kategorie: Modelování a simulace

Pokyny:

1. Seznamte se se způsoby výpočtu elektrických obvodů ve světových standardech (Dymola, Matlab, Maple, Spice).
2. Seznamte se, pro účely výpočtu elektrických obvodů, s metodami řešení diferenciálních rovnic.
3. Navrhněte model CMOS invertoru (včetně parazitních kapacit) a ověřte správnost funkce modelu. Podobně navrhněte modely CMOS NAND a CMOS NOR. Pro řešení dynamických vlastností CMOS obvodů použijte numerickou integrační metodu s přímým využitím Taylorovy řady.
4. Analyzujte paralelní vlastnosti metody Taylorovy řady při řešení CMOS obvodů.
5. Navrhněte a implementujte simulaci CMOS obvodů ve vhodné architektuře.
6. Simulaci CMOS obvodů ověřte na příkladech a určete možné zrychlení ve srovnání s klasickými metodami.

Literatura:

- Kunovský, J.: Modern Taylor series method. Habilitation work 1994 VUT Brno
- M. Kubíček, M. Dubcová, D. Janovská: Numerické metody a algoritmy. VŠCHT Praha, 2005. ISBN
- Murina, M.: Teorie obvodů, skripta VUT, Brno 2000
- Online circuit simulator with spice, University of Bercley
- H. Elmqvist , M. Otter, and C. Schlegel:Physical Modeling with Modelica and Dymola and Real-Time Simulation with Simulink and Real Time Workshop

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% 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 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Táto diplomová práca pojednáva o spôsoboch výpočtu elektrických obvodov vo svetových štandardoch, ktorými sú v posledných rokoch aplikácie Dymola, MATLAB, Maple či SPICE. Výpočet obvodov súvisí s metódami riešenia lineárnych diferenciálnych rovníc, využitými pri overení správnosti funkcie navrhnutých modelov CMOS invertoru, CMOS NAND, CMOS NOR. Numerická integračná metóda s nasadením Taylorovej rady je vhodnou metódou i pri paralelizácii výpočtov CMOS VLSI obvodov. Simulácia CMOS obvodov s využitím tejto metódy bola implementovaná do aplikácií v jazyku MATLAB, riešiacich obvody, popísane diferenciálnymi rovnicami. Funkčnosť aplikácií bola overená na príkladoch. Signifikantné zrýchlenie výpočtov využitím Taylorovej rady v porovnaní s ostatnými metódami je významným faktorom pri voľbe metód použitých pri simuláciách obvodov.

Abstract

This diploma thesis deals with processes of electrical circuits calculations in the last years' worldwide standards like Dymola, MATLAB, Maple or SPICE applications. Circuits calculations are linked with methods for solving linear differential equations, used in this work also by verification of functionality of designed models for CMOS inverter, CMOS NAND, CMOS NOR. Numerical integration method in combination with Taylor series is a suitable method also for parallel calculations of CMOS VLSI circuits. CMOS circuits simulation was implemented with this method in applications in MATLAB language, solving circuits, represented by differential equations. Functionality of the applications was verified by some real examples. Significant acceleration of calculations using Taylor series compared to other methods is an important factor in choosing methods used in circuit simulations.

Kľúčové slová

CMOS VLSI obvod, CMOS invertor, Matlab, Dymola, Maple, Spice, diferenciálne rovnice, moderná metóda Taylorovho radu, paralelizácia, simulácia elektrického obvodu.

Keywords

MOS VLSI circuit, CMOS inverter, Matlab, Dymola, Maple, Spice, differential equations, modern method of Taylor series, a parallelization, an electric circuit simulation.

Citácia

ŠŤASTNÁ, HILDA. *Simulace CMOS VLSI obvodů.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce ŠÁTEK VÁCLAV.

Simulace CMOS VLSI obvodů

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracovala samostatne pod vedením Ing. Václava Šátka, Ph.D. Ďalšie informácie mi poskytol Ing. Filip Kocina. Uviedla som všetky literárne pramene a publikácie, z ktorých som čerpala.

.....
HILDA ŠŤASTNÁ
24. mája 2017

Poďakovanie

Chcela by som sa poďakovať vedúcemu mojej diplomovej práce Ing. Václavovi Šátkovi, Ph.D. za odbornú pomoc a pripomienky pri jej vypracovaní a taktiež za vrelú podporu mojich najbližších.

Obsah

Kapitola 1

Úvod

V posledných rokoch počítačové simulácie zaplavili svet v hojnej miere ako nesmierne využiteľný nástroj. Mnohé odvetvia, či už je to priemysel alebo výskum, dizajn nových výrobkov, si bez neho už nedokážu predstaviť svoju existenciu. Výhody simulácií skutočného modelu bez prítomnosti jeho fyzickej podoby sú obrovské. Simulácie umožňujú modelovať akékoľvek prostredie, ľubovoľne meniť počiatočné podmienky a hodnoty parametrov, následkom čoho model adekvátne reaguje. Tieto zmeny sú pritom mnohonásobne lacnejšie a časovo menej náročné, než pri použití skutočného modelu a opakovaného manuálneho nastavovania jeho parametrov a zabezpečovania prostredia potrebného pre dôkladné preskúmanie správania sa modelu v rozličných situáciách. Náročnosť zabezpečiť požadované podmienky a prostredie vo fyzickom svete je nezanedbateľná.

Svet moderných technológií pretkaný čipmi, elektronikou, obvodmi si za obdobie poslednej desiatky rokov vynútil vznik množstva simulačných nástrojov spolu s rozsiahlym zoznamom požiadavok na tieto nástroje. Simulátor elektrického obvodu by mal byť schopný vykonať potrebné výpočty v prijateľnom čase, no zároveň by výpočty mali dosahovať určitú presnosť. Balansovanie medzi rýchlosťou vypočtov a ich presnosťou podnietilo rozvoj metód výpočtov elektrických obvodov, algoritmov, optimalizácie či paralelizácie výpočtov, ktorá je obzvlášť významná v rámci stúpajúcich nárokov na výpočtové zdroje.

Na to, aby simulácia prebehla v rozumnom čase, je nutné zvoliť vhodný level abstrakcie danej úlohy. To simulátory zabezpečujú pomocou simulačných algoritmov. Z historického hľadiska sa v nástrojoch implementovali buď analógové alebo len digitálne simulačné algoritmy. V sučasnosti je však bežné implementovanie oboch prístupov, pre zabezpečenie vyššej presnosti výpočtov a tým vyššej kvality výsledkov simulácií [49].

Simulácie CMOS obvodov sú významnou oblasťou, ktorá sa dotýka aj tejto diplomovej práce a prinesie výsledky.

Cieľom tejto práce je preskúmať niektoré moderné simulačné nástroje (*Dymola, MATLAB, Maple, SPICE*), pochopiť ich princípy a významnosť z hľadiska oblasti simulácií a spolu so zameraním sa na problematiku CMOS VLSI obvodov odsimulovať jednoduché obvody obsahujúce vopred navrhnuté modely CMOS invertou, NAND a NOR a následne naznačiť simuláciu zložitejších VLSI obvodov. Návrhy týchto modelov obsahujú miesto klasických tranzistorov rezistory a kapacitory, pričom ich funkcionalita ostáva zachovaná [40].

CMOS obvody zložené z miliónov tranzistorov môžu byť spájané do VLSI systémov (*Very Large Scale Integration*), tvoriacich čipy počítačov, s obrovským využitím.

Existujú možnosti verifikácie modelov, v podobe zabudovaných funkcionalít niektorých nástrojov, ako napríklad nástroje v softvéri *Simulink*. Návrhom verifikácie môže byť použitie tohto alebo obdobných nástrojov. V súčasnosti stúpa množstvo aplikácií a nových riešení,

implementovaných na vhodných architektúrach, preto je práca s dostupnými nástrojmi jednoduchšia, menej časovo a výpočtovo náročná (s väčším počtom jadier).

Paralelné vlastnosti *modernej metódy Taylorovho radu* [41] pri riešení CMOS obvodov sú nezanedbateľné. Analýza týchto vlastností smeruje k lepšiemu využitiu týchto vlastností pri výpočtoch spojených s riešením elektrických obvodov. Metódy s využitím Taylorovho radu majú preto svoje miesto pri obvodoch v mikroprocesoroch, v pamäti a v ďalších oblastiach.

Moja implementácia simulácie CMOS logických obvodov pomocou metódy Taylorovho radu spočíva vo vytvorení jednoduchého programu využívajúceho túto metódu a nástroje MATLAB-u pri riešení lineárnych diferenciálnych rovníc reprezentujúcich daný systém, vykazujúci stabilné riešenie. Preto bola použitá predovšetkým explicitná Taylorova metóda, patriaca do skupiny jednokrokových metód a tiež metód vyššieho rádu.

Kapitola 2

Metódy výpočtu elektrických obvodov

Výpočet elektrického obvodu slúži na predvídanie správania sa navrhovaného zapojenia prvkov, tvoriacich elektrický obvod. Pri správaní sa obvodu nás zaujímajú hodnoty veličín, typických pre každý z prvkov obvodu. Ďalšie veličiny mnohokrát zanedbávame. Jedná sa o parazitné vlastnosti kapacitorov, vlastný odpor jednotlivých komponentov a iné.

Elektrický obvod môžeme skonštruovať a pripojiť do obvodu senzory a merače napätia, prúdu, zaznamenávať výsledky a tým získať hodnoty veličín, potrebných pre výpočet. Vyjadrenie správania sa obvodu pomocou funkcie je však jednoduchšie a úspornejšie než tabuľka nameraných hodnôt. Preto i obvody potrebujú byť popísané pomocou rovníc. Na popis zmeny veličín sa preto do popredia dostávajú diferenciálne rovnice. Metódy riešenia elektrických obvodov pomocou diferenciálnych rovníc tak dostávajú svoj priestor pri simulácií čoraz zložitejších sústav tranzistorov, rezistorov či iných komponentov obvodu.

2.1 Metódy riešenia elektrických obvodov pomocou diferenciálnych rovníc

Diferenciálne rovnice sú silným nástrojom, vyjadrujúcim zmenu veličiny, závislú na mnohých parametroch. Vyjadrenie zapojenia elektrických obvodov (schéma) a zmien hodnôt veličín je uskutočniteľné pomocou diferenciálnych rovníc, preto sa rovnice bežne využívajú ako nástroj pri ich riešení. V súvislosti s diferenciálnymi rovnicami sa v histórii zapísali hlavne mená matematikov ako *Isaac Newton* a *Gottfried Wilhelm Leibniz*.

Pod pojmom diferenciálnej rovnice [39] rozumieme vzťah

$$F(x_1, ..., x_n, y, \frac{\delta y}{\delta x_1}, \frac{\delta y}{\delta x_2}, ..., \frac{\delta^k y}{\delta x_n^k}) = 0, \tag{2.1}$$

kde x_1 až x_n sú nezávislé premenné, y je hľadaná funkcia (n premenných x_1 až x_n), za ktorou v zápise nasledujú jej (prvé až k-te) derivácie podľa nezávislých prememných.

Diferenciálne rovnice s jedinou nezávislou premennou x označujeme pojmom *obyčajné diferenciálne rovnice* (ODR, angl. *ordinary differential equations*, ODEs). Naopak pri prítomnosti viacerých premenných ide o tzv. parciálne diferenciálne rovnice. V tejto práci sa budeme zaoberať hlavne riešením obyčajných diferenciálnych rovníc, kde nezávislú premennú predstavuje čas.

Rád diferenciálnej rovnice vyjadruje informáciu o najvyššej derivácii, prítomnej v rovnici. Diferenciálne rovnice s prvou deriváciou y sa tiež nazývajú diferenciálnymi rovnicami 1. rádu:

$$F(x_1, ..., x_n, y, \frac{\delta y}{\delta x_1}) = 0 \tag{2.2}$$

Obyčajná diferenciálna rovnica (s jednou nezávislou premennou x) 1. rádu preto bude mať tvar

$$F(x, y, y') = 0, \tag{2.3}$$

kde sme použili zjednodušený zápis derivácie y:

$$y' = \frac{\delta y}{\delta x} \tag{2.4}$$

ODR 1. rádu môžeme zapísať i nasledovným spôsobom:

$$y' = f(x, y) \tag{2.5}$$

Funkcia jednej premennej $y(x)$ je riešením takejto rovnice pre všetky hodnoty x, teda:

$$y' = f(x, y(x)) \tag{2.6}$$

Pridaním tzv. počiatočnej podmienky (2.7) riešime *počiatočnú úlohu*, nazývanú i *Cauchyho úloha* [39].

$$y(x_0) = y_0 \tag{2.7}$$

Pre riešenie počiatočných úloh diferenciálnych rovníc vyššieho (k-teho) rádu je potrebné definovať počiatočné podmienky pre 0. až $k-1$. derivácie premennej y, teda napríklad pre diferenciálnu rovnicu 2. rádu by sme definovali dve počiatočné podmienky $y(x_0)$ a $y'(x_0)$.

Pri riešení diferenciálnych rovníc sa môžeme stretnúť i s tzv. okrajovými úlohami, ktorých postup pri riešení je komplikovanejší. Taktiež počet (okrajových) podmienok je vyšší v porovnaní s počiatočnými úlohami, kde špecifikujeme iba počiatočný stav nejakého deja (napríklad zmena napätia na kapacitore v čase).

Rozpoznávame dva základné typy riešenia rovníc - analytické a numerické riešenie. K riešeniu lineárnych obvodov postačuje riešenie *lineárnych diferenciálnych rovníc*, ktoré je na rozdiel od nelineárnych menej náročné na množstvo operácií či čas.

Lineárna obyčajná diferenciálna rovnica 1. rádu $y' = f(x)y$ je rovnica, ktorej riešenie môže byť zapísané v tvare

$$\begin{aligned} y(x) &= CR(x), \\ R(x) &= e^{\int f(x)dx}, \end{aligned} \tag{2.8}$$

kde C je neznáma konštanta a $\int f(x)dx$ je integrál funkcie $f(x)$. Takáto diferenciálna rovnica predstavuje špeciálny prípad rovníc so *separovateľnými premennými* v tvare $y' = f(x)g(y)$, pri ktorých riešenie získame delením $g(y)$ a následným integrovaním oboch strán rovnice:

$$\int \frac{1}{g(y)} dy = \int f(x)dx + C \tag{2.9}$$

2.1.1 Analytické riešenie lineárnych diferenciálnych rovníc

Existuje len niekoľko typov diferenciálnych rovníc, ktoré vieme riešiť určitým postupom, analyticky (lineárne rovnice s konštantnými či nekonštantnými koeficientami, niektoré z nelineárnych rovníc). Analytické riešenie mnohých rovníc je však natoľko náročné, že si s týmto typom riešenia nevystačíme a potrebné sú iné nástroje. V súčasnosti dokážeme numericky riešiť akúkoľvek diferenciálnu rovnicu, dokonca sústavy mnohých takýchto rovníc.

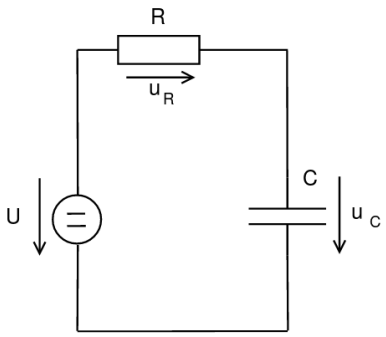
Pri analytickom riešení sa využívajú mnohé postupy, ktorých voľba je závislá na tvare rovnice. Riešenie je možné dosiahnuť napríklad pomocou charakteristickej rovnice, substitúcií, metódy znižovania rádu diferenciálnej rovnice, postupným integrovaním, separovaním premennej a podobne [39].

Nech je daný jednoduchý RC obvod, pozostávajúci z rezistora, kapacitora a jednosmerného zdroja napätia. Uvažujme, že ide o ideálne prvky obvodu, teda zanedbajú sa ďalšie veličiny, charakteristické pre každý komponent obvodu (obrázok 2.1).

Uvažujme vybitý kondenzátor, ktorý sa od času $t = 0$ po pripojení do obvodu začne nabíjať. Táto podmienka môže byť vyjadrená nasledovne:

$$u_c(0) = 0V \tag{2.10}$$

Vzťah (2.10) je tiež označovaný ako počiatočná podmienka.



Obr. 2.1: Schéma jednoduchého RC obvodu z jednosmerným zdrojom napätia (vpravo).

Riešeniu rovníc predchádza ich správne zostavenie [13], ktoré využíva tiež *Ohmov* (2.11) a *Kirchhoffove zákony* [47]. II. Kirchhoffov zákon vyjadruje vzťah algebraického súčtu všetkých napätí v obvode (2.12).

$$u_R = R.i \tag{2.11}$$

$$u - u_R - u_C = 0 \tag{2.12}$$

Do popisu obvodu pomocou rovníc vstupuje diferenciálna rovnica, reprezentujúca zmenu napätia na kapacitore C (2.13):

$$u'_C = \frac{1}{C}.i \tag{2.13}$$

Dosadením rovnice (2.11) do vzťahu (2.12) získame rovnicu (2.14), do ktorej následne dosadíme prúd i, vyjadrený z rovnice (2.13) a dostaneme (2.15).

$$\begin{aligned} u &= u_R + u_C \\ u &= R.i + u_C \end{aligned} \tag{2.14}$$

$$\begin{aligned} i &= u'_C.C \\ u &= R.u'_C.C + u_C \end{aligned} \tag{2.15}$$

Vzťah (2.16) predstavuje časovú konštantu. Ide o vyjadrenie času trvania prechodného deja, počas ktorého sa kapacitor C nabije [47]. Získavame nehomogénnu diferenciálnu rovnicu (pravá strana rovnice nenulová) (2.17) [14].

$$\tau = R.C \tag{2.16}$$

$$u = \tau u'_C + u_C \tag{2.17}$$

Tvar homogénneho riešenia s vyjadrením charakteristickej rovnice dostaneme nasledovným spôsobom (obecné riešenie):

$$\begin{aligned} 0 &= \tau\lambda + 1 \\ \lambda &= -\frac{1}{\tau} \\ u_C &= C(t)e^{\lambda t} = C(t)e^{-\frac{1}{\tau}t} \end{aligned} \tag{2.18}$$

Očakávané riešenie po dopočítaní konštanty derivujeme a dostávame vzťah (2.19), ktorý dosadíme spolu s (2.18) do nehomogénneho riešenia (2.17) a upravíme (2.20):

$$u'_C = C'(t)e^{-\frac{1}{\tau}t} + \lambda C(t)e^{-\frac{1}{\tau}t} \tag{2.19}$$

$$\begin{aligned} u &= \tau(C'(t)e^{-\frac{1}{\tau}t} + \lambda C(t)e^{-\frac{1}{\tau}t}) + C(t)e^{\lambda t} \\ u &= \tau C'(t)e^{-\frac{1}{\tau}t} + \tau\lambda C(t)e^{-\frac{1}{\tau}t} + C(t)e^{\lambda t} \\ u &= \tau C'(t)e^{-\frac{1}{\tau}t} \end{aligned} \tag{2.20}$$

Zo získaného vzťahu (2.20) vyjadríme $C'(t)$ a integrujeme, čím získame vyjadrenie neznámej $C(t)$ (2.21), ktorú následne môžeme dosadiť do homogénneho riešenia, čím rovnica napätia na kapacitore C dostane tvar (2.22), v ktorom jedinou neznámou ostáva konštanta k.

$$\begin{aligned} C'(t) &= \frac{u}{\tau e^{-\frac{1}{\tau}t}} \\ \int C'(t) &= \int \frac{u}{\tau} e^{\frac{1}{\tau}t} \\ C(t) &= Ue^{\frac{1}{\tau}t} + k \end{aligned} \tag{2.21}$$

$$\begin{aligned} u_C &= C(t)e^{\lambda t} = C(t)e^{-\frac{1}{\tau}t} \\ u_C &= (Ue^{\frac{1}{\tau}t} + k)e^{-\frac{1}{\tau}t} \\ u_C &= U + ke^{-\frac{1}{\tau}t} \end{aligned} \tag{2.22}$$

Pri výpočte konštanty k (2.23) prichádza na rad počiatočná podmienka (2.10).

$$\begin{aligned} u_C(0) &= U + ke^{-\frac{1}{\tau}.0} \\ u_C(0) &= U + ke^0 \\ u_C(0) &= U + k \\ k &= u_C(0) - U \end{aligned} \tag{2.23}$$

Konštantu dosadíme do (2.22) a konečné analytické riešenie má tvar (2.24):

$$u_C = U + (u_C(0) - U)e^{-\frac{1}{\tau}t} \tag{2.24}$$

Do výsledného riešenia je následne možné dosadzovať konkrétne hodnoty veličín prvkov zapojených v danom RC elektrickom obvode (obrázok 2.1). Pri zapojení striedavého zdroja napätia postupujeme podobne, riešenie je však o niečo náročnejšie:

$$u_C = \frac{a\omega}{a^2+\omega^2}e^{-at} - \frac{a\omega}{a^2+\omega^2}cos(\omega t) + \frac{a^2}{a^2+\omega^2}sin(\omega t), a = \frac{1}{RC} \tag{2.25}$$

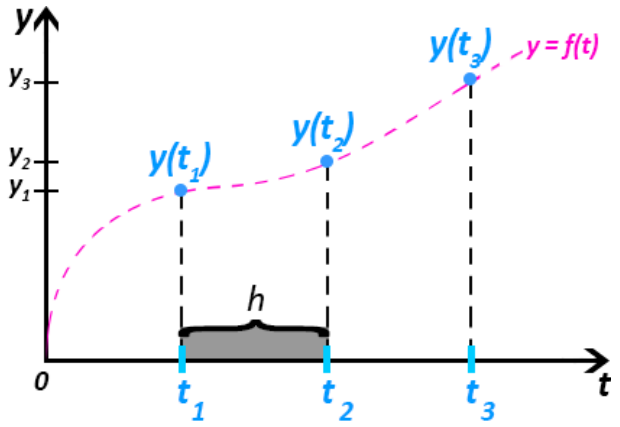
Tieto analytické riešenia budú využité pri simulácii jednoduchého RC obvodu s jednosmerným i striedavým zdrojom napätia.

Analytické riešenie mnohých obvodov nie je triviálne, preto sa v súčasnosti diferenciálne rovnice popisujúce daný obvod riešia numericky. Výkonné počítače zvládajú výpočet numerického riešenia tisícov rovníc reprezentujúcich zložité obvody.

2.1.2 Numerické riešenie lineárnych diferenciálnych rovníc

Numerické riešenie rovníc sa používa hlavne v prípadoch, kedy analytické riešenie nestačí, alebo je jeho získanie veľmi zdĺhavé. Uľahčuje výpočet derivácií, určitých integrálov, napomáha pri rozličných úlohách technického charakteru. Vyznačuje sa o niečo menšou presnosťou voči analytickému riešeniu, no tento rozdiel je mnohokrát zanedbateľný.

Základný princíp numerického riešenia rovníc sa nezakladá na hľadaní neznámej funkcie, ale na postupnom algoritmickom výpočte hodnôt hľadanej funkcie pre jednotlivé kroky volenej metódy výpočtu, a teda výstupom sú diskrétne hodnoty, aproximujúce hľadanú funkciu (obrázok 2.2).



Obr. 2.2: Princíp numerického riešenia diferenciálnej rovnice $y' = f(t, y)$, kde t je premenná reprezentujúca čas. Hľadaná funkcia $f(t)$ je aproximovaná diskrétnymi hodnotami $y(t_1)$ až $y(t_3)$, algoritmicky vypočítanými v jednotlivých krokoch t_1 až t_3 veľkosti h pre časový interval $\langle 0, tmax \rangle$.

Existuje viacero metód, z ktorých sú známe *Eulerova metóda, Taylorova metóda*, metódy *Runge-Kutta* a iné. Metódy tiež môžeme deliť na jednokrokové (napríklad Eulerova metóda)

a viackrokové metódy (napríklad metóda Adams-Bashforth), podľa počtu hodnôt potrebných pri výpočte v jednom kroku metódy. U viackrokových metód vstupujú do výpočtu v danom kroku i viaceré hodnoty z predchodzích krokov výpočtu, čo prináša vyššiu presnosť výpočtu.

Významným prvkom pri numerickom riešení problému je určenie chyby numerickej metódy. Takáto chyba často vzniká pri náhrade skutočnej hodnoty za jej aproximáciu a je dôležitým aspektom pri porovnávaní analytického a numerického riešenia. Existuje viacero typov chýb, z ktorých najzákladnejšou je *absolútna chyba* riešenia [45], ktorá môže byť vyjadrená ako rozdiel analytického a numerického riešenia (2.26), teda napríklad hodnôt napätia v čase, získaných analyticky a pomocou metód pre numerické riešenie diferenciálnych rovníc. Chyba súvisí s presnosťou numerických výpočtov, ktorej veľkosť v simulačných nástrojoch definujme pomocou premennej *eps*.

$$E(x) = x_{num} - x_{an} \tag{2.26}$$

Známa je aj *L2 chyba* [22], ktorá súvisí s normou vektora. Pre ukážku riešenia RC obvodu vo viacerých systémoch boli vyjadrené oba typy chýb (kapitola 3), pričom pre jednoduchosť uvádzam hodnoty absolútnej chyby v absolútnej hodnote.

Numerické riešenie lineárnych diferenciálnych rovníc je významné, pretože množstvo problémov popísateľných zložitými nelineárnymi rovnicami sa dá rozličnými spôsobmi previesť na lineárne. Je časovo náročné, no častokrát vďaka nemu obdržíme riešenie zložitých rovníc, ktoré by odvodiť analyticky a následne vyčísliť ani nebolo možné. Najjednoduchšie je riešenie lineárnej diferenciálnej rovnice 1. rádu (analyticky i numericky). Rovnice vyššieho rádu však dokážeme previesť na rovnice 1. rádu.

Numerické metódy sa spájajú i s pojmami, akými sú napríklad konvergencia k presnému riešeniu či stabilita [45]. Metódu považujeme za *konvergentnú*, ak pre ľubovoľnú počiatočnú úlohu platí

$$\lim_{h\to 0, n\to\infty} y_n = y(x), x = x_0 + n.h, \tag{2.27}$$

kde x_0 je hodnota počiatočnej podmienky, h je veľkosť kroku a n je počet krokov využitých pri výpočtoch. Numerická metóda je *stabilná*, ak sa hodnoty chyby výpočtu nezväčšujú, vzhľadom k presnému riešeniu.

Správne zostavenie rovníc je základným krokom pri analytickom či numerickom riešení elektrických obvodov. Okrem Ohmovho a II. Kirchhoffovho zákona je nápomocným aj I. Kirchhoffov zákon, ktorý hovorí, že "algebraický súčet všetkých prúdov v uzle je rovný nule" [47]:

$$\sum i = 0 \tag{2.28}$$

2.1.2.1 Numerická metóda výpočtu elektrických obvodov využitím Taylorovho radu

Metóda Taylorovho radu patrí medzi jednokrokové numerické metódy riešenia diferenciálnych rovníc [41] [45], využíva hodnoty z predchádzajúceho kroku výpočtu. Výhodou jednokrokových metód je, že pri zmene integračného kroku nemusíme hodnoty získané v predchodzích krokoch znova prepočítavať. Základom je však výpočet členov Taylorovho radu. S dostatočným počtom členov Taylorovho radu je možné aproximovať akúkoľvek funkciu.

Pri výpočte členov Taylorovho radu je potrebné určiť vyššie derivácie funkcie. To je možné za predpokladu, že daná funkcia f je diferencovateľná do potrebného rádu (pre danú funkciu existuje jej derivácia do potrebného rádu) [46].

Pre diferenciálnu rovnicu (2.5) má výpočet riešenia v $n+1$-vom kroku využitím Taylorovho radu tvar [41]:

$$y_{n+1} = y_n + h.f(t_n, y_n) + \frac{h^2}{2!}.f^{[1]}(t_n, y_n) + ... + \frac{h^p}{p!}.f^{[p-1]}(t_n, y_n), \tag{2.29}$$

kde y_n je hodnota riešenia v kroku n (predchodzí krok), h je integračný krok. Z horeuvedeného zápisu je zrejmé, že riešenie v $n+1$-vom kroku si vyžaduje i hodnoty derivácií funkcie f až do rádu $p-1$. Hodnotu p nazývame tiež *rádom metódy*. V tomto zápise teda funkciu y aproximujeme pomocou Taylorovho polynómu stupňa p.

2.1.2.2 Analýza paralelných vlastností metódy Taylorovho radu pri riešení CMOS obvodov

Pri tzv. *modernej metóde Taylorovho radu* [41] ide o tzv. *priame využitie Taylorovho radu*. Rád metódy (ORD) je volený automaticky. Zvyšovanie rádu metódy znamená zvyšovanie počtu členov Taylorovho radu pre výpočet hodnoty v danom kroku. Toto zvyšovanie sa uskutočňuje, až dokým pridanie ďalšieho člena radu neprestane zlepšovať presnosť riešenia (teda dokým sa nová hodnota neprestane meniť). So zvyšovaním rádu metódy však klesá rýchlosť výpočtu, no vďaka tomu dosiahneme vysokú presnosť. Rád metódy sa počas výpočtu môže ale i nemusí meniť. Presnosť výpočtu zvyšuje i zmenšovanie integračného kroku, no opäť na úkor rýchlosti výpočtu. Zmenšovanie kroku je však obmedzené výpočtovými zdrojmi.

Výpočet hodnôt aproximujúcich riešenie diferenciálnej rovnice (2.5) pre jednotlivé kroky má nasledovný tvar (pričom y_1 značí hodnotu riešenia v 1. kroku výpočtu a obdobne pre ďalšie kroky výpočtu):

$$\begin{aligned} y_0 &= y(0) \\ y_1 &= y_0 + h.y_0' + \frac{h^2}{2!}.y_0'' + \frac{h^3}{3!}.y_0''' + ... \\ y_2 &= y_1 + h.y_1' + \frac{h^2}{2!}.y_1'' + \frac{h^3}{3!}.y_1''' + ... \\ &... \end{aligned} \tag{2.30}$$

Tento zápis (2.30), ktorého prvý riadok predstavuje počiatočná podmienka úlohy, je možné upraviť. Označme 1. člen Taylorovho radu v prvom kroku výpočtu $DY1(0) = h.y_0'$. Obdobne označme ďalšie členy v ďalších krokoch výpočtu, teda:

$$\begin{aligned} DY1(0) &= h.y_0' \\ DY2(0) &= \frac{h^2}{2!}.y_0'' \\ DY3(0) &= \frac{h^3}{3!}.y_0''' \\ &... \\ DY1(1) &= h.y_1' \\ DY2(1) &= \frac{h^2}{2!}.y_1'' \\ DY3(1) &= \frac{h^3}{3!}.y_1''' \\ &... \end{aligned} \tag{2.31}$$

Touto substitúciou získame nasledovný tvar rovníc pre výpočet hodnôt hľadanej funkcie (hľadaného riešenia):

$$\begin{aligned} y_0 &= y(0) \\ y_1 &= y_0 + DY1(0) + DY2(0) + DY3(0) + ... \\ y_2 &= y_1 + DY1(1) + DY2(1) + DY3(1) + ... \\ &... \end{aligned} \tag{2.32}$$

U jednotlivých členov Taylorovho radu sú však problémom vyššie derivácie. Vyšším deriváciám sa matematici odjakživa snažili vyhnúť. Metóda s využitím Taylorovho radu však rieši výpočet jednotlivých členov $DY1(0), DY2(0), ...$ pomocou predchodzích členov radu, teda:

$$\begin{aligned} DY1(0) &= h.y'_0 \\ DY2(0) &= \frac{h}{2}.DY1(0) \\ DY3(0) &= \frac{h}{3}.DY2(0) \\ &... \\ DY99(0) &= \frac{h}{99}.DY98(0) \\ &... \end{aligned} \tag{2.33}$$

Potom numerické riešenie diferenciálnej rovnice, napríklad $y' = y$ s počiatočnou podmienkou $y(0) = 1$ je veľmi jednoduché:

$$\begin{aligned} DY1(0) &= h.y'_0 = h.y_0 = h.1 = h \\ DY2(0) &= \frac{h}{2}DY1(0) = \frac{h^2}{2} \\ DY3(0) &= \frac{h}{3}DY2(0) = \frac{h}{3}.\frac{h^2}{2} = \frac{h^3}{6} \\ &... \end{aligned} \tag{2.34}$$

Horeuvedeným spôsobom je možné vypočítať všetky potrebné členy pre výpočet hodnoty v jednom kroku i pre všetky kroky výpočtu a obdržať výsledné numerické riešenie, v podobe diskrétnych hodnôt, aproximujúcich hľadanú funkciu y. Touto metódou je možné riešiť i sústavy rovníc, pričom výpočty sa môžu uskutočňovať paralelne.

Zo spôsobu výpočtu vyšších derivácií Taylorovho radu vyplýva, že konkrétny člen radu je možné počítať na základe jeho predchodzieho člena. Bez neho nie je možné určiť aktuálny člen radu. Výpočet vyšších derivácií v rámci počítania hodnoty hľadanej funkcie v kroku výpočtu nesmeruje k možnej paralelizácii výpočtov. Tá vyplýva z výpočtov jednotlivých krokov metódy, vzhľadom na veľké množstvo na sebe vzájomne nezávislých výpočtov, ktoré je možné vykonávať paralelne:

$$\begin{aligned} &DY1(0) + DY2(0) + DY3(0) + ... \\ &DY1(1) + DY2(1) + DY3(1) + ... \\ &DY1(2) + DY2(2) + DY3(2) + ... \\ &... \end{aligned} \tag{2.35}$$

Pri implementácii paralelného riešenia systému diferenciálnych rovníc (5.1), pri ktorom môže byť výpočet distribuovaný do viacerých výpočtových jednotiek, môžeme prepísať vzťah (2.29) nasledovne [51]:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h(\mathbf{A}.\mathbf{y}_n + \mathbf{b}) + \frac{h^2}{2!}\mathbf{A}(\mathbf{A}.\mathbf{y}_n + \mathbf{b}) + ... + \frac{h^p}{p!}\mathbf{A}^{p-1}(\mathbf{A}.\mathbf{y}_n + \mathbf{b}) \tag{2.36}$$

Potom pre jedno jadro j, $j \in \{1...m\}$ (m je počet dostupných jadier) systému bude prináležať výpočet

$$\mathbf{A}_j = \sum_{k=0}^{\frac{n}{m}-1} \frac{h^{mk+j}}{(mk+j)!}\mathbf{A}^{mk+j-1} \tag{2.37}$$

Vzťah (2.36) môže byť potom vyjadrený nasledovne:

$$\mathbf{y}_{n+1} = ((\sum_{j=1}^{m} \mathbf{A}_j)\mathbf{A} + \mathbf{I})\mathbf{y}_n + (\sum_{j=1}^{m} \mathbf{A}_j)\mathbf{b} \tag{2.38}$$

pričom $\mathbf{I}$ predstavuje jednotkovú maticu.

V prípade, že sa rozhodneme použiť implicitnú Taylorovu metódu, postup je podobný, pričom vzťah (2.36) nadobudne tvar:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h(\mathbf{A}.\mathbf{y}_{n+1} + \mathbf{b}) - \frac{h^2}{2!}\mathbf{A}(\mathbf{A}.\mathbf{y}_{n+1} + \mathbf{b}) - ... - \frac{(-h)^p}{p!}\mathbf{A}^{p-1}(\mathbf{A}.\mathbf{y}_{n+1} + \mathbf{b}) \tag{2.39}$$

Implicitná Taylorova metóda je vhodná pre riešenie systémov vykazujúcich tuhosť. V tejto práci sa týmito systémami nebudeme zaoberať.

Kapitola 3

Simulácie elektrických obvodov vo svetových štandardoch

Funkčnosť navrhnutého elektrického obvodu sa doporučuje pred jeho skonštruovaním otestovať. Na to slúži množstvo nástrojov umožňujúcich daný obvod simulovať. Pri simulácii obvodu sa sledujú hodnoty veličín reprezentujúcich jednotlivé prvky zapojené v elektrickom obvode, ich zmeny, závislosti a podobne. Simulácia elektrického obvodu nám dokáže zodpovedať, ako sa daný obvod bude správať pri určitých parametroch, zadaných podmienkach, čím sa ušetrí množstvo zdrojov potrebných pre realizáciu funčného elektrického obvodu s požadovaným správaním, keď je možné vďaka výpočtom návrh obvodu podľa potrieb upravovať ešte pred jeho realizáciou.

3.1 Prehľad nástrojov používaných pri simulácii elektrických obvodov

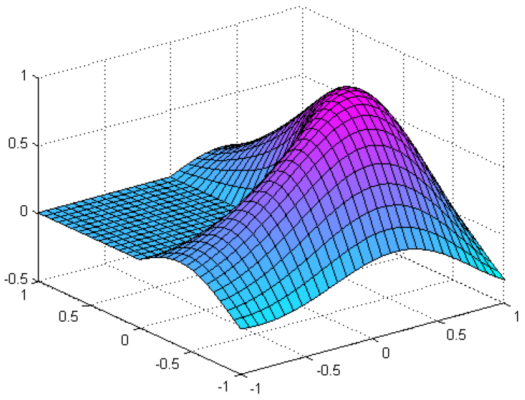
Existuje množstvo nástrojov umožňujúcich simulácie elektrických obvodov. Vo svetových štandardoch sú známe nástroje ako *MATLAB, Maple, Dymola* či *Spice*. V nasledujúcich podkapitolách budú predstavené základy práce s týmito nástrojmi, tvorba schém, modelov elektrických obvodov.

3.1.1 MATLAB

MATLAB (matrix laboratory) od spoločnosti *MathWorks* predstavuje rozsiahly nástroj umožňujúci širokú škálu matematických výpočtov, numerickú analýzu, využíva sa pri výrobe zariadení, produktov, mechanizmov, ktoré sa vyskytujú všade okolo nás a skvalitňujú náš život, napríklad v automobiloch, leteckých či komunikačných systémoch, pri strojovom učení. Je to platforma optimalizovaná pre riešenie mnohých problémov vedeckého či technického charakteru. Nápomocný je i v oblastiach, akými sú napríklad i robotika, medicína či štatistika [34].

MATLAB je taktiež programovacím jazykom, pomocou ktorého je možné implementovať rôzne algoritmy, manipulovať či vizualizovať dáta (i v 3D, obrázok 3.1). Spolupracuje s programovacími jazykmi ako C, C++, C#, Python, Java a iné. Podporuje objektovo-orientované programovanie [32]. Práca s vektormi, maticami, štruktúrami, grafmi i tvorba používateľských rozhraní je pre tento nástroj samozrejmosťou.

Tento matematický softvér má rozsiahle množstvo nástrojov, možností, nastavení, je vytvorený tak, aby používateľovi poskytol čo najväčší komfort pri práci, ale taktiež aby uspokojil rôznorodých používateľov s odlišnými potrebami. Preto je rozšírený, ako na školách tak i medzi vedeckými či technickými pracovníkmi. Prehľadná dokumentácia ako i množstvo existujúcich návodov zjednodušujú jeho použitie. Podpora pre Windows, Linux i MacOS umožňuje jeho využitie používateľom takmer všetkých kategórií.



Obr. 3.1: Ukážka 3D modelu vygenerovaného pomocou nástroja MATLAB [12], tvarovo podobného jeho samotnému logu.

3.1.1.1 Simulink

Rozšírenie MATLAB-u, veľmi užitočné pre modelovanie elektrických obvodov, predstavuje *Simulink* - grafické prostredie umožňujúce jednoduchú tvorbu a následnú simuláciu a analýzu širokej škály dynamických systémov, tzv. *Model-Based Design* [31]. Použitím intuitívnych nástrojov, prednastavených funkcií a možností je možné zostrojiť akýkoľvek obvod či celý systém, a to vo forme diagramu. Nie je možné používať Simulink mimo MATLAB.

Simulink umožňuje i automatické generovanie kódu, testovanie i verifikáciu zostrojeného modelu. Výsledky testovania možno exportovať do MATLAB-u pre ich ďalšie spracovanie. K Simulinku patrí i možnosť pripojenia modelu k hardwaru a následného testovania v reálnom čase [33].

Súčasťou tohto nástroja je grafický editor pre zostrojovanie modelu reprezentovaného blokovou schémou so svojou hierarchiou, funkcie umožňujúce spravovanie súborov a dát súvisiacich s modelom, ďalej knižnice obsahujúce množstvo predefinovaných blokov – pre systémy so spojitým i diskrétnym časom či blok spájajúci MATLAB algoritmy so zostrojeným modelom, nástroje podporujúce riešenie obyčajných diferenciálnych rovníc, obrazovky pre zobrazenie výsledkov simulácií, prostriedky analýzy modelu či nástroje zabezpečujúce import kódu C, C++ do modelu.

Spustenie nástroja Simulink je možné cez MATLAB v hlavnom hornom menu pomocou ikony Simulink alebo spustením príkazu `simulink` v príkazovom riadku MATLAB-u. Prvé spustenie zväčša trvá dlhšie. Po chvíli sa zobrazí okno Simulink, ktoré umožňuje hneď niekoľko možností, ako napríklad otvorenie existujúceho projektu či vytvorenie nového, tvorbu nového modelu, generovanie kódu a podobne. Následne zvolíme možnosť tzv. *Blank Model template* (prázdna šablóna modelu) a potom klikneme na tlačidlo *Create Model*

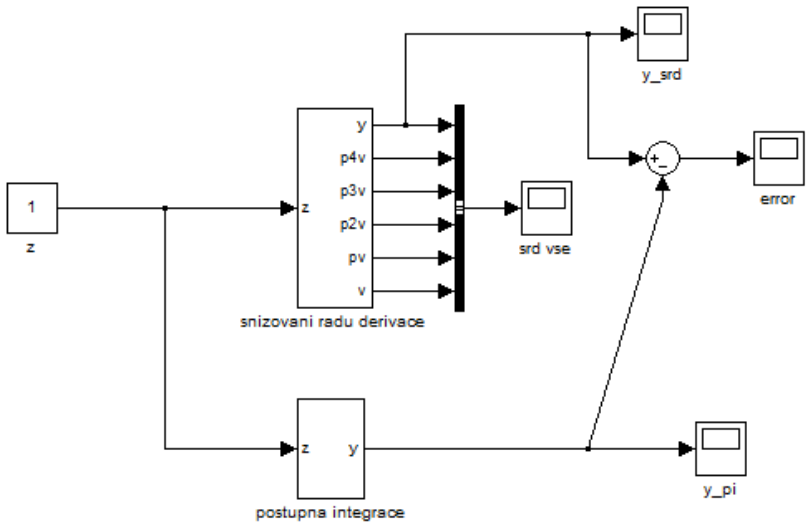
(vytvoriť model), ak chceme vytvárať vlastné modely. Rôzne verzie MATLAB-u sa v tomto môžu čiastočne líšiť.

Modely vytvárame spájaním blokov z prehliadača knižnice Simulinku (*Simulink Library Browser*). Pre každý blok je možné získať jeho bližšiu špecifikáciu, a to pravým kliknutím na vybraný blok a zvolením nápovedy (*Help for the...*). Parametre bloku sa zobrazia v dialógovom okne pomocou možnosti Parametre bloku (*Block Parameters*).

Pred spustením simulácie modelu sa doporučuje úprava štandardných parametrov konfigurácie podľa požiadavok. Sú to parametre ako čas simulácie, maximálna veľkosť kroku a iné nastavenia. Možno k nim pristúpiť v menu editora (*Simulation -> Model Configuration Parameters*) či pomocou ikony v jeho paneli nástrojov.

Zahájenie simulácie správania sa modelu nastáva voľbou *Simulation -> Run* (*Simulácia -> Spustiť*) v menu editora (alebo kliknutím na príslušnú ikonu v paneli nástrojov). Simuláciu možno kedykoľvek prerušiť či zastaviť.

Vizualizácia výsledkov simulácie sa uskutočňuje pomocou bloku *Scope* (rámec). Okno znázorňujúce graf stvárňujúci priebeh zmien hodnôt sledovaných veličín sa zobrazí dvojitým kliknutím na blok.



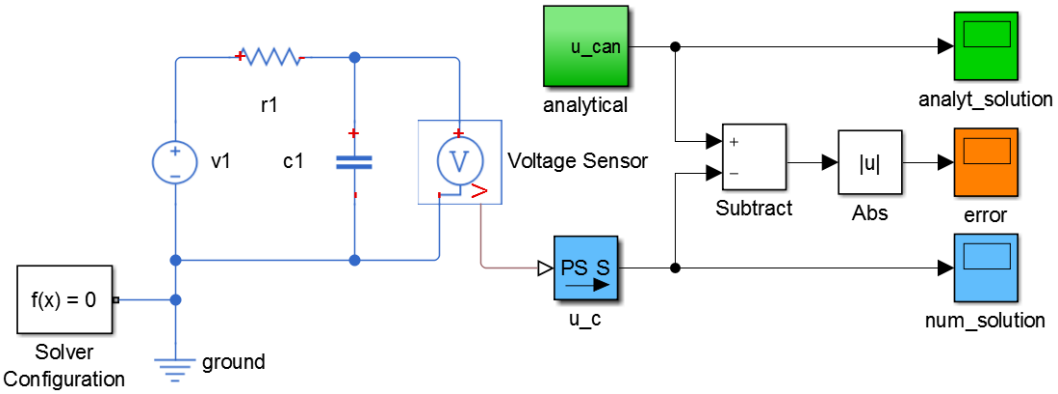
Obr. 3.2: Ukážka zostrojenia schémy v Simulinku pre riešenie lineárnej diferenciálnej rovnice štvrtého rádu (3.1) so vstupným signálom z, pričom bola využitá metóda postupnej integrácie alebo metóda znižovania rádu derivácie.

Pomocou nástroja Simulink je možné konštruovať modely riešiace diferenciálne rovnice, ako napríklad model riešiaci lineárnu diferenciálnu rovnicu štvrtého rádu (3.1) využitím metódy postupnej integrácie (PI). Vhodnou metódou je v tomto prípade i metóda znižovania rádu derivácie s pomocnou premennou v [38] (obrázok 3.2).

$$y'''' + a_3 y''' + a_2 y'' + a_1 y' + a_0 y = b_4 z'''' + b_3 z''' + b_2 z'' + b_1 z' + b_0 z \tag{3.1}$$

Spustením simulácie modelu z obrázku 3.2 obdržíme riešenie diferenciálnej rovnice (3.1), ktorým je funkcia y. Model je zostrojený tak, aby bolo možné graficky porovnať hodnoty riešenia získané metódou postupnej integrácie i metódy znižovania rádu derivácie (blok *error*).

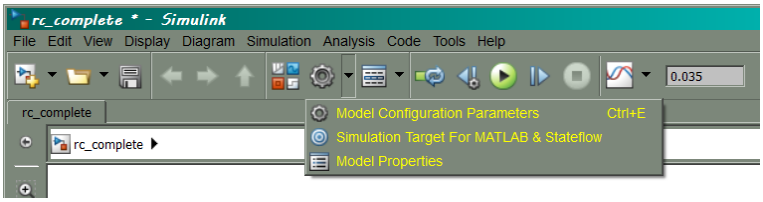
V Simulink-u som skonštruovala jednoduchý RC obvod, ktorého schéma je na obrázku 3.3. Tento obvod pozostáva z jednosmerného zdroja napätia $v_1 = 100V$ pripojeného do obvodu v čase $t_0 = 0s$, rezistora $r_1 = 200\Omega$ a kapacitora $c_1 = 10^-5F$. Ku kapacitoru je pripojený senzor na meranie napätia. Pre simulačné potreby som zaviedla do obvodu napríklad i blok konfigurácie tzv. *solvru* (*Solver Configuration*), ktorého hodnoty ostali nezmenené. Bez tohto bloku nie je možné v Simulink-u solvre použiť [42].



Obr. 3.3: Model RC obvodu s konštantným zdrojom napätia, vytvorený v prostredí Simulink, spolu s blokom pre analytické riešenie a výpočet absolútnej chyby riešenia.

Solvre reprezentujú súbor nástrojov riešiacich diferenciálne rovnice, bežne obsiahnuté v moderných simulačných systémoch. Simulink prináša hneď viacero typov solvrov, ktorých voľba závisí na type problému či zvolených podmienkach. Pre každý solver sú k dispozícii ďalšie nastavenia, bližšie špecifikujúce spôsob výpočtu u simulácie obvodu.

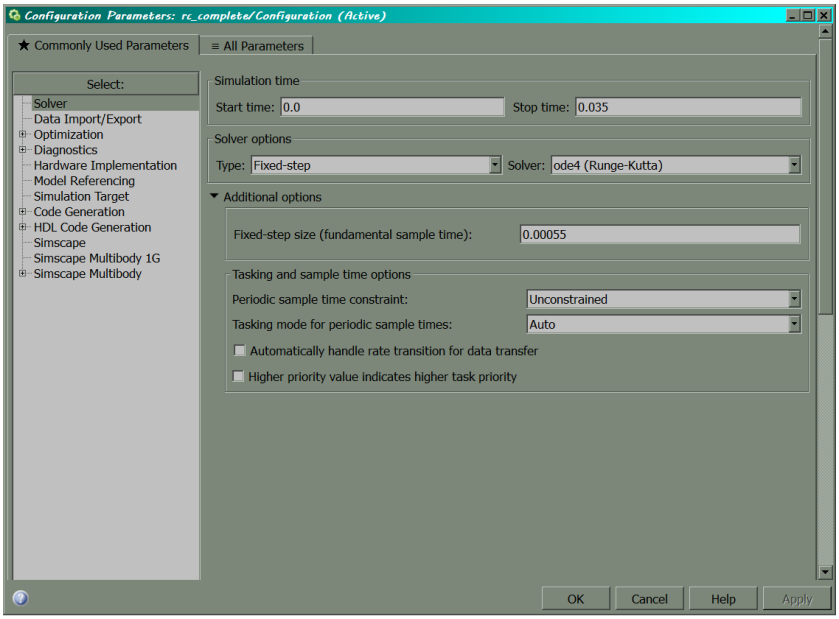
Riešením elektrického obvodu riešime tzv. počiatočnú úlohu. Preto je nevyhnutné definovať počiatočné podmienky, ktorými je v tomto prípade hodnota napätia na kapacitore v čase t_0 (čas zahájenia simulácie): $u_{c1}(t_0) = 0V$. Nastavenie počiatočnej podmienky pre kapacitor sa uskutočňuje v rámci špecifikácií samotného kapacitora.



Obr. 3.4: Kliknutím na ikonu *Model Configuration Parameters* získame v prostredí Simulink prístup ku možnostiam konfigurácie modelu a jeho simulácie.

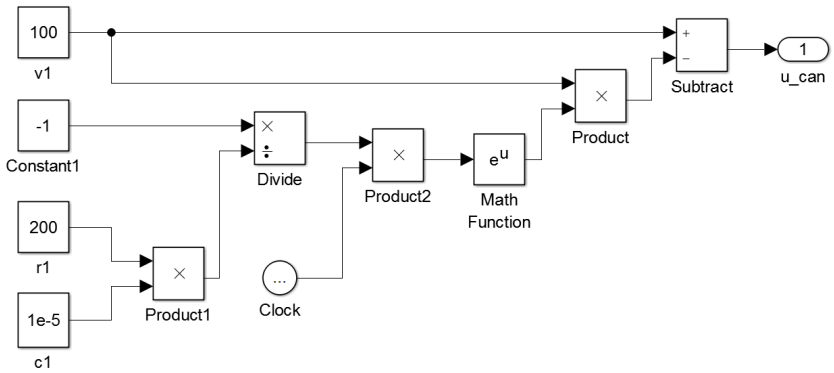
Pre simuláciu nabíjania kapacitora boli zvolené parametre ako čas zahájenia a ukončenia simulácie, typ metódy alebo solvru riešiaceho diferenciálne rovnice (napríklad *ode4 (Runge-Kutta)*). Voľba fixného kroku (*Fixed-step*) smeruje na špecifikáciu presnej hodnoty tohto kroku. Simulink umožňuje nastavenie i ďalších parametrov solvra či simulácie (obrázok 3.5). Rýchly prístup ku možnostiam konfigurácie modelu či jeho simulácie zabezpečuje

ikona *Model Configuration Parameters* (obrázok 3.4).



Obr. 3.5: Správna špecifikácia parametrov simulácie, akými sú napríklad čas počiatku, ukončenia simulácie, typ solvra a iné, sú dôležitou súčasťou každého simulačného procesu, nielen v prostredí Simulink.

Pomocou vstavaných solvrov príslušného simulačného prostredia získavame numerické riešenie navrhnutého modelu. Niektoré diferenciálne rovnice je možné riešiť i analyticky. Preto k modelu RC obvodu sa pripája i blok analytického riešenia (*analytical*) (obrázok 3.6), čím sa v Simulinku využije podpora tvorby podblokov, ich hierarchického usporadania.

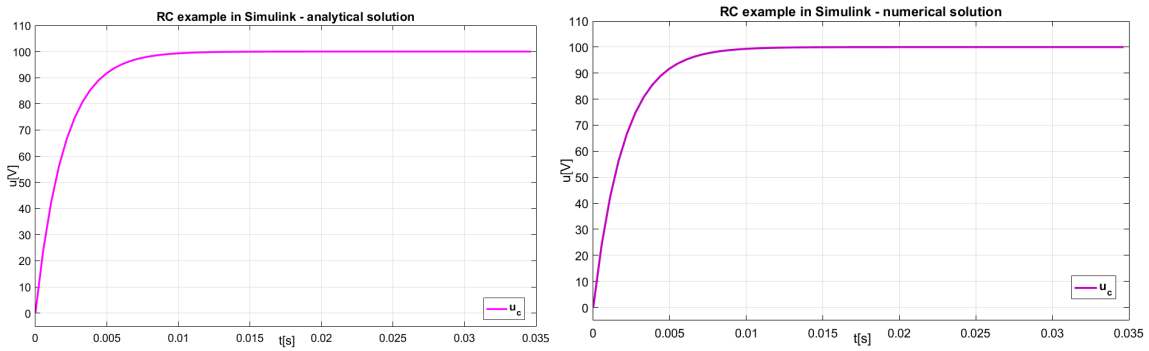


Obr. 3.6: Blok *analytical* v Simulink-u ako súčasť modelu RC obvodu z obrázka 3.3, riešiaci diferenciálne rovnice pre daný model analyticky.

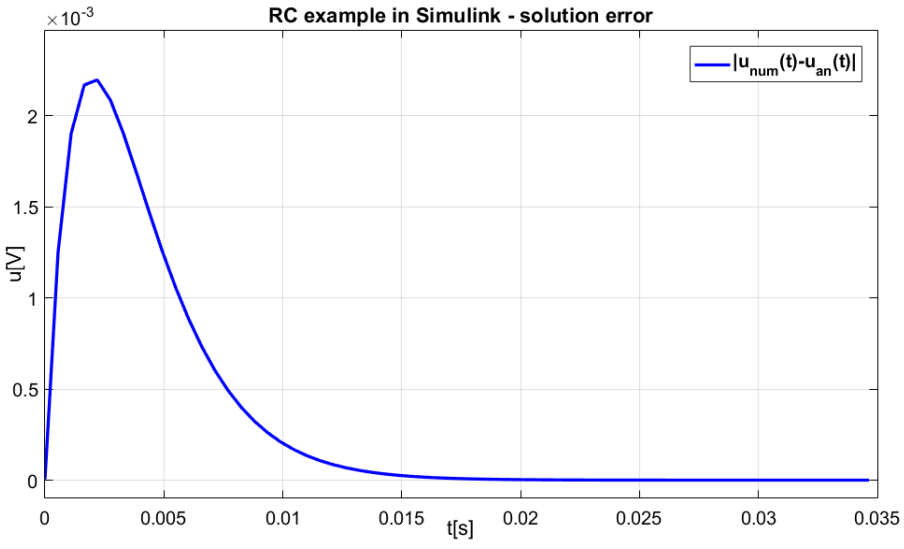
Blok analytického riešenia som skonštruovala pomocou blokov matematických operácií, konštánt. Prítomný je i blok reprezentujúci časové kroky simulácie [5]. Hodnota tzv.

decimovania (*deciamtion*) v tomto bloku je nastavená na 1, pre dosiahnutie požadovaného kroku.

Výsledkom simulácie je riešenie diferenciálnych rovníc, predstavujúce hodnoty napätia na kapacitore. Čím menšiu hodnotu kapacity na kapacitore pred spustením simulácie zvolíme, tým rýchlejšie sa kapacitor nabije, a teda tým rýchlejšie riešenie konverguje. Pri vhodne zvolenom čase simulácie môžeme z grafu obdržať hodnoty napätia pri prechodnom jave, predchádzajúcemu ustálenému stavu, kedy kapacitor dosiahne maximálnu hodnotu napätia, závislú na jeho kapacite.



Obr. 3.7: Grafy analytického (vľavo) a numerického riešenia RC obvodu - príklad nabíjania kapacitora, s jednosmerným zdrojom napätia, riešeného v prostredí Simulink.



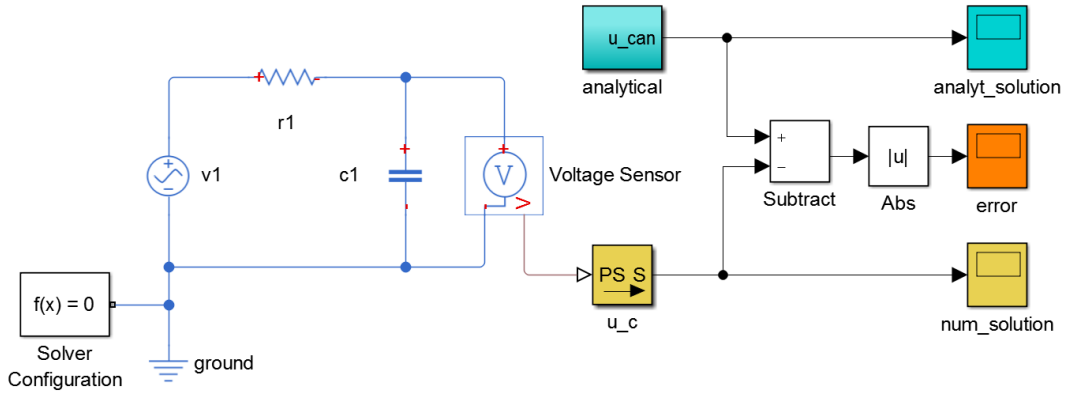
Obr. 3.8: Absolútna chyba analytického a numerického riešenia RC obvodu z obrázka 3.3, modelovaného v prostredí Simulink.

Grafy výsledného analytického a numerického riešenia sa na prvý pohľad neodlišujú (obrázok 3.7). Rozdiel riešení však existuje a vyjadruje ho absolútna chyba týchto riešení (absolútna hodnota rozdielu analytického a numerického riešenia v danom čase), taktiež

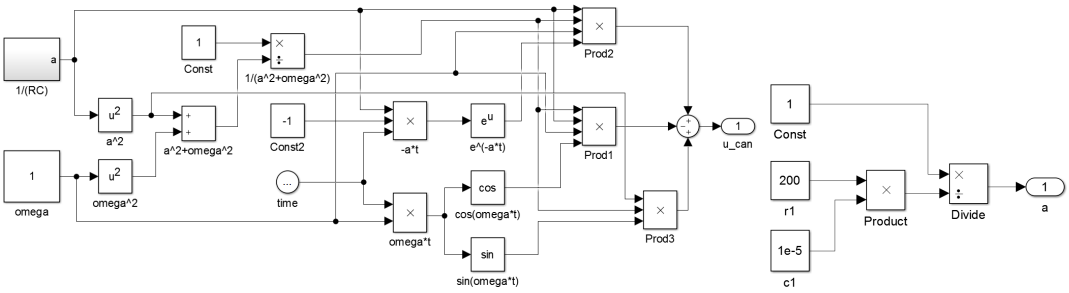
vynesená do grafu (obrázok 3.8). Táto chyba je malá, rádovo 10^{-3}, čo dokazuje správnosť riešení, pretože táto chyba by sa rádovo mala pohybovať v hodnotách konštanty *eps*, vyjadrujúcej presnosť výpočtu v MATLAB-e alebo Simulink-u, a táto podmienka je splnená. Parametre a nastavenia simulácie je potrebné voliť vhodne v závislosti na type problému. V tomto prípade boli všetky parametre vhodne zvolené.

V Simulink-u som skonštruovala obdobný model RC obvodu (obrázok 3.9), so striedavým zdroja napätia $u = sin(\omega t)$, $\omega = 1$, pripojeného do obvodu v čase $t_0 = 0s$. Pre ďalšie prvky obvodu som nastavila rovnaké hodnoty parametrov ako v prípade RC obvodu s jednosmerným zdrojom napätia. Ku kapacitoru je opäť pripojený senzor merania jeho napätia.

Analytické riešenie tohto obvodu je o niečo zložitejšie, čo sa odrazilo i pri zostavení bloku (*analytical*) (obrázok 3.10). Pre zjednodušenie schémy analytického riešenia obvodu so striedavým zdrojom napätia bola zadefinovaná konštanta a.



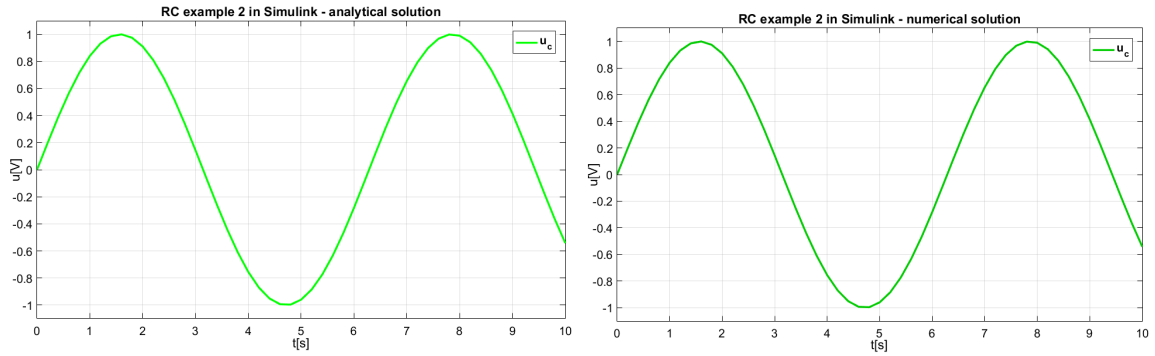
Obr. 3.9: Model RC obvodu so striedavým zdrojom napätia, vytvorený v prostredí Simulink, spolu s blokom pre analytické riešenie a výpočet absolútnej chyby riešenia.



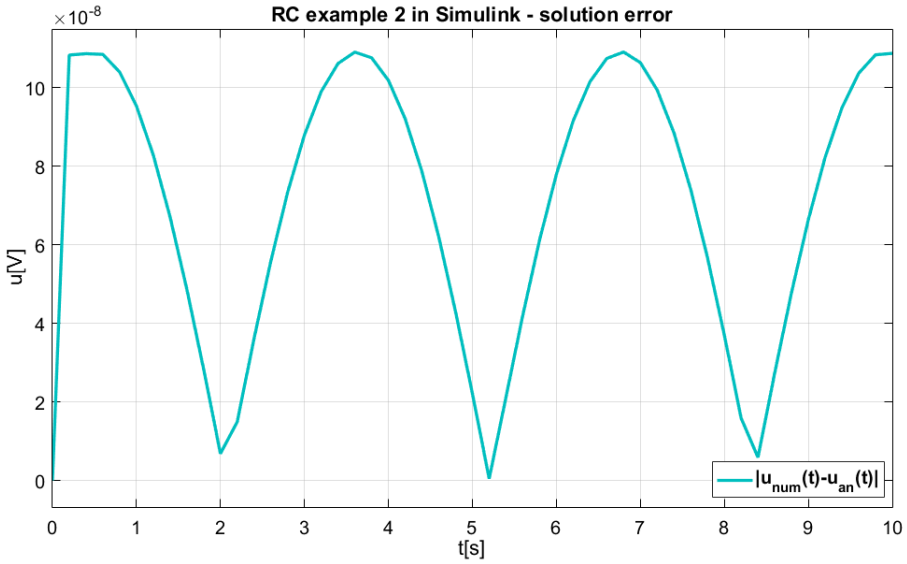
Obr. 3.10: Súčasťou modelu RC obvodu z obrázka 3.9 je blok *analytical*, analyticky riešiaci v Simulink-u diferenciálne rovnice pre daný model (vľavo), ktorého súčasť tvorí i podblok pre konštantu a (vpravo), závislý na hodnote odporu na rezistore r_1 a kapacity na kapacitore c_1.

Parametre simulácie som zvolila odlišné v porovnaní s prvým príkladom obvodu (RC obvod s konštantným zdrojom napätia): $t = 10s$, fixný integračný krok s veľkosťou $dt = 0.2s$, automatický výber solvra (*Automatic solver selection*), čo zabezpečilo uspokojivé výsledky výpočtov.

Grafy výsledného analytického a numerického riešenia sa opäť na prvý pohľad neodlišujú (obrázok 3.11). Absolútna chyba týchto riešení je opäť veľmi malá, rádovo 10^{-8}, voľba automatického solvra je pre tento model obvodu a jeho parametre vhodná.



Obr. 3.11: Grafy analytického (vľavo) a numerického riešenia RC obvodu so striedavým zdrojom napätia $u = sin(\omega t)$ - príklad nabíjania kapacitora, riešeného v prostredí Simulink.



Obr. 3.12: Absolútna chyba analytického a numerického riešenia RC obvodu so striedavým zdrojom napätia z obrázka 3.9, modelovaného v prostredí Simulink.

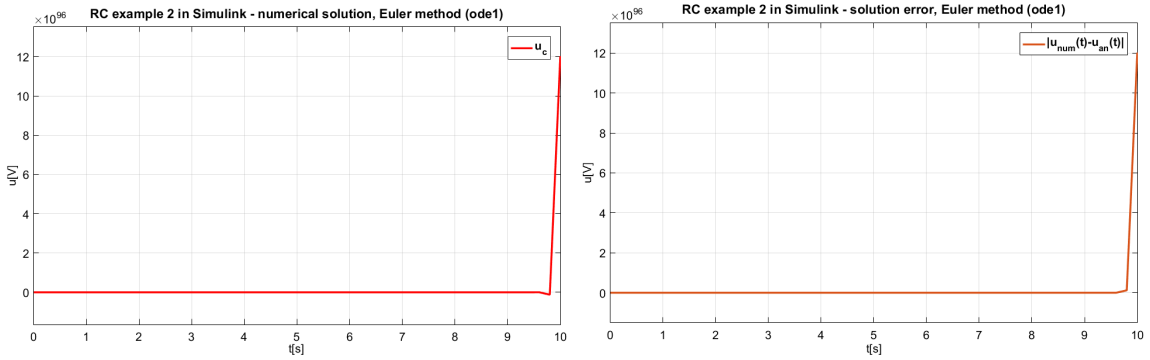
V prípade voľby *ode1* solvra však výsledky výpočtov nie sú uspokojivé, riešenie diverguje (obrázok 3.13). Tento typ solvra pre danú úlohu (RC obvod) nepostačuje. Dôvodom je použitá numerická metóda (Euler). Voľba solvra a ďalších parametrov simulácie i správny návrh modelu zohráva dôležitú úlohu pri získaní relevantných výsledkov, podľa možností čo najviac priblížujúcich sa výsledkom, ktoré by sme namerali na skutočných obvodoch.

Rozdiel analytického a numerického riešenia sa vyjadruje i pomocou *L2 chyby*. Hodnotu tejto chyby môžeme vyčísliť napríklad v MATLAB-e, pridaním bloku *To Workspace* do schémy, čím si zabezpečíme prístup k hodnotám riešení zo Simulink-u. Príkazom *norm*

Tabuľka 3.1: L2 chyba analytického a numerického riešenia príkladov RC obvodu v prostredí Simulink, získaného voľbou rozličných numerických metód.

Zdroj napätia	Solver	L2 chyba
konštantný	Runge Kutta	0.0057
striedavý	auto (Automatic solver selection)	5.6094e-07
striedavý	ode1 (Euler)	1.2025e+97

obdržíme výslednú hodnotu, ktorá je opäť veľmi malá, podobná hodnotám absolútnej chyby z grafu 3.8 alebo 3.12 (tabuľka 3.1).



Obr. 3.13: Graf numerického riešenia RC obvodu so striedavým zdrojom napätia v Simulinku, získaného pomocou *ode1* solvra (Eulerova metóda) (vľavo) a graf absolútnej chyby numerického a analytického riešenia, vykazujúci enormný nárast chyby v čase medzi 9. a 10. sekundou simulácie.

3.1.2 Maple

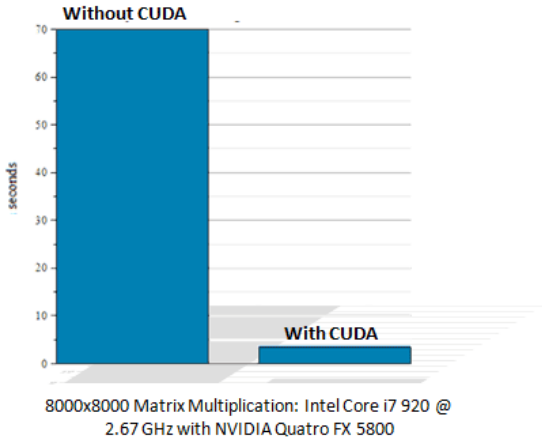
Ďalším matematickým nástrojom so všestranným použitím je *Maple* od firmy *Maplesoft*. Jeho silnou stránkou je kombinácia presných a rýchlych matematických výpočtov s používateľským prostredím, ktoré zabezpečuje jednoduché použitie, vizualizáciu i analýzu problémov rôznorodého charakteru, pričom použiteľnosť sa neznižuje na úkor kvality výpočtov a naopak. Podobne ako MATLAB, je tento softvér vhodný pri výskume i pri výuke [35], podporovaný na všetkých rozšírených platformách: Windows, Linux i MacOS.

Maple je silným matematickým prostriedkom pre riešenie značného množstva úloh, disponuje širokou škálou funkcií z takmer všetkých oblastí matematiky, ako napríklad lineárna algebra, štatistika, analýza dát, geometria, výpočet rovníc (vrátane diferenciálnych) a podobne. Obsahuje nástroje, ktoré hravo zvládnu nejeden optimalizačný problém. Funkcie, ako nastavenie vlastných jednotiek alebo výber z ponuky, nastavenie požadovaných rozmerov či určitej tolerancie pri výpočtoch, sú štandardom. Maple nás upozorní, ak zadáme nesprávne jednotky, nekompatibilné s našimi už existujúcimi vstupmi, čím sa vylúči značné množstvo chýb, ktoré môžu pri práci s týmto nástrojom vzniknúť. Podporuje *MathML 2.0*, rozšírenie HTML jazyka, definujúce štandardný formát matematických výrazov, spon-

zorovaný konzorciom *W3C* (*World Wide Web Consortium*). Vďaka tomuto rozšíreniu je používateľovi umožnený export výrazov Maple do webstránok, obohatených o MathML [36].

Maple obsahuje množstvo vizualizačných nástrojov, podporuje 2D i 3D zobrazovanie. Nejedno kontextovo-senzitívne menu zjednodušuje používanie matematických operácií - operácia sa z menu aplikuje jedným kliknutím. Takzvaní interaktívni asistenti sú ďalším podporným nástrojom, umožňujúcim používateľovi vyriešiť množstvo drobných záležitostí a otázok, uľahčuje zobrazovanie nastavení, monitoruje priebeh algoritmov, dovoľuje vytvárať jednoduché aplikácie nápomocné pri riešení širokej variety úloh.

Obdobne ako MATLAB, Maple disponuje svojim vlastným programovacím jazykom spolu s integrovaným vývojovým prostredím (IDE) ideálnym pre tvorbu jednoduchých programov, skriptov s možnosťou odladenia kódu a ďalšími nástrojmi patriacimi k programovaniu (napríklad zvýrazňovanie syntaxe v editore kódu, overovanie správneho uzatvorenia zátvoriek, správne odsadenie, import kódu z textového súboru a opačne). Možno tu využívať princípy objektovo-orientovaného i funkcionálneho programovania. Otázka bezpečnosti pritom nie je opomenutá. Statická analýza kódu napomáha detekovaniu chýb už pri písaní samotného kódu, čo vo vysoke miere zefektívňuje prácu [23]. Generovanie kódu do ďalších jazykov (napríklad Java, JavaScript, MATLAB, Perl, Python, Fortran, C, C#) umožňuje využiť výpočtovú silu Maple v mnohých aplikáciách.



Obr. 3.14: Graf rýchlosti výpočtu maticového násobenia v *Maple* s využitím *CUDA* a bez [24]. Modré obdĺžniky poukazujú na markantný rozdiel v čase potrebnom na daný výpočet, ktorý vďaka CUDA podpore možno mnohonásobne znížiť.

Paralelizácia výpočtov sa uskutočňuje automaticky po detekovaní prítomných výpočtových zdrojov (jadier). To znamená, že paralelizácia sa vykonáva bez akéhokoľvek zásahu používateľa, čo prináša výsledky kalkulácií oveľa rýchlejšie než u iných nástrojov či v porovnaní so získaním výsledkov bez použitia podpory paralelizácie.

Ďalšou výhodou v súvislosti s rýchlosťou vykonávania matematických výpočtov v Maple je podpora hardvéru s tzv. *CUDA* (*NVIDIA Compute Unified Device Architecture*), čo nie je len jazyk alebo API, no predovšetkým významná technológia, platformoa a zároveň programovací model, zjednodušujúci využitie grafických kariet pri vykonávaní a urýchľovaní

paralelných výpočtov (obrázok 3.14) [24].

matrix size (N)	Maple 2016.1 (64 bit)	Mathematica 11 (64 bit)	Matlab 2016a (64 bit)
500	0.036	0.024	0.0414
1000	0.141	0.134	0.138
1500	0.650	0.616	0.634
2000	2.08	2.033	2.053
2500	4.548	4.504	4.549
3000	2.313	8.393	2.595
3500	3.523	13.865	3.465
4000	5.215	22.088	5.052
4500	7.108	30.846	6.89
5000	8.129	43.079	8.005
5500	10.375	58.216	10.181
6000	13.543	75.655	13.466
6500	15.884	96.048	15.915
7000	19.673	120.505	19.000
7500	23.141	148.593	22.529
8000	28.789	180.311	28.095

Tabuľka 3.2: Rýchlosť výpočtu hodnosti štvorcovej matice v nástroji *MATLAB 2016a, Maple 2016.1, Mathematica 11*, na 64-bitovom operačnom systéme Windows 7 Home Premium, 16GB RAM. Rýchlosť je uvedená v sekundách. Ľavý stĺpec predstavuje veľkosť matice N [16].

Pri porovnaní MATLAB-u a Maple je možné nájsť odlišnosti (tabuľka 3.2), pričom vždy záleží na tom, z akého hľadiska nástroje konfrontujeme, napríklad z hľadiska symbolických či numerických výpočtov, vizualizácie, možností programovania algoritmov a podobne [21]. Množstvo ľudí neustále uskutočňuje testy produkujúce údaje o rýchlosti výpočtov, pretože rýchlosť sa zdá byť jedným z rozhodujúcich parametrov pri výbere vhodného nástroja pre zvládnutie úloh, predovšetkým tých náročnejších. Jeden z mnohých experimentov merajúci rýchlosť výpočtu hodnosti štvorcovej matice v nástroji MATLAB 2016a (64bit), Maple 2016.1 (64bit), naznačuje, že tieto nástroje sú si takmer rovnocenné. V mnohých prípadoch potreboval MATLAB na získanie výsledku o niečo menej času, no tento rozdiel sa týkal len desatín až stotín sekúnd, a to aj pri veľkosti matice $N = 8000$ [16]. Namerané hodnoty pochopiteľne závisia od použitého hardvéru, verzie softvéru i veľkosti matíc, algoritmov zasadených do jednotlivých výpočtových systémov a ďalších aspektov. Ideálne je pracovať s najnovšími verziami, voliť náročnejšie úlohy a bežne dostupný hardvér pre získanie podľa možnosti čo najaktuálnejších a najrelevantnejších výsledkov. Do algoritmov implementovaných v nástrojoch vo väčšine prípadov nie je používateľovi umožnené alebo vhodné zasahovať, no súčasné trendy sa snažia o čo najširšie možnosti použitia nástrojov, čo robí implementáciu vlastných alebo neštandardných algoritmov jednoduchšou.

3.1.2.1 MapleSim

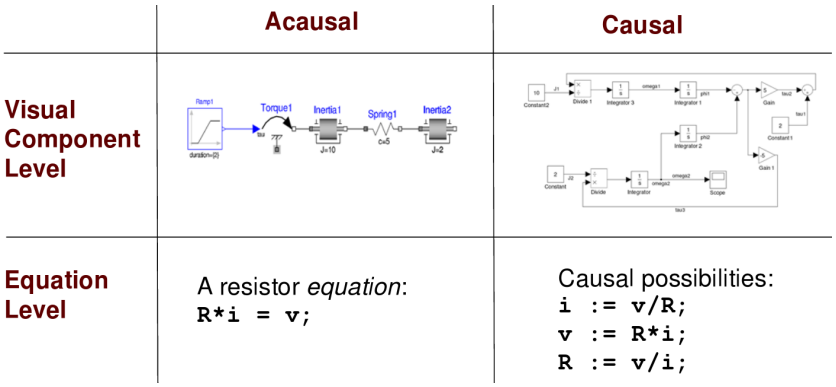
MapleSim je významný simulačný prostriedok, pokrývajúci mnoho oblastí vedy a techniky. Rozširuje funkcionalitu Maple podobne ako Simulink v MATLAB-e, v značnej miere redukuje čas potrebný pre vyvinutie prototypov, modelov, poskytuje hlbší náhľad do systému, ktorý chceme modelovať. Na rozdiel od Simulink-u si však vyžaduje osobitnú inštaláciu, následnú aktiváciu a funkčný Maple.

Pri práci s MapleSim spolu s výpočtovou silou Maple existuje nemalé množstvo možností, ktoré poskytujú rýchlejšie a presnejšie výsledky, pohodlnejšiu manipuláciu s prvkami daného

systému. Je postavený na novorešpektovanom štandarde, definujúcom popis modelov a ich súčastí, zvanom *Modelica*, jazyku, ktorý je otvorený, objektovo-orientovaný, čo podnecuje jeho rozšíriteľnosť a jednoduché použitie [25].

Rovnako ako Simulink, aj MapleSim disponuje širokou ponukou zabudovaných komponentov (nad 600), vďaka ktorým vytvoríme model bez väčšej námahy [26]. Pripúšťa sa pritom i kombinácia prvkov z viacerých sfér, pričom overenie správnosti vybraných prvkov, ktoré umiestnime pri modelovnaí do vzájomnom vzťahu, je samozrejmosťou. Ide o komponenty alebo tzv. bloky z oblastí ako mechanika, termodynamika, magnetizmus, hydraulika a iné. My sa však budeme zaujímať predovšetkým o rovinu zostrojovania modelov elektrických obvodov využitím komponentov ako tranzistory, rezistory, diódy, kondenzátory a iné. Pomocou doplňujúcich produktov MapleSim dokážeme zostrojiť i bloky podľa vlastných potrieb a následne ich použiť. Komponent sa z prehľadnej knižnice zasadí do pracovnej plochy vyhradenej pre model obdobne ako u Simulinku – vybratím komponentu a premiestnením do plochy na zvolené miesto.

Pre MapleSim je typické tzv. *nekauzálne* modelovanie (acausal), ktoré sa na rozdiel od väčšiny nástrojov určených pre simulácie s kauzálnym (causal alebo i signal-flow) modelovaním (napr. Simulink) zakladá na inom prístupe, smerujúcom ku skutočnému správaniu sa systému (modelu) (obrázok 3.15). Rozdiel okrem iného i v tom, že toto modelovanie pripúšťa tok signálu (informácií) oboma smermi, nie iba jedným. V MapleSim však máme k dispozícii oba prístupy, ktoré spolu môžu dobre fungovať a navzájom sa dopĺňať [44].

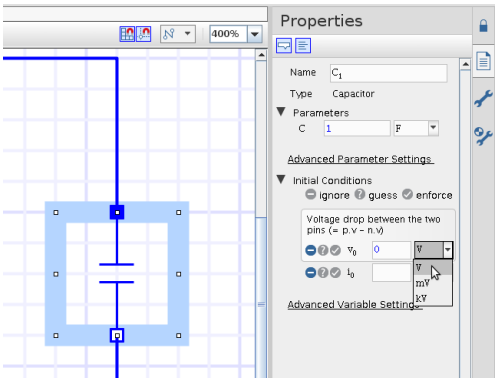


Obr. 3.15: Rozdiel medzi nekauzálnym (acausal, ľavý stĺpec) a kauzálnym (causal alebo i signal-flow, pravý stĺpec) prístupom k vytváraniu modelov pomocou počítačových systémov [50]. Prvý riadok reprezentuje vizuálnu rovinu, ktorú nekauzálny prístup nielen zjednodušuje, ale i približuje skutočnosti a umožňuje lepšie pochopenie modelu. Z hľadiska rovníc potrebných na popísanie systému ide pri nekauzálnom prístupe opäť o zjednodušenie, redukciu počtu rovníc.

Diagram (schému), znázorňujúci elektrický obvod, je možné zostrojiť ako v Simulinku – výberom vhodných blokov a vytvorením spojení medzi nimi. Bloky (obrázok 3.16) môžeme vyberať na pracovnú plochu MapleSim (*Model Workspace*) z knižnice z príslušnej sekcie (*Electrical*) [44].

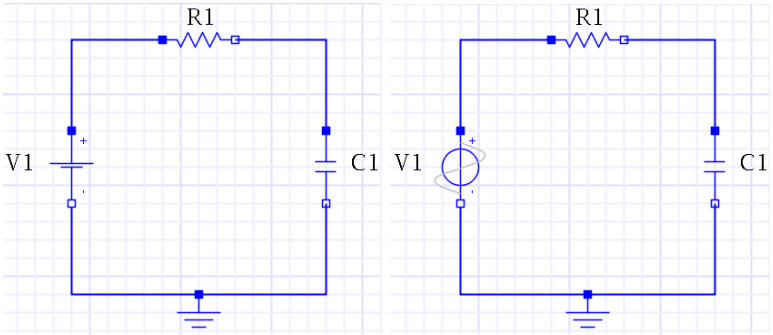
Voľby simulácie (*Simulation Settings*) v ľavej časti prostredia umožňujú presné definovanie dĺžky simulácie t_d a ďalších parametrov. V niektorých príkladoch (sekcia *Príklady* alebo *Examples*) sa doporučuje možnosť Kompilátora (*Compiler*) v nastaveniach simulácie

nechať vypnutú. Počiatočné podmienky je možné upresniť v nastaveniach v pravej časti prostredia MapleSim (*Settings*). Spustenie simulácie kliknutím na tlačidlo ➤ z hlavného menu automaticky vygeneruje rovnice popisujúce systém, ktoré následne i vyrieši pomocou vstavaného nástroja riešiaceho diferenciálne rovnice a vykreslí graf reprezentujúci hodnoty veličín nášho záujmu.



Obr. 3.16: Výber bloku – kapacitora a zobrazenie jeho konkrétnych parametrov, nastavenie počiatočných podmienok, označenie (pomenovanie) bloku v nástroji MapleSim.

Pochopiteľne, pri modelovaní používame bloky predstavujúce matematické modely skutočných komponentov obvodu, ktoré charakterizujeme jednou významnou fyzikálnou veličinou, vlastnosťou a ich ďalšie vlastnosti sú menej rozhodujúce. Model rezistoru či kapacitoru (ideálne súčiastky) sú lineárnymi elektrickými súčiastkami, a preto obvody z nich pozostávajúce nazývame lineárne elektrické obvody [47].



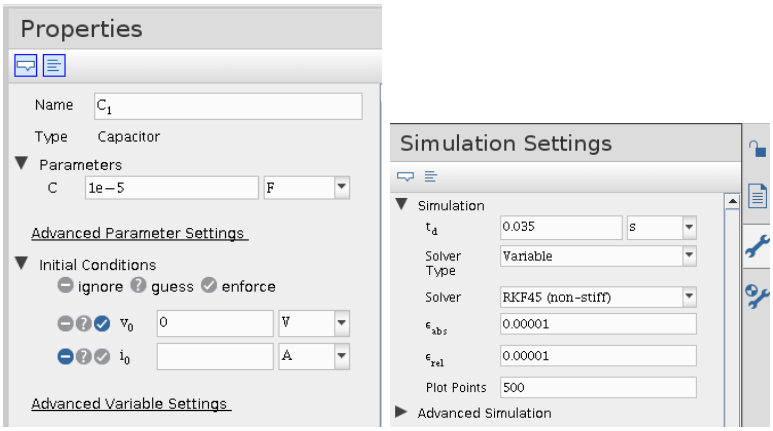
Obr. 3.17: Modely RC obvodov s konštantným i striedavým zdrojom napätia, vytvorené v prostredí MapleSim.

Simuláciu obvodov v MapleSim som otestovala pre varianty RC obvodu - s jednosmerným i striedavým zdrojom napätia, podobne ako v Simulink-u. Obvod na obrázku 3.17 vľavo odpovedá obvodu vytvoreného v Simulink-u na obrázku 3.3. Náročnosť konštruovania sa v porovnaní so Simulink-om javí nižšia, keďže nie je potrebné do modelu obvodu zavádzať špeciálne bloky (ako napríklad Solver Configuration, senzory na meranie hodnôt

veličín) na to, aby korektne prebehla simulácia obvodu a požadované výsledky mohli byť reprezentované grafom.

Pre zobrazenie numerického riešenia RC obvodu v MapleSim postačuje špecifikovať hodnoty veličín pre jednotlivé prvky obvodu, počiatočnú podmienku pre napätie na kapacitore a nastavenia simulácie (obrázok 3.18). Získanie analytického riešenia však vyžaduje odlišný prístup, pričom možno využiť výsledky výpočtov prostredia Maple, alebo upraviť existujúci model obvodu v MapleSim využitím jazyka Modelica [8]. Pre oba modely RC obvodu som zvolila druhú možnosť (spolu s vytvorením vlastného komponentu), z dôvodu triviálnej úpravy zdrojového súboru v jazyku Modelica, ktorý je univerzálnym jazykom pri vytváraní modelov v mnohých odvetviach.

Možnosť *Custom Component* zabezpečuje tvorbu vlastného komponentu, ktorý sa môže ľubovoľne použiť pri vytváraní modelu obvodu. V prípade, že chceme, aby bol komponent definovaný pomocou jazyka Modelica, vytvoríme tzv. *Modelica Custom Component*: klikneme na ikonku *Pridať aplikácie alebo šablóny* (*Add Apps or Templates*) v ľavej časti prostredia, kde pod záložkou *Templates* vyberieme Modelica Custom Component a otvoríme nové okno pre zadefinovanie tohto špeciálneho komponentu [44].



Obr. 3.18: Špecifikovanie hodnoty počiatočnej podmienky - napätia na kapacitore $C1$ a jeho kapacity v pravej časti prostredia MapleSim, v záložke *Properties* (vľavo) a nastavenie ďalších premenných potrebných pre uskutočnenie simulácie daného elektrického obvodu, v záložke *Simulation* Settings (vpravo).

Editácia kódu v jazyku Modelica, reprezentujuceho numerické riešenie RC obvodu, vygenerovaného pomocou MapleSim, sa uskutoční tak, aby sa v ňom nachádzal aj kód analytického riešenia, čo umožní výpočet chyby riešenia triviálnym spôsobom. Preto skopírujeme kód numerického riešenia (napríklad klikneme na ikonku *Code View* v hornom paneli prostredia). Následne volíme možnosť *New -> Model* a vytvoríme Modelica Custom Component vyššie uvedeným spôsobom. Vložíme zdrojový kód skopírovaného numerického riešenia a pridáme doň riadky pre analytické riešenie a chybu výpočtu:

```
constant Real rr1 = 200;
constant Real cc1 = 1e-5;
constant Real vv1 = 100;
Real u;
```

```
    Real er;
```

Definovanie konštánt nie je nutné, no sprehľadňuje a zjednodušuje definovanie analytického riešenia a chyby výpočtu. Do sekcie `equation` pridáme nasledujúce (RC obvod s konštantným zdrojom napätia):

```
    u = vv1-vv1*Modelica.Math.exp((-1/(rr1*cc1))*time);
    er = abs(u-C1.v);
```

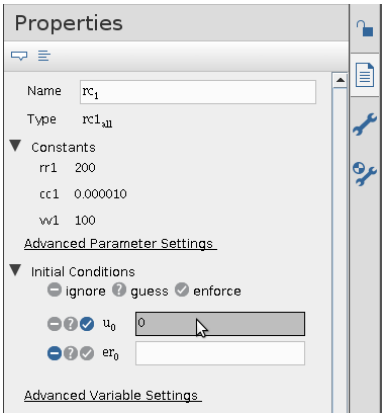
Pre obvod so striedavým zdrojom napätia použijeme odlišné rovnice:

```
    constant Real rr1 = 200;
    constant Real cc1 = 1e-5;
    constant Real om  = 1;
    Real a;
    Real u2;
    Real er2;
    ...
equation
    a = 1/(rr1*cc1);
    u2 = (1/(a*a+om*om))*a*om*exp(-a*time)-(1/(a*a+om*om))*
         a*om*cos(om*time)+(1/(a*a+om*om))*a*a*sin(om*time);
    er2 = abs(u2-C1.v);
    ...
```

Vytvorený komponent premenujeme, napríklad na $rc1_{all}$ pre obvod s konštantným zdrojom napätia (inak než *Main*, čo je štandardne vygenerované meno) a uložíme. Vyberieme tento komponent z ľavej záložky *Local Components* a umiestnime ho do pracovnej plochy pre tvorbu modelu v MapleSim. Tento jediný komponent predstavuje nový model - pre numerické i analytické riešenie a chybu výpočtu. Opäť nastavíme parametre simulácie (v *Simulation Settings*, obrázok 3.18 vpravo), počiatočnú podmienku (v *Properties*) a zahájime simuláciu. Nastavenie počiatočnej podmienky je v tomto prípade odlišné od nastavenia pri pôvodnom modeli pre samotné numerické riešenie - počiatočnú podmienku nedefinujeme pre kapacitor $C1$ (tá je už obsiahnutá v pôvodnom zdrojovom kóde pre numerické riešenie), ale pre napätie u na kapacitore (konštanta $cc1$), reprezentujúce analytické riešenie (obrázok 3.19). Obdobne postupujeme pre príklad so striedavým zdrojom napätia. Ak simulácia prebehne v poriadku, automaticky sa otvorí okno pre vyobrazenie výsledkov (*Analysis Window*).

Pre simuláciu obvodu v MapleSim som zvolila variabilný krok a adekvátne k nemu typ solvra (pre stabilný systém). Výsledky simulácie pre jednosmerný i striedavý zdroj napätia RC obvodu (obrázky 3.20 a 3.21) sú veľmi podobné výsledkom získaných v programe Simulink. Pri voľbe fixného kroku taktiež dostaneme uspokojivé výsledky. Krivka na grafe, predstavujúca vývoj hodnôt napätia na kapacitore v čase pri jeho nabíjaní, však nemusí byť zaoblená, čo súvisí s veľkosťou voleného integračného kroku. Ak sa zvolí príliš veľký krok a Eulerova metóda výpočtu, riešenie nebude stabilné.

Grafy absolútnych chýb riešení (obrázok 3.22) sa však od grafov, ktoré som obdržala v Simulink-u, líšia, a to najmä vďaka voľbe iného solvra, kroku (variabilný) a nastavení presnosti výpočtov pre MapleSim. Hodnoty na grafoch sú však stále veľmi malé, a preto sú výsledky týchto simulácií prijateľné.

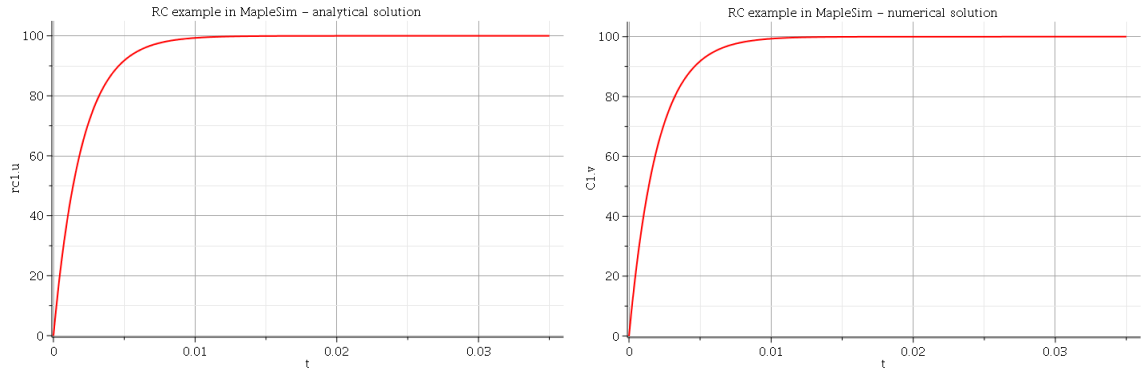


Obr. 3.19: Nastavenie počiatočnej podmienky - napätia u na kapacitore pre výpočet analytického riešenia modelu RC obvodu v prostredí MapleSim.

Pre dosiahnutie ešte väčšej presnosti výsledkov (a menšej chyby riešenia) je vhodné striedavý zdroj napätia definovaný $u = sin(\omega t)$, $\omega = 1$, nahradiť tzv. *tvoriacimi diferenciálnymi rovnicami* (3.2), čím sa zníži chyba, ktorá sa môže pri použití matematickej operácie *sinus* kumulovať a v najhoršom prípade zapríčiniť nestabilitu riešenia [37].

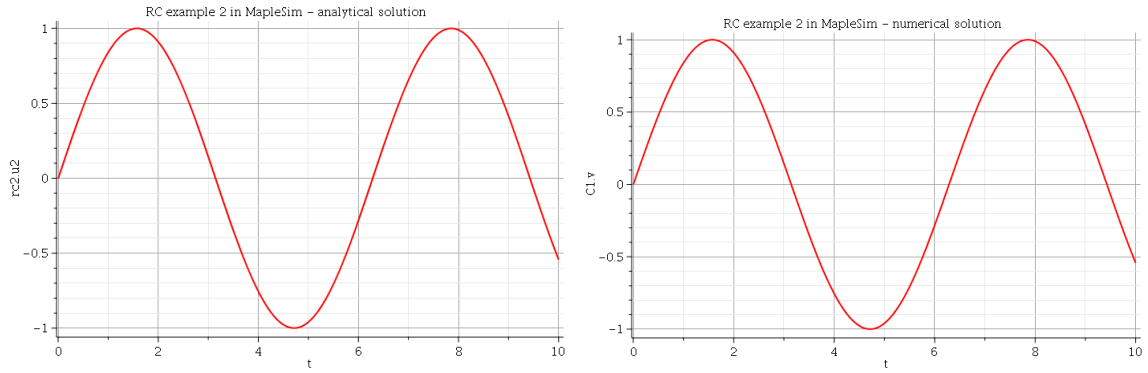
$$\begin{aligned} z' &= u, z(0) = 0 \\ u' &= z, u(0) = 1 \end{aligned} \tag{3.2}$$

Pre ďalšie spracovanie výsledkov MapleSim podporuje export dát pomocou kliknutia na ikonu *Export Data* v okne grafického vykreslenia výstupov simulácie [44].

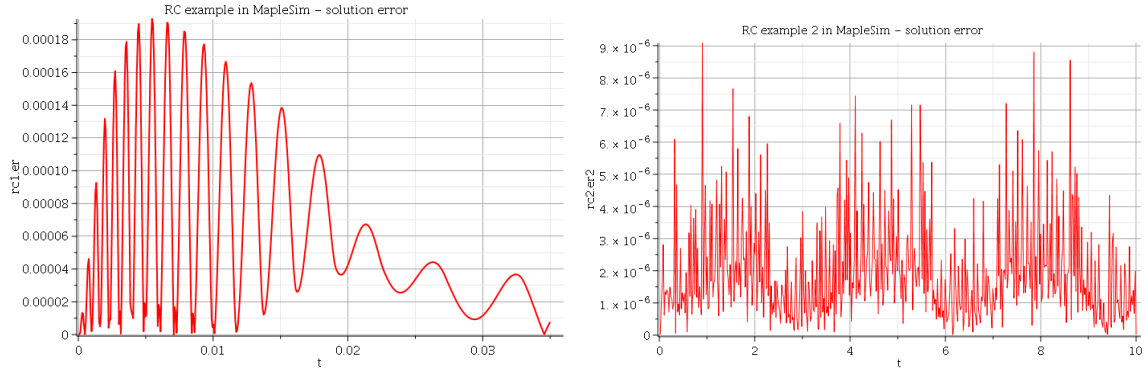


Obr. 3.20: Graf analytického a numerického (vpravo) riešenia RC obvodu s jednosmerným zdrojom napätia v MapleSim, získaného pomocou *RKF45* solvra a variabilného integračného kroku.

Rozdiel analytického a numerického riešenia som opäť vyjadrila i pomocou L2 chyby. Hodnotu tejto chyby je možné vyčísliť viacerými spôsobmi - v MATLAB-e alebo v Maple (tabuľka 3.3).



Obr. 3.21: Graf analytického a numerického (vpravo) riešenia RC obvodu so striedavým zdrojom napätia v MapleSim, získaného pomocou *RKF45* solvra a variabilného integračného kroku.



Obr. 3.22: Absolútne chyby analytického a numerického riešenia RC obvodov s jednosmerným a striedavým zdrojom napätia z obrázka 3.17, modelovaných v prostredí MapleSim.

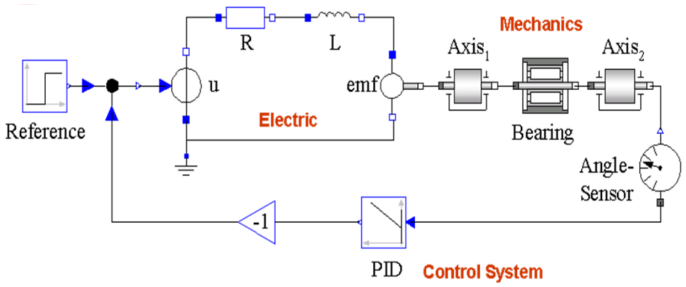
Tabuľka 3.3: L2 chyba analytického a numerického riešenia príkladov RC obvodu riešených v prostredí MapleSim.

Zdroj napätia	Solver	L2 chyba
konštantný	RKF45, variabilný krok	0.0022
striedavý	RKF45, variabilný krok	5.7715e-05

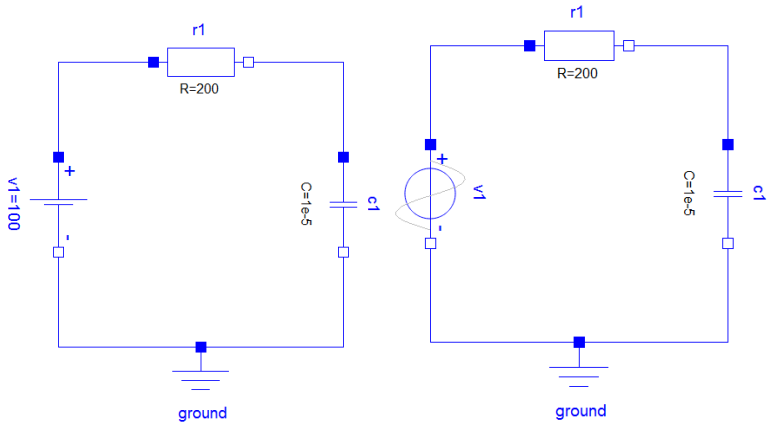
3.1.3 Dymola

Dymola (*Dynamic Modeling Laboratory*) je ďalším nástrojom, ktorý umožňuje pokročilé modelovanie a simulácie systémov z rozličných vedných disciplín. Tento produkt firmy *Dassault Systèmes* obohacuje mnohé odvetvia, podobne ako MATLAB či Maple. Kombinovanie súčastí viacerých sfér v rámci jedného modelu je i pri tomto nástroji dovolené (obrázok 3.23). Výsledkom sú potom presnejšie výsledky lepšie odpovedajúce realite [18]. Podporovanými platformami sú Windows a Linux.

Rovnako ako MapleSim, aj Dymola využíva benefity jazyka Modelica, ktorý je zároveň i štandardom, popisujúcim modely a ich súčasti, čím zvyšuje svoju flexibilitu a pripúšťa kombináciu komponentov z rozličných sfér, čím umožňuje tvorbu komplexnejších fyzikálnych systémov [50]. Disponuje komponentami vyvinutými tak, aby pokrývali viaceré sféry použitia. Modely z nich zostrojené sa stávajú viacúčelovými modelmi, ktoré je možné znova použiť [18].



Obr. 3.23: Ukážka modelu vytvoreného v nástroji Dymola, kombinujúceho prvky rozličných oblastí (mechanika, elektrické obvody, riadiace systémy a iné), využitím štandardu Modelica [50].



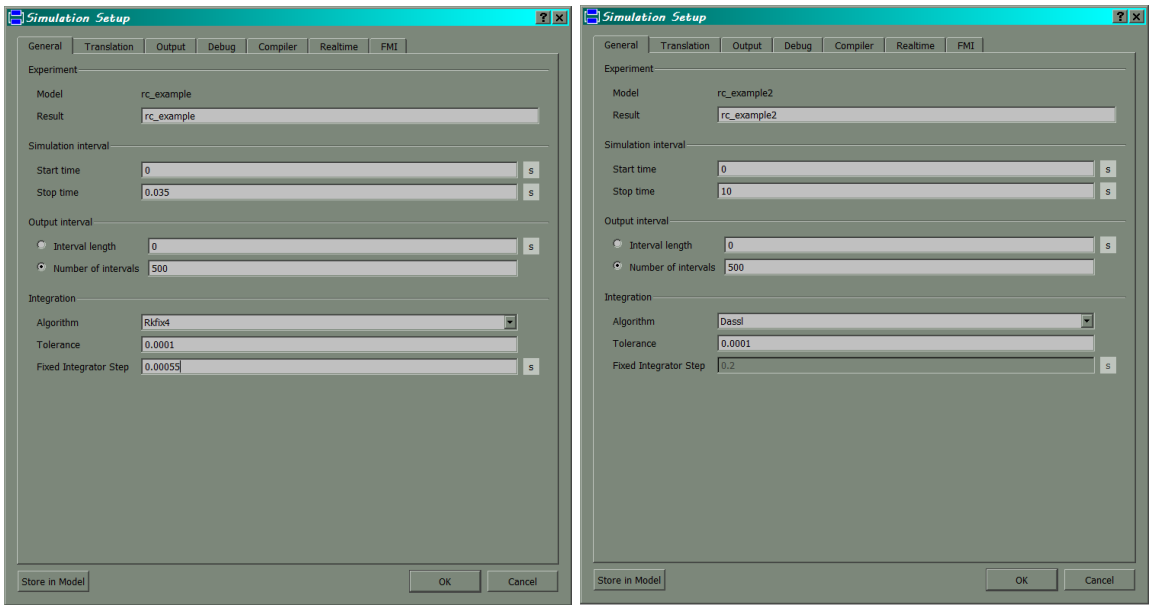
Obr. 3.24: Modely RC obvodov s jednosmerným a striedavým zdrojom napätia, vymodelované v prostredí Dymola.

Mnohé modely (štandard Modelica) reprezentujú systémy, ktoré dokážeme dostatočne popísať jedine pomocou väčšieho množstva parametrov, čo môže byť niekedy obtiažne. Dymola rieši tento problém kalibráciou modelu, odhadom daných parametrov, ktorý sa uskutočňuje na základe skutočne nameraných hodnôt parametrov. Vďaka tomu sa eliminuje riziko, že by sa výsledky simulácie pri použití daného modelu príliš odklonili od výsledkov reálnych komponentov [18].

Veľkú časť prostredia systému Dymola pokrýva pracovná plocha (editačné okno), ktorá je hlavným pôsobiskom tvorby modelov či ich vizualizácie (podobne ako u predchodzích nástrojoch). Situovaná je v strede okna. V prípade, že pracujeme na viacerých modeloch, využívame prepínanie sa medzi modelmi pomocou záložiek v ľavom hornom rohu tejto

plochy. Vrchnú časť tvorí hlavné menu so širokou škálou možností a nastavení, ikonky pre rýchly prístup k nim.

Vyhľadávanie súčiastok potrebných pre zostrojovanie modelov či príkladov (tzv. *Package browser*) alebo orientovanie sa v hierarchickom usporiadaní prvkov modelu (stromová štruktúra) (tzv. *Component browser*) je situované v ľavom segmente okna. K jednotlivým prvkom modelu je možné pristupovať i pomocou pravého tlačidla myši a kontextového menu – vybratím možnosti *Zobraziť komponent*, na ktorý klikneme v rámci modelu na pracovnej ploche. Obdobne pristupujeme i ku konkrétnym parametrom komponentu (zvolíme možnosť *Parametre*), ktoré sú pre prehľadnosť štrukturované v niekoľkých záložkách.



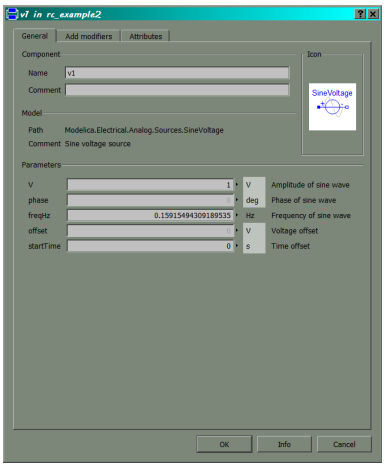
Obr. 3.25: Nastavenie parametrov simulácie pre modely RC obvodov s jednosmerným (vľavo) a striedavým zdrojom napätia (vpravo), vymodelované v prostredí Dymola.

Keď sme s vytvoreným modelom spokojní, môžeme prejsť k simulácii jeho správania sa. K simulácii sa môžeme dostať kliknutím na záložku *Simulácia* v pravom dolnom rohu prostredia. Tým prechádzame z editačného módu do módu simulovania. Sprístupnené sú podrobnejšie nastavenia simulácie v hlavnom menu v hornej časti (Simulácia). Mnohé zabudované príklady nástroja Dymola však obsahujú skripty s prednastavenými možnosťami simulácie. Takýto skript spustíme z hlavného menu v hornej časti prostredia cez *Príkazy -> Simulovať*. Prebehne simulácia, prípadne animácia daného modelu, vykreslenie výsledkov, priebehu simulácie v podobe grafu. Skript sa postará o preklad, beh simulácie, všetky potrebné nastavenia, i nastavenia zobrazenia výsledkov (graf). Dymola umožňuje vytvorenie vlastných skriptov [17].

Pre správny preklad modelu je nevyhnutné, aby operačný systém používateľa disponoval kompilátorom jazyka C. Bez neho nie je možné navrhnuté modely simulovať. Dymola neobsahuje tento kompilátor a je potrebné, aby si ho používateľ zabezpečil a správne nastavil pre funkčnú kooperáciu s nástrojom Dymola [19], čo môže byť netriviálne pre bežných používateľov počítačov alebo pre používateľov Dymola na Windows OS.

Kliknutím na *Simulácia* v hornom menu (v simulačnom móde) a výberom možnosti *Nastavenia* (*Set-up*) sa zobrazí okno pre podrobnejšie nastavenia simulácie i nastavenia kompilátora – v záložke *Kompilátor* (*Compiler*), kde zvolíme jeden z vopred nainštalovaných kompilátorov a následne overíme jeho funkčnosť kliknutím na *Verifikovať kompilátor*. Ak sme spokojní s nastaveniami simulácie, zmeny uložíme (tlačidlo OK) a v hlavnom okne spustíme simuláciu pomocou tlačidla ➤.

Pri simulácii nás neraz budú zaujímať konkrétne hodnoty premenných pre konkrétnu časť modelu. Kliknutím pravého tlačidla na komponent nášho záujmu a výberom možnosti *Zobraziť premenné* sa v ľavej časti prostredia Dymola vykreslí okno s pomenovaniami jednotlivých premenných a ich hodnotami [17].



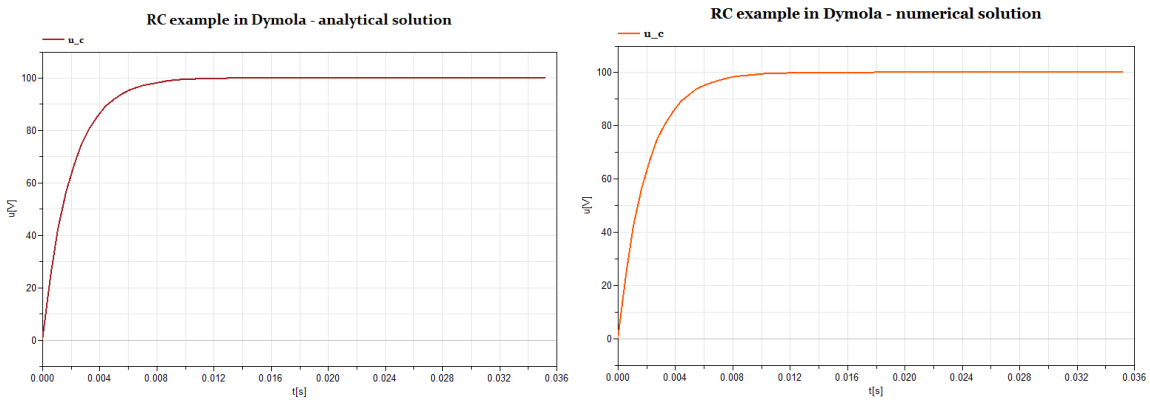
Obr. 3.26: Nastavenie parametrov pre striedavý zdroj napätia jednoduchého modelu RC obvodu v programe Dymola.

Ako konkrétne príklady modelov v Dymola opäť uvádzam modely RC obvodov z predchodzích podkapitol (pre Simulink a MapleSim) (obrázok 3.24). Pred simuláciou modelu s konštantným zdrojom napätia som zvolila pre potreby numerických výpočtov algoritmus postavený na Runge-Kutta metóde 4. rádu s fixným integračným krokom (*Rkfix4*) (obrázok 3.25 vľavo). Pre model so striedavým zdrojom napätia nebolo v súvislosti s nastaveniami simulácie možné postupovať rovnako, z dôvodov netriviálnych nastavení kompilátora a funkčnosti softvéru na operačnom systéme Windows 7, ktoré som pre jednoduchosť ponechala nezmenené. Použitý bol v tomto prípade algoritmus *Dassl* s variabilným integračným krokom (obrázok 3.25 vpravo).

Nastavenie parametrov pre striedavý zdroj napätia nebolo triviálne, hodnotu frekvencie sínusoidy bolo potrebné predvypočítať, čím nastáva závislosť výsledkov simulácie tohto modelu od presnosti tohto výpočtu (obrázok 3.26).

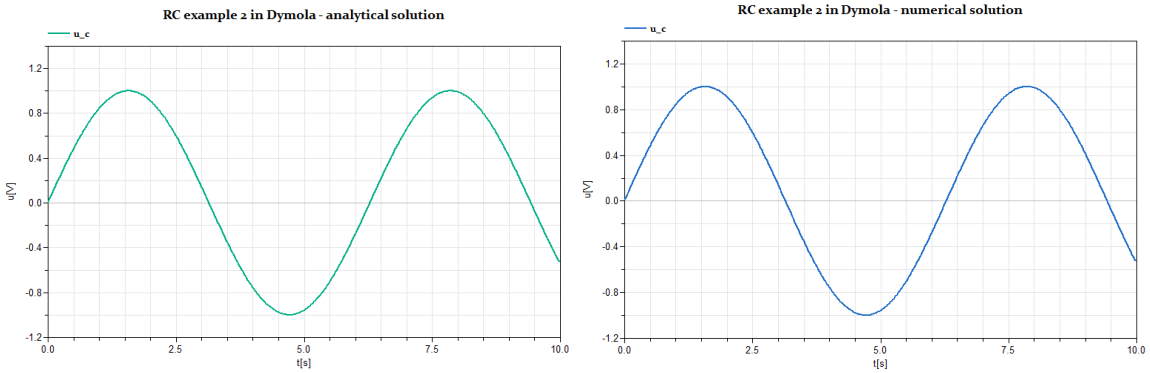
Grafy analytického a numerického riešenia pre príklad RC obvodu s jednosmerným zdrojom napätia môžeme porovnávať na obrázku 3.27, na ktorých je možné rozoznať prechodný dej pred ustálením riešenia. Pre obvod so striedavým zdrojom napätia sú časové priebehy riešení vizuálne vzájomne nerozoznateľné (obrázok 3.28), chyby výpočtu analytického a numerického riešenia však potvrdzujú (obrázok 3.29), že tieto grafy nie sú zhodné, no chyby dosahujú veľmi malých hodnôt, a preto je možné dosiahnuté výsledky riešení považovať

za prijateľné.



Obr. 3.27: Grafy časového priebehu napätia na kapacitore modelu RC obvodu, ktorého schéma je na obrázku 3.24 vľavo, predstavujúce analytické a numerické riešenie.

L2 chybu som vyjadrila pomocou nástroja MATLAB, v ktorom som načítala dáta reprezentujúce číselné hodnoty riešení a následne získala výslednú hodnotu chyby (tabuľka 3.4).



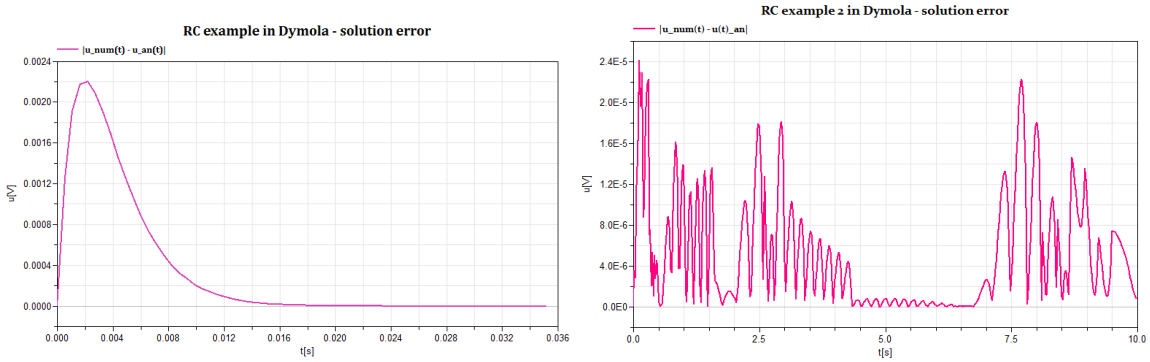
Obr. 3.28: Grafy časového priebehu napätia na kapacitore modelu RC obvodu, ktorého schéma je na obrázku 3.24 vpravo, predstavujúce analytické a numerické riešenie tohto modelu.

Číselné hodnoty riešení príkladov v Dymola je možné získať najjednoduchšie pomocou pravého tlačidla myši, ktorým označíme krivku v grafe a zvolíme možnosť *Kopírovať*. Hodnoty krivky sa skopírujú do dočasnej pamäte počítača, následkom čoho ich môžeme uložiť napríklad do textového súboru príkazom *Vložiť* a využiť pri ďalšom spracovaní či analýze.

Výpočet L2 chyby v MATLAB-e je možné pomocou predpripraveného skriptu, ktorého spustením prebehne výpočet tejto chyby a výsledok sa vypíše na štandardný výstup v konzoli MATLAB-u alebo priamo v príkazovom riadku po načítaní potrebných dát (číselné hodnoty časových krokov do premennej t, analytického riešenia do u_{an} a numerického do u):

```
error = [];
```

```
for i = 1:length(t)
    error(i) = abs(u(i) - u_an(i)); % abs is not necessary
end
norm(error) % this is L2 error
```



Obr. 3.29: Absolútne chyby analytického a numerického riešenia RC obvodov z obrázka 3.24, modelovaných v softvéri Dymola.

Tabuľka 3.4: L2 chyba analytického a numerického riešenia príkladov RC obvodu riešených v prostredí MapleSim.

Zdroj napätia	Solver	Integračný krok	L2 chyba
konštantný	Rkfix4	fixný	0.0057
striedavý	Dassl	variabilný	1.6550e-04

3.1.4 SPICE

SPICE (*Simulation Program with Integrated Circuit Emphasis*), podobne ako MATLAB, Maple či Dymola, je vhodným nástrojom pre simuláciu správania sa elektrických obvodov. Existujú viaceré obdoby tohto nástroja, voľne dostupné i pre komerčné účely (*PSPICE* [15], *XSPICE, HSPICE, CUSPICE, LTSPICE, TopSPICE* [11], *NGSPICE* [4]) so zaujímavou históriou. Pomenovanie SPICE preto vo všeobecnosti označuje určitý typ programov a zároveň aj dátový formát, ktorý tieto nástroje využívajú.

Používateľov open-source technológií zaujme voľne prístupný *NGSPICE*, spustiteľný i z webového rozhrania, kde predložením kódu (v skriptovacom jazyku typickom pre SPICE) reprezentujúceho obvod prebehne simulácia a jej výsledky sa náležite vykreslia do grafu [4]. Tieto výsledky sú čítané z textového alebo binárneho súboru vygenerovaného simulátorom. NGSPICE projekt však obsahuje viac funkcionalít.

Spice3f5 – hlavný predchodca NGSPICE, podporuje rozličné typy analýz (analýza prechodných dejov – transientná/časová analýza, AC (kmitočtová) a DC (jednosmerná) analýza a iné), z ktorých každá prináša užitočné informácie o danom obvode, rozličné typy komponentov tvoriacich obvod (zdroje, kapacitory, rezistory, nelineárne prvky ako diódy, rozličné typy vedenia), množstvo voliteľných parametrov a volieb vykreslenia, nenahraditeľných

pri modelovaní a simulácii elektrických obvodov, hlavne tých zložitejších [6]. Vo všeobecnosti však NGSPICE predstavuje aplikáciu použiteľnú v príkazovom riadku (konzolová aplikácia). Mnohé ďalšie obdoby SPICE obsahujú i grafické nástroje pre schématický návrh obvodov a zdokonalené vykresľovanie výsledkov do grafov.

Obvody súčasnosti sú tvorené kombináciou analógových (pracujúcich s analógovým, spojitým signálom) i digitálnych obvodov (zmiešané obvody), na čo bolo potrebné reagovať v podobe adekvátnych nástrojov. NGSPICE využíva analógové i digitálne simulačné algoritmy a tým sa stáva efektívnym simulačným prostriedkom podporujúcim simulácie v zmiešanom režime (*mixed-mode simulation*), získavajúcim výhody z oboch typov algoritmov [49].

Pri analógovej simulácii NGSPICE využíva *Kirchhoffove zákony* produkujúce množstvo rovníc a overuje, či sa riešenie rovníc v čase približuje ku stabilným hodnotám a či numerické aproximácie integrácií dosahujú požadovanú presnosť. Rovnice sú reprezentované maticou a výpočet riešenia sa uskutočňuje pre každý časový krok, v závislosti od zvolenej dĺžky simulácie a veľkosti časového kroku. Takéto výpočty sú v porovnaní s výpočtami digitálnych simulačných algoritmov pomalšie [49].

U digitálnej simulácii sú aplikované odlišné princípy. Tá nevyžaduje riešenie rovníc vyplývajúcich z Kirchhoffových zákonov. Jej podstata je postavená na tzv. udalostiach alebo zmenách v logickom stave daného uzla, ktoré, ak nastanú, distribuujú sa následne k pripojeným prvkom obvodu, a teda stačí, aby simulátor preveril iba tie prvky obvodu, na ktoré udalosť priamo pôsobí, nemusí pri každej iterácii vykonať výpočet pre celý obvod. Tým sa vo veľkej miere skracuje čas výpočtu [49].

Jadrom práce nielen s NGSPICE je tzv. *netlist*. Ide o zdrojový súbor obsahujúci zoznam komponentov obvodu, popisujúci ich vzájomné vzťahy, hodnoty veličín reprezentujúce dané komponenty, prípadne ďalšie špecifikácie obvodu. Predstavuje adekvátnu náhradu za grafické prevedenie schém obvodov. Netlist zostrojujeme v špecifickom popisnom jazyku, vo formáte SPICE, ktorý má svoje presné pravidlá, bližšie rozobraté v [49]. Bol navrhnutý tak, aby sa s ním čo najjednoduchšie pracovalo, a to vďaka zrozumiteľnej syntaxe. V súvislosti s niektorými obdobami SPICE, nie je vždy nutné ovládať syntax netlistu, ak nástroj disponuje grafickým rozhraním pre zostrojenie návrhu obvodu, no je to výhodou. Každý takýto nástroj nakoniec i tak preloží návrh do netlistu a následne uskutoční simuláciu [10]. Netlist je kompatibilný i s programom *Micro-Cap*, v ktorom je možné ho vytvoriť a následne činnosť obvodu i simulovať [53].

Pri písaní netlistu je vhodné dodržiavať predpísaný formát. Netlist nerozoznáva malé a veľké písmená. Prvý riadok je venovaný názvu obvodu (simulácie). Každý ďalší riadok je vyhradený pre jeden prvok obvodu, prípadne pre príkazy súvisiace so simuláciou. Prvky obvodu sú reprezentované počiatočným písmenom (napríklad `R` pre rezistor, `V` pre zdroj napätia, `C` pre kapacitor) a ďalšími znakmi, pomenúvajúcimi daný prvok. Pre zdroj napätia je potrebné konkretizovať typ zdroja (napríklad `dc`). Uzly obvodu sa doporučuje očíslovať, na základe čoho sa v netliste definujú prepojenia medzi jednotlivými prvkami obvodu. Uzemnenie (ground) sa vždy označuje nulou. Číselné hodnoty ukončujúce riadky predstavujú hodnoty parametrov. U NGSPICE je nutné, aby používateľ uviedol aj typ analýzy (`.dc, .ac` alebo `.tran`) a veličiny, ktorých hodnoty chce vypísať (príkaz `.print`) alebo vykresliť (príkaz `.plot`) [49].

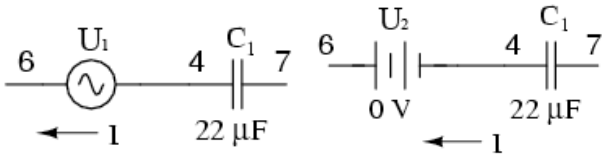
V prípade, že máme k dispozícii netlist popisujúci požadovaný obvod (pomenovaný napríklad `obvod.cir`), simuláciu pomocou SPICE spustíme v príkazovom riadku:

```
$ spice < obvod.cir
```

Pre NGSPICE použijeme príkaz `ngspice`. Spustením simulácie sa vygeneruje textový výstup s výslednými hodnotami veličín, v závislosti od konkrétneho obvodu a zvoleného typu analýzy. Výstup je možné uložiť do súboru. Niektoré obdoby SPICE automaticky generujú a zobrazujú grafické výstupy v podobe grafov, v závislosti od špecifikácií vykreslenia výsledkov v netliste. Výstupné hodnoty uložené do textového súboru môžeme následne upravovať, prezerať či spracovať.

Meranie napätia prechádzajúceho cez vybraný komponent (napríklad kapacitor zapojený medzi uzlami 4 a 7, obrázok 3.30) obvodu uskutočníme pridaním nasledovných riadkov do netlistu:

```
c1 4 7 22u
.print ac v(4,7)
```



Obr. 3.30: Schéma zapojenia zdrojov napätia U_1, U_2 a kapacitora $C1$ v sérii a označenie jednotlivých uzlov číslami (zľava uzly 6, 4 a 7) pre potreby implementácie merania prúdu pretekajúceho cez kapacitor v programe SPICE využitím zdrojového súboru netlist, v ktorom správne uvedenie uzlov a tým správne prepojenie komponentov obvodu zohráva dôležitú rolu pre potreby získania správnych výsledkov simulácie daného obvodu [1].

Meranie prúdu pretekajúceho cez komponent je o niečo menej triviálne (príkladom je opäť kapacitor z obrázka 3.30), pričom takéto meranie musíme vždy špecifikovať vo vzťahu ku konkrétnemu zdroju napätia, napríklad (obrázok 3.30 vľavo):

```
u1 6 4 ac 1 sin
.print ac i(u1)
```

V prípade, že referenčný zdroj napätia nie je v obvode k dispozícii uvedeným spôsobom, pridáme zdroj napätia s nulovým napätím do obvodu na potrebné miesto (obrázok 3.30 vpravo, obrázok 3.31), čím umožníme, aby v SPICE prebehlo meranie prúdu prechádzajúceho cez nami zvolený komponent korektne, výsledky získame v množine kladných čísel a zároveň tým správanie sa obvodu nijak nepozmeníme [1]. Do netlistu pridáme nasledovné:

```
c1 4 7 22u
u2 6 4 dc 0
.print ac i(u2)
```

Využitím NGSPICE som vytvorila netlist, popisujúci elektrický obvod, ktorého schéma je na obrázku 3.31. Počiatočnú podmienku napätia na kapacitore som stanovila na 0. Na začiatku simulácie je kapacitor vybitý, s kapacitou $0,001F$. Odpory oboch rezistorov som nastavila na 100Ω, zdroj napätia na $5V$. Do obvodu bolo potrebné pridať ďalší zdroj napätia s nulovým napätím, pre korektný výpočet. Jednotlivé uzly som očíslovala hodnotami 0 až 3. Zvolila som transientnú analýzu a čas od $0s$ do $0,45s$ s krokom $0,01s$. Netlist vyzerá nasledovne:

```
My RC example
v1 1 0 dc 5
v2 3 0 dc 0
r1 1 2 100
r3 2 0 100
c1 2 3 0.001 ic=0
.tran 0.03 0.45 uic
.plot tran v(2,3)
.end
```

Po spustení simulácie začne pretekať obvodom jednosmerný elektrický prúd a na kapacitore sa začne zvyšovať napätie až po ustálený stav, kedy hodnota napätia dosiahne hodnotu $2,5V$. Spustením aplikácie v konzoli pomocou `ngspice < example.cir` sa vygeneroval nasledujúci výstup:

```
Circuit: my rc example

Doing analysis at TEMP = 27.000000 and TNOM = 27.000000

Initial Transient Solution
--------------------------
Node                                   Voltage
----                                   -------
1                                            0
3                                            0
2                                            0
v2#branch                                    0
v1#branch                                    0

No. of Data Rows : 63
--------------------------------------------------------------------------------
                            my rc example
                Transient Analysis Wed May  3 09:44:37  2017

Legend:  + = v(2)-v(3)
--------------------------------------------------------------------------------
 time       v(2)-v(3 0.00e+00         1.00e+00         2.00e+00         3.00
----------------------|----------------|----------------|----------------|
 0.000e+00  0.000e+00 +                .                .                .
 3.000e-02  1.130e+00 .                . +              .                .
 6.000e-02  1.747e+00 .                .           +    .                .
 9.000e-02  2.087e+00 .                .                .+               .
 1.200e-01  2.274e+00 .                .                .   +            .
 1.500e-01  2.376e+00 .                .                .     +          .
 1.800e-01  2.432e+00 .                .                .      +         .
 2.100e-01  2.463e+00 .                .                .      +         .
 2.400e-01  2.480e+00 .                .                .       +        .
 2.700e-01  2.489e+00 .                .                .       +        .
 3.000e-01  2.494e+00 .                .                .       +        .
```

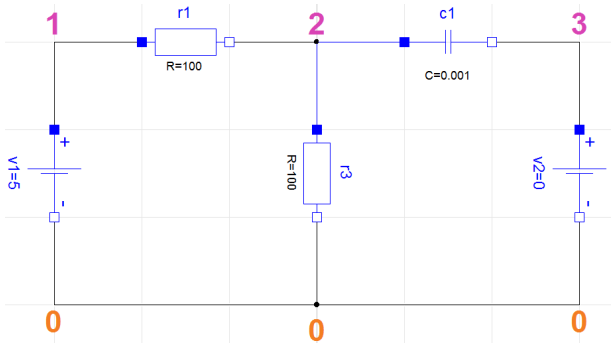
```
 3.300e-01  2.497e+00 .                .                .       +        .
 3.600e-01  2.498e+00 .                .                .       +        .
 3.900e-01  2.499e+00 .                .                .       +        .
 4.200e-01  2.499e+00 .                .                .       +        .
 4.500e-01  2.500e+00 .                .                .       +        .
----------------------|----------------|----------------|----------------|
 time       v(2)-v(3 0.00e+00          1.00e+00         2.00e+00         3.00
```

Z výstupu jasne rozpoznať krivku reprezentujúcu vývoj napätia na kapacitore v čase, dosahujúcu svoju maximálnu hodnotu v čase $0,43s$ s očakávanou hodnotou $2,5V$. Výstup je možné tiež uložiť do súboru a hodnoty ďalej spracovať alebo ich zobraziť pomocou grafu v inom nástroji (napríklad MATLAB).

V prípade, že nás nezaujíma vývoj napätia v závislosti od času, ale iba výsledná hodnota, zvolíme DC analýzu. Transientná analýza je rozšírením tejto analýzy. V netliste uvedenom vyššie postačí nahradiť riadky charakterizujúce analýzu a vykreslenie výsledkov za nasledovné:

```
.dc v1 5 5 1
.print dc v(2,3)
```

Pri výpise som definovala typ analýzy a uzly, medzi ktorými som chcela merať napätie. Schému obvodu som navrhla tak, že kapacitor som umiestnila medzi uzly 2 a 3.

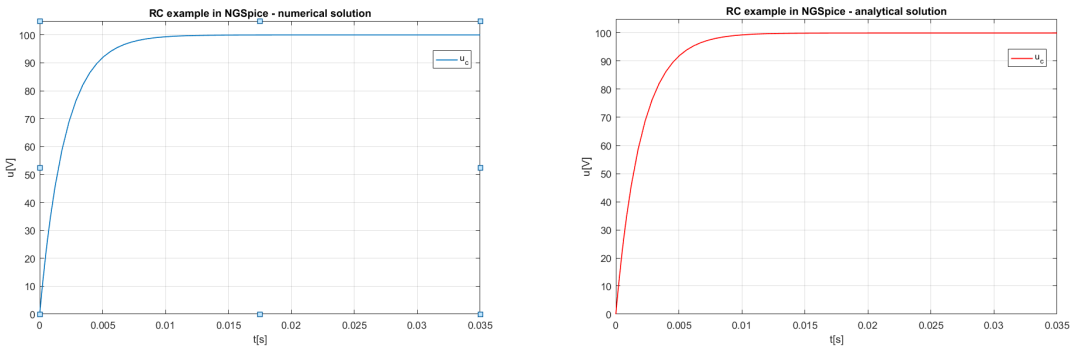


Obr. 3.31: Schéma jednoduchého obvodu s dvoma rezistormi, zdrojom konštantného napätia a kapacitorom, spolu s očíslovanými uzlami hodnotami 0 až 3. Do obvodu bolo potrebné pridať ďalší zdroj napätia s nulovým napätím, pre korektný výpočet pomocou nástroja NGSPICE [1].

Pre porovnanie som v tomto nástroji odsimulovala i RC obvody z predchádzajúcich podkapitol (ktorých schémy sú na obrázku 3.24). Pre numerické riešenie obvodu s jednosmerným zdrojom napätia som zostavila nasledujúci netlist, pričom som zvolila transientnú analýzu, rovnaké hodnoty parametrov prvkov obvodu ako pri predchádzajúcich nástrojoch:

```
Rc example
v1 1 0 dc 100
r1 1 2 200
c1 2 0 1e-5 ic=0
.tran 0.00055 0.035 uic
.print tran v(2)
.end
```

Pre výpis hodnôt napätia medzi uzlami 2 a 0 v NGSPICE netliste postačuje uviesť iba uzol 2 (`.print tran v(2)`). Tieto hodnoty som po úspešnom vykonaní simulácie uložila do súboru a následne spracovala v prostredí MATLAB (obrázok 3.32).



Obr. 3.32: Grafy numerického a analytického riešenia RC obvodu so schémou na obrázku 3.24 vľavo, s jednosmerným zdrojom napätia, odsimulovaného v nástroji NGSpice.

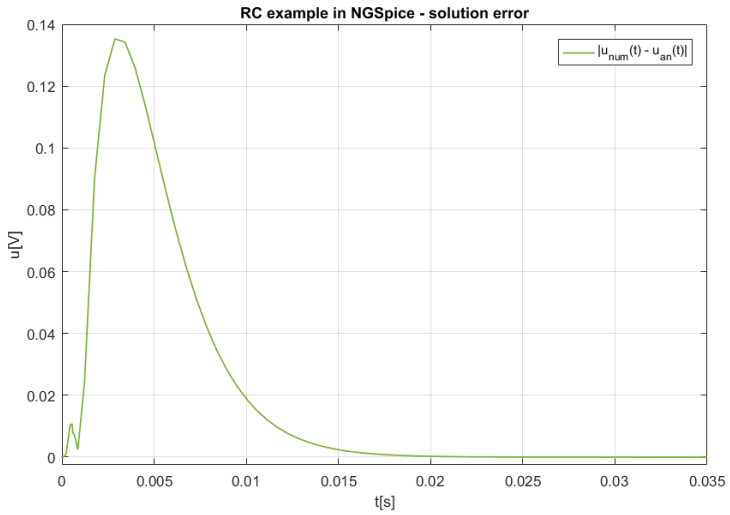
Pre implementáciu analytického riešenia v NGSpice bolo potrebné v netliste definovať funkciu (pomenovanú $analytic(x)$) pre napätie na kapacitore $C1$, konštanty nevyhnutné pre výpočet ($tmax$, dt) a využitím cyklu `while` postupne vypísať hodnoty pre všetky časové úseky. NGSpice disponuje podporou pre všetky bežné matematické operácie a prácu s cyklami:

```
Rc example - analytic
.control
define analytic(x) 100-100*exp((-1/(200*1e-5))*x)
let tmax=0.035
let dt=0.00055
let t = 0
while t <= tmax
*  print analytic(t)
  print t
  let t = t + dt
end
.endc
.end
```

Tento spôsob získania analytického riešenia je prijateľný, no nepostačujúci pre porovnanie analytického a numerického riešenia v tomto nástroji, pretože týmto spôsobom získame iný počet krokov, než pri numerickom riešení, a to i po zachovaní veľkosti integračného kroku a časového intervalu. Pre porovnanie analytického a numerického riešenia potrebujeme hodnoty riešenia pre tie isté časové úseky, v opačnom prípade nie je možné tieto hodnoty priamo porovnávať. Dopočítavaním hodnôt na základe hodnôt pre iné (blízke) časové úseky môže dôjsť k nepresnostiam. Preto som hodnoty analytického riešenia dopočítala postupne pre časy získané z numerického riešenia, čím som obdržala hodnoty riešení pre tie isté časové úseky a mohla ich priamo porovnávať. Pre získanie hodnoty analytického riešenia napríklad v čase $t = 0.035s$ bol použitý nasledovný kód:

```
Rc example - analytic
.control
define analytic(t) 100-100*exp((-1/(200*1e-5))*t)
print analytic(3.500000e-02)
.endc
.end
```

Výsledkom porovnania analytického a numerického riešenia RC obvodu s jednosmerným zdrojom napätia je graf na obrázku 3.33.

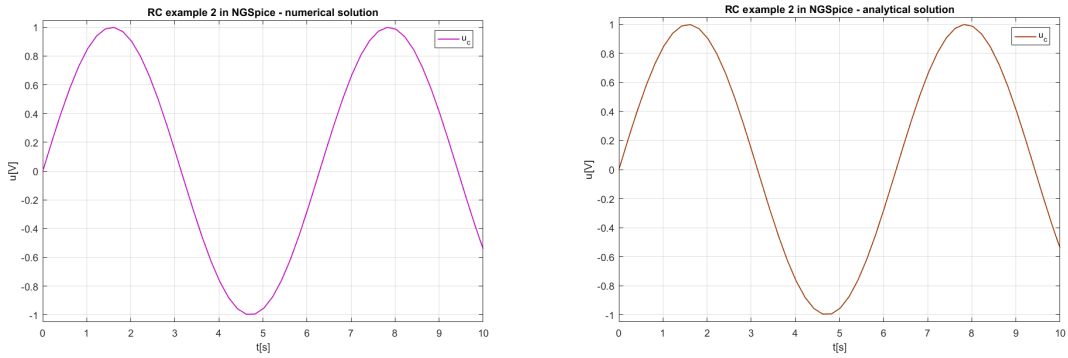


Obr. 3.33: Absolútna chyba numerického a analytického riešenia RC obvodu, ktorého schéma je na obrázku 3.24 vľavo, s jednosmerným zdrojom napätia, odsimulovaného v nástroji NGSpice.

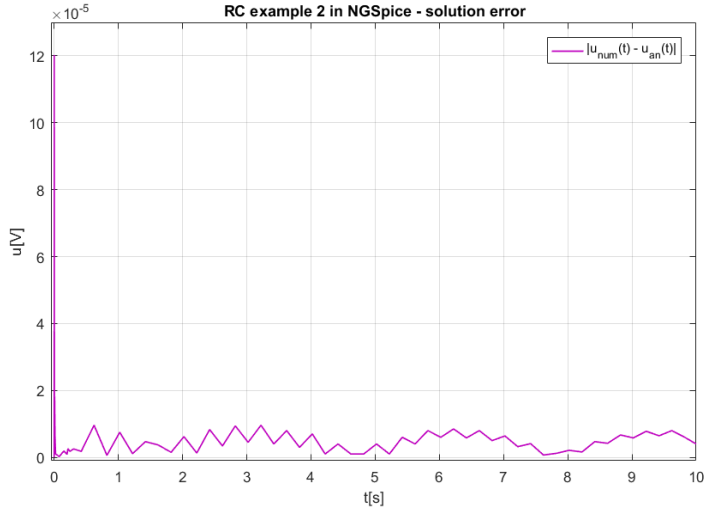
Pre obvod so striedavým zdrojom napätia a jeho numerické riešenie som upravila existujúci netlist do nasledovnej podoby s pomocou [49]:

```
Rc example 2
v1 1 0 dc sin(0 1 0.15915494309189535 0 0)
r1 1 2 200
c1 2 0 1e-5 ic=0
.tran 0.2 10 uic
.print tran v(2)
.end
```

Pri tejto úprave som zmenila i typ zdroja, *tmax* a integračný krok, čím som pre simuláciu obvodu nastavila rovnaké podmienky ako u simulačných nástrojoch z predchádzajúcich podkapitol. Nenulové hodnoty pri definícii zdroja napätia $v1$ predstavujú amplitúdu a vyčíslenú frekvenciu $\frac{1}{2\pi}$. Pri analytickom riešení som postupovala obdobne ako u obvodu s jednosmerným zdrojom napätia. Výsledky riešení predstavujú grafy na obrázkoch 3.34 a 3.35. Pre oba príklady RC obvodov bola vyčíslena i L2 chyba (tabuľka 3.5).



Obr. 3.34: Grafy numerického a analytického riešenia RC obvodu, ktorého schéma je na obrázku 3.24 vpravo, so striedavým zdrojom napätia, odsimulovaného v nástroji NGSpice.



Obr. 3.35: Absolútna chyba numerického a analytického riešenia RC obvodu, ktorého schéma je na obrázku 3.24 vpravo, so striedavým zdrojom napätia, odsimulovaného v nástroji NGSpice.

Algoritmy použité pri výpočtoch v NGSpice úzko súvisia s použitým typom analýzy obvodu, preto nie je možné triviálnym spôsobom meniť metódy výpočtu a porovnávať ich tak, ako to bolo možné u predchádzajúcich nástrojoch, pri tom istom type analýzy. Na to sa vzťahuje i relatívne vyššia chyba výpočtu pre príklad RC obvodu s jednosmerným zdrojom napätia: NGSpice využíva vlastné optimalizácie výpočtov či úpravu integračného kroku, do ktorej nie je jednoduché zasahovať. Metódy výpočtov sú však v NGSpice vytvorené tak, aby pre daný typ analýzy priniesli uspokojivé riešenie pre čo najširšie spektrum problémov.

Tabuľka 3.5: L2 chyba analytického a numerického riešenia príkladov RC obvodu, riešených v prostredí NGSpice.

Zdroj napätia	Počet krokov	L2 chyba
konštantný	76	0.3523
striedavý	66	2.2205e-04

3.1.5 Ďalšie simulačné nástroje

V oblasti navrhovania a simulácií obvodov v súčasnosti existujú široké možnosti, dvíhajúce sa priamoúmerne s rastúcim výpočtovým výkonom počítačov a nárokov z iných oblastí vedy a techniky. Z toho dôvodu sa vo svete objavujú nové nástroje, využívajúce a kombinujúce rôzne prístupy, zatiaľčo tie existujúce prichádzajú so snahou o vylepšenie. Obdobne je tomu i na univerzitách, kde pod iniciatívou univerzitných pracovníkov i študentov vznikajú alternatívy simulačných programov, ideálne pre výuku i pre experimenty, ktorých výstupmi sú neraz zaujímavé výsledky ako podnety pre ďalší výskum.

3.1.5.1 TKSL/FOS

TKSL je názov pre simulačný systém, riešiaci úlohy, reprezentované sústavami diferenciálnych rovníc 1. rádu (výskyt výrazov s prvou deriváciou). Diferenciálne rovnice sú riešené numericky, využitím modernej metódy Taylorovho radu [41] [7], ktorá poskutuje "extrémne presné riešenie" v relatívne krátkom čase, s vysokou stabilitou. Systém je schopný riešiť rovnice i vyššieho rádu, po ich úprave na systém rovníc 1. rádu (napríklad pomocou metódy znižovania rádu derivácie [7] [38]). Vysoká presnosť sa dosahuje i automatickou voľbou rádu metódy (počet členov Taylorovho radu), preto sa vzhľadom na jednotlivé kroky výpočtu môže použitý rád metódy lísiť, ale to "pozitívne ovplyvňuje stabilitu a rýchlosť výpočtu".

Existuje viacero variánt tohto nástroja, napríklad TKSL/386, TKSL/C a ďalšie. Výhodou je, že pred použitím nie je potrebná jeho inštalácia. Syntax zdrojového kódu, popisujúceho daný systém rovníc, je zrozumiteľná, uspôsobená pre rýchle použitie. Po deklarácii premenných nasleduje definícia konštánt, vrátane definovania "maximálnej hodnoty analyzovaného časového priebehu" *tmax* a presnosť výpočtu *eps*.

Jadro programu tvorí sústava rovníc, zadávaná spolu s počiatočnými podmienkami pre jednotlivé diferenciálne rovnice. Taktiež je možné definovať ďalšie nastavenia, napríklad pre vykresľovanie výsledkov do grafu. Tento nástroj sa pre svoje jednoduché použitie hojne využíva aj pri výuke.

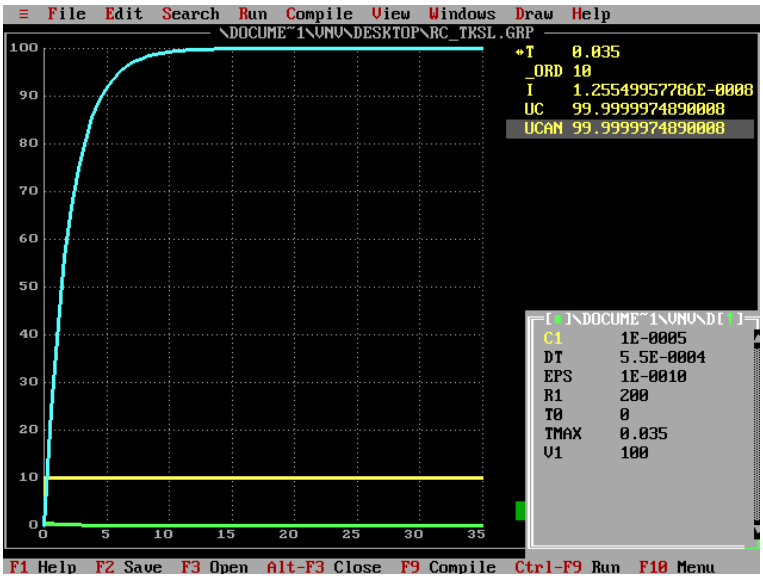
TKSL je pre svoje vlastnosti vhodným prostriedkom pre simuláciu elektrických obvodov. Vytvorenie zdrojového súboru si vyžaduje znalosť odvodenia rovníc popisujúcich elektrický obvod, o čom pojednáva kapitola 2.

Zdrojový súbor jednoduchého RC obvodu z predchodzích podkapitol (konštantný zdroj napätia) pre TKSL vyzerá nasledovne:

```
var i, uc, ucan;
const dt=0.00055, tmax=0.035, eps=1e-10, c1=1e-5, r1=200, v1=100;

system
i = 1/r1*(v1-uc);
```

```
uc' = 1/c1*(i) &0;
ucan = v1-v1*exp((-1/(r1*c1))*t);
sysend.
```

Výstupom simulácie obvodu je graf na obrázku 3.36 . TKSL umožňuje zobraziť výsledok simulácie i formou tabuľky hodnôt. Rád metódy pri výpočtoch pre príklad RC obvodu rýchlo dosiahol hodnotu 10 a ďalej sa nemenil.



Obr. 3.36: Graf numerického a analytického riešenia RC obvodu, ktorého schému znázorňuje obrázok 3.24 vľavo (jednosmerný zdroj napätia), odsimulovaného v nástroji TKSL. V grafe je znázornený i rád metódy a hodnoty veľkosti prúdu pretekajúceho obvodom.

Pre výpočet absolútnej alebo L2 chyby riešenia sú však vhodnejšie iné nástroje než TKSL, napríklad relatívne nový nástroj *FOS* [2], rozširujúci TKSL o ďalšie možnosti, podporujúci použitie podmienok, vytváranie animácií a iné. Tento nástroj bol využitý pri simulácii návrhov CMOS invertoru, CMOS NAND a CMOS NOR obvodov, ktoré bližšie predstavuje kapitola 4.

Zaujímavým nástrojom je i projekt študenta Lukáša Kraicingera - *Simulátor logických obvodů*, obsahujúci logické prvky, multiplexory, hodiny a iné prvky, z ktorých je možné skladať obvody a následne simulovať ich správanie sa [43].

Kapitola 4

Modely CMOS VLSI obvodov

Hojné rozšírenie digitálnych obvodov v súčasnoti umožnilo konštrukciu mikroprocesorov, výkonnejších počítačov a ďalších elektronických zariadení, ako ich poznáme. V porovnaní s analógovými obvodmi sú pre ne charakteristické iba dve úrovne napätia (signálu), čo odpovedá binárnej číselnej sústave (hodnoty 1, 0 - dva možné stavy), využívanej v číslicových zariadeniach, počítačoch. Signál prúdiaci týmito obvodmi sa mení skokovo.

Logické obvody podporujú vykonávanie bežných logických operácií ako NOT, NAND, NOR, XOR a iné v Booleovej algebre [52]. Tok signálu sa v týchto obvodoch realizuje jedným smerom (vstup -> výstup).

CMOS obvody (*Complementary Metal–Oxide–Semiconductor)* využívajú pre vykonávanie logických operácií tzv. *MOSFET* tranzistory (*Metal Oxide Semiconductor Field Effect Transistors*), využívajúce komplementárne dvojice polovodičov typu p a typu n, prvý krát popísané v roku 1963. Typickými črtami CMOS obvodov sú vysoká odolnosť voči tzv. šumu, nízka spotreba energie, relatívne vysoká rýchlosť vykonávania operácií [48].

Úspešná snaha neustále zmenšovať priestor na plochách čipov, zaberajúci tranzistormi a zjednodušovanie výrobných procesov viedlo k zvyšujúcemu sa počtu tranzistorov v mikroprocesoroch (zvyšovanie stupňa integrácie). Podľa stupňa integrácie rozoznávame [48]:

- *SSI* obvody (*Small-Scale Integration*, obvody malej integrácie)
- *MSI* obvody (*Middle-Scale Integration*, obvody strednej integrácie)
- *LSI* obvody (*Large-Scale Integration*, obvody veľkej integrácie)
- *VLSI* obvody (*Very Large-Scale Integration*, obvody veľmi veľkej integrácie)
- *ULSI* obvody (*Ultra Large-Scale Integration*, obvody väčšej integrácie než VLSI obvody)

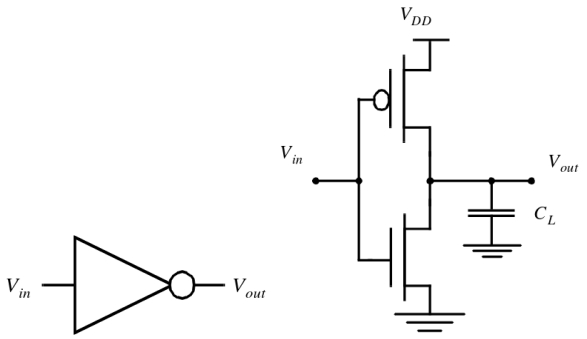
Počet logických obvodov SSI obvodu dosahoval hodnotu menšiu než 10, s takmer pol tuctom tranzistorov na jeden logický obvod, zatiaľčo u VLSI sú to milióny až miliardy tranzistorov na jednom čipe.

Súčasťou tejto práce je návrh modelov CMOS invertoru, CMOS NAND a CMOS NOR. Návrhy vychádzajú z [40] a [48], kde sú klasické tranzistory nahradené v obvode rezistormi a kapacitormi, pre zabezpečenie adekvátnej funkcionality týchto logických prvkov.

4.1 Návrh modelu CMOS invertoru

Invertor tvorí najzákladnejší (kombinačný) logický obvod (hradlo), v súvislosti so stavbou číslicových systémov. Vykonáva logickú operácu negácie nad jedným operandom (len jedna vstupná hodnota) známu z Booleovej algebry [52]. Výstupom invertoru je vždy opačná logická hodnota v porovnaní so vstupnou hodnotou.

Pojem CMOS invertoru (alebo NOT) je významným z hľadiska konštrukcie zložitejších logických obvodov, mikroprocesorov (obrázok 4.1). Pri návrhu je potrebné zohľadňovať množstvo parametrov, špecifikácií, ktorým má výsledný návrh odpovedať. CMOS invertor je možné analyzovať z mnohých hľadísk, ako napríklad výkon, energetická účinnosť a podobne [9].



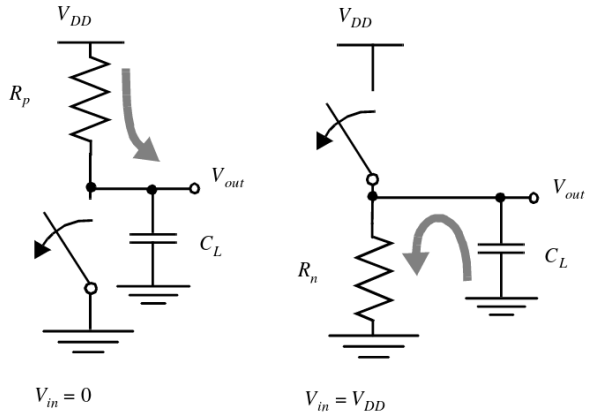
Obr. 4.1: Schématická značka a schéma statického CMOS invertoru spolu s parazitným kapacitorom C_L [9].

Invertor môže byť realizovaný použitím MOS tranzistorov (jedného PMOS alebo jedného NMOS tranzistoru spolu s jedným rezistorom). Iný spôsob realizácie je kombinácia oboch spomenutých tranzistorov, čo vedie na CMOS logiku a znižuje energetické nároky. Pomocou dvoch tranzistorov CMOS invertoru sa uskutočňuje prepínanie medzi vysokým (nekonečným) odporom (vypnutie, alebo logická 0) a odporom konečnej (malej) hodnoty (logická 1). Keď vstupné napätie V_{in} je vysoké a odpovedajúce napájaciemu napätiu V_{dd}, tak NMOS tranzistor je zapnutý, zatiaľčo naopak PMOS je vypnutý. To umožní existenciu priamej cesty medzi V_{out} a uzemnením, čo vedie k nulovému výslednému napätiu ($0V$) (obrázok 4.2 vpravo). Keď je vstupné napätie naopak nízke alebo nulové, zapnutý je PMOS tranzistor a NMOS je vypnutý a výstupom je vysoké napätie (obrázok 4.2 vľavo). Toto správanie sa odpovedá logickej operácii NOT (tabuľka 4.1).

Tabuľka 4.1: Logické hodnoty odpovedajúce vstupnému V_{in} a výstupnému napätiu V_{out} pri využití CMOS invertoru, ekvivalentné logickej operácii NOT.

Vstup	Výstup
1	0
0	1

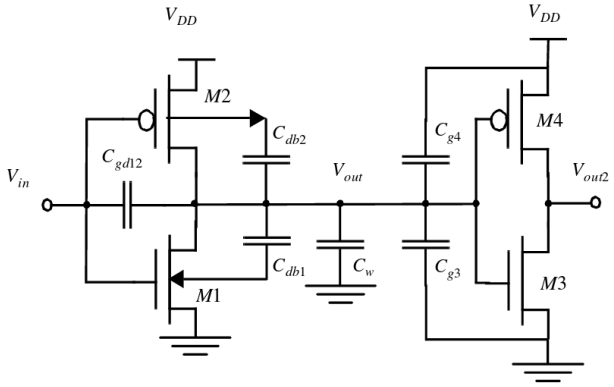
V súvislosti s dynamickým správaním sa CMOS invertoru a analýzou prechodných dejov je možné poznamenať, že čas, za ktorý je možné získať odpoveď (výstup) invertoru, závisí



Obr. 4.2: Modely prepínačov pre dynamické správanie sa CMOS invertoru, vľavo model pre vysoké a vpravo pre nízke vstupné napätie V_{in} [9].

na tom, ako rýchlo sa nabije/vybije kapacitor C_L pomocou rezistora R_p. Vo všeobecnosti požadujeme rýchle odpovede, čo je možné dosiahnuť nízkou kapacitou C_L alebo odporom R_p. Podľa možnosti čo najnižšia kapacita C_L je významná pri realizácii vysoko výkonných CMOS obvodov [9].

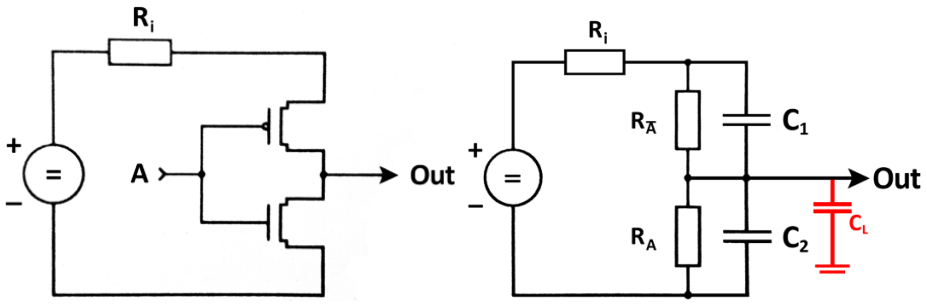
V rámci jedného či viacerých prepojených CMOS invertorov je vhodné podotknúť, že ich vlastnosti môžeme v praxi reprezentovať viacerými parazitnými kapacitami, ovplyvňujúcimi prechodné deje (obrázok 4.3). Analýza správania sa takého obvodu môže byť náročná, preto sa obvod zjednodušuje a parazitné vlastnosti sa zjednocujú pod jeden kapacitor, už spomenutý C_L, ktorý sa v obvode umiestňuje medzi výstup napätia V_{out} daného invertoru a uzemnenie (ground).



Obr. 4.3: Parazitné kapacitory pri zapojení páru CMOS invertoru, vplývajúce na správanie sa obvodu. Zjednodušenie zapojenia môžeme uskutočniť nahradením kapacitorov s parazitnými kapacitami za jeden C_L, pričom kapacitor sa vkladá medzi výstup napätia V_{out} daného invertoru a uzemnenie [9].

Návrh CMOS invertoru je realizovaný nahradením každého tranzistora párom rezistor-kapacitor. Na výstup som pridala jeden kapacitor C_L s parazitnou kapacitou so zámerom

priblíženia sa správaniu reálnej súčiastky (obrázok 4.4).



Obr. 4.4: Model CMOS invertoru, ktorý miesto dvoch tranzistorov typu p a n (vľavo) pozostáva z dvoch párov rezistor-kapacitor (vpravo). Na výstup je umiestnený i kapacitor C_L reprezentujúci parazitné javy, vyskytujúce sa v reálnych obvodoch.

Parazitné kapacity v elektrických obvodoch nie sú vo väčšine prípadov elementom, ktorý by bol pre správanie sa obvodu žiadúci. Reálne prvky obvodov však disponujú viacerými vlastnosťami (napríklad vlastná indukčnosť u rezistora), ktoré môžu vyvolávať neželané (parazitné) javy, na rozdiel od ideálnych prvkov, ktoré sú reprezentované len jednou veličinou (napríklad kapacita pre kondenzátor) a tým je ich správanie sa v obvode zjednodušené. Pre demonštráciu reálnych obvodov je však vhodné brať do úvahy viaceré vlastnosti jednotlivých prvkov, čo si však vyžiada pri simuláciách vyšší počet rovníc a adekvátne počtu i vyššie nároky na výpočtové zdroje. Platí tiež, že čím menšie hodnoty parazitné kapacity nadobúdajú, tým menej ovplyvňujú daný obvod. Parazitné javy môžu mať rôznorodý pôvod a v niektorých prípadoch majú pozitívny vplyv na správanie sa obvodu [48].

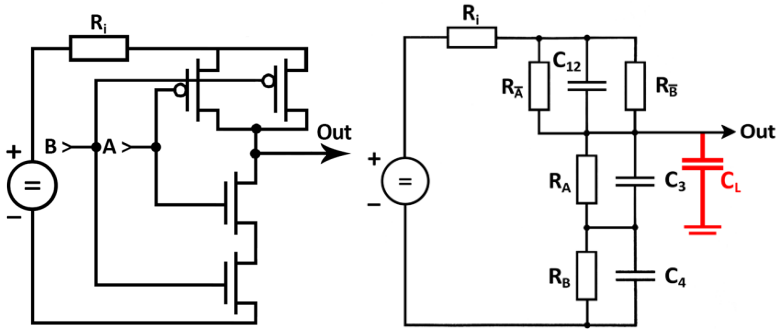
4.2 Návrh modelov CMOS NAND, CMOS NOR

Obdobne som navrhla modely CMOS NAND a CMOS NOR obvodov, opäť som jednotlivé tranzistory nahradila kapacitormi a rezistormi, nechýbajú ani parazitné kapacity. Logické hodnoty odpovedajúce vstupnému V_{in} a výstupnému napätiu V_{out} odpovedajúce operáciám NAND a NOR sú znázornené v tabuľke 4.2. Pomocou CMOS invertora a týchto obvodov je možné skladať zložitejšie logické obvody, ktoré riešia náročnejšie logické operácie.

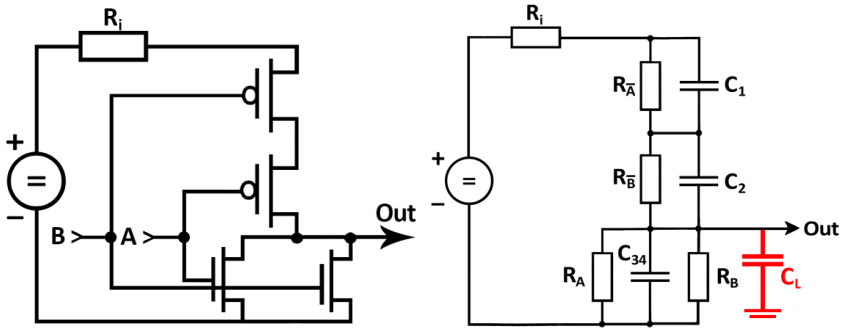
Tabuľka 4.2: Logické hodnoty odpovedajúce vstupnému V_{in} a výstupnému napätiu V_{out} pri využití CMOS NAND a CMOS NOR.

V_{in}		NAND	NOR
1	1	0	0
1	0	1	0
0	1	1	0
0	0	1	1

Logická operácia NAND (negácia AND), predstavuje negáciu logického súčinu daných vstupov. Je hodne využívaná, pretože pomocou tejto operácie dokážeme vytvoriť akékoľvek



Obr. 4.5: Model logického obvodu CMOS NAND, ktorý miesto štyroch tranzistorov typu p a n (vľavo) pozostáva zo štyroch párov rezistor-kapacitor (vpravo). Na výstupe sa nachádza i kapacitor C_L reprezentujúci parazitné javy pre priblíženie sa reálnym obvodom.



Obr. 4.6: Model logického obvodu CMOS NOR, ktorý miesto štyroch tranzistorov typu p a n (vľavo) pozostáva zo štyroch párov rezistor-kapacitor (vpravo), podobne ako u CMOS NAND. Na výstupe je i kapacitor C_L s parazitnou kapacitou pre lepšiu reprezentáciu skutočných obvodov.

ďalšie logické operácie. Operácia NOR (negácia OR) naopak predstavuje negáciu logického súčtu.

4.3 Overenie funkčnosti modelov pomocou nástroja FOS

Pre overenie funkčnosti návrhov CMOS invertoru, CMOS NAND a CMOS NOR obvodov som využila nástroj FOS [2], ktorý svojimi vlastnosťami spĺňa predpoklady pre dostatočne presný výpočet diferenciálnych rovníc, popisujúcich dané návrhy, a tým umožňuje experimentovať s uvedenými modelmi logických obvodov, podobne ako TKSL [7].

Rovnice pre model CMOS invertoru (obrázok 4.4) s použitím dvojích rezistor-kapacitor (a bez C_L), pre diferenciálne rovnice i s počiatočnými podmienkami, sú nasledovné (4.1) [40]:

$$
\begin{aligned}
i &= \frac{1}{R_i}.(U - u_{C_1} - u_{C_2}) \\
u'_{C_1} &= \frac{1}{C_1}.(i - \frac{1}{R_{\bar{A}}}.u_{C_1}), \qquad u_{C_1}(0) = 3.3 \\
u'_{C_2} &= \frac{1}{C_2}.(i - \frac{1}{R_A}.u_{C_2}), \qquad u_{C_2}(0) = 0
\end{aligned} \tag{4.1}
$$

Model CMOS invertoru by mal pracovať nasledovným spôsobom: ak obdrží na vstupe logickú hodnotu 1, uzavrie sa horný tranzistor (typ p) a otvorený ostane spodný tranzistor (typ n). Pre variantu invertoru s rezistormi a kapacitormi je toto správanie simulované dvoma rezistormim z ktorých jeden sa vyznačuje vysokým a naopak druhý malým odporom.

Zdrojový súbor pre systém FOS (vrátane C_L s parazitnou kapacitou) môže vyzerať nasledovne (pre stručnosť neuvádzam parametre vykresľovania grafu a nastavenie hodnôt premenných 4.2):

```
R1 = Rclosed if t < tmax/2 else Ropen;
R2 = Ropen   if t < tmax/2 else Rclosed;

i = 1/Ri*(U - uc1 - uc2);

uc1' = 1/C1     *(i - uc1/R1) &uC10;
uc2' = 1/(C2+CL)*(i - uc2/R2) &uC20;

in = 1 if t < tmax/2 else 0;
out = uc2; // = ucl
```

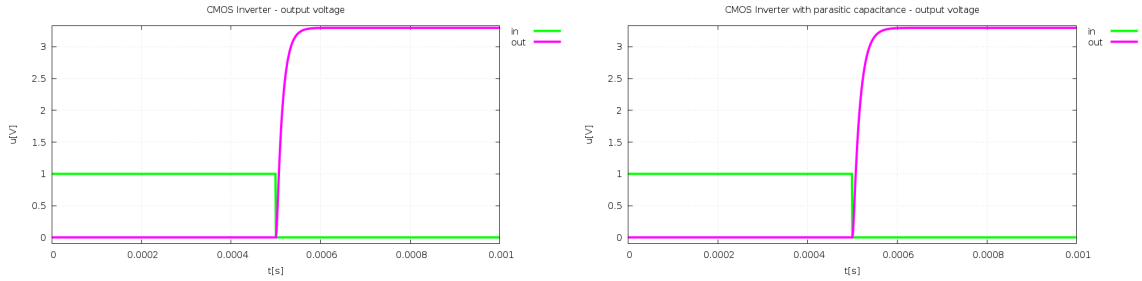
Pre vytvorenie zdrojového súboru a následného výpočtu CMOS invertoru v programe FOS som zvolila tieto hodnoty parametrov:

$$
\begin{aligned}
&C_1 = C_2 = 5.10^{-6}F \quad && R_A, R_{\bar{A}} \in \{1, 10^{10}\}\Omega \\
&C_L = 10^{-6}F && R_i = 1\Omega \\
&U = 3.3V
\end{aligned} \tag{4.2}
$$

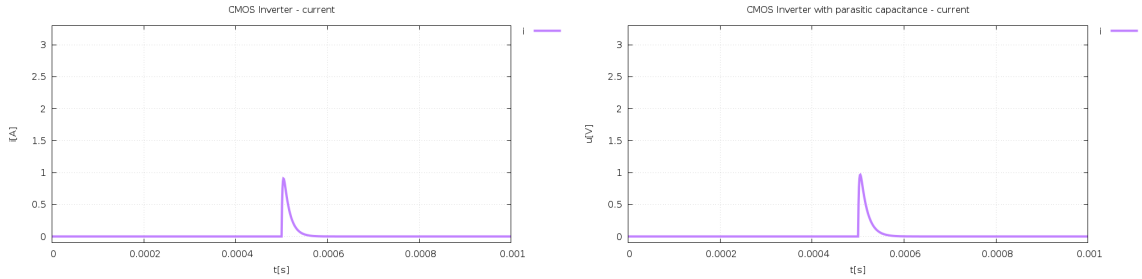
Rovnaké hodnoty kapacitorov C_1, C_2 ako aj prednabitie C_1 na hodnotu napätia rovnú zdroju U, zabezpečujú najlepšie správanie sa obvodu. Zdroj napätia s napätím $3.3V$ je blízkym skutočnosti. Parazitná kapacita C_L je podla možností čo najmenšia, rádovo $10^{-6}V$. Výstupom obvodu je hodnota napätia u_{C_2} alebo u_{C_L}, pretoße tieto hodnoty napätí sa rovnajú. Pri určovaní výsledného napätia vždy berieme do úvahy napätie voči zemi. Čas simulácie modelu som nastavila na $10^{-3}s$, počas ktorého dôjde k prepnutiu vstupnej hodnoty z 1 na 0.

Graficky je simulácia spracovaná na obrázku 4.7, kde sa mení hodnota napätia v závislosti od času. Pri zmene vstupnej logickej hodnoty z 1 na 0 v čase $t = 0.0005s$ dochádza k prechodnému deju, počas ktorého hodnota napätia rastie na hodnotu $U = 3.3V$, odpovedajúcej logickej 1. Toto výstupné napätie môže následne riadiť naňho nadväzujúce obvody. Výsledky v programe FOS získame v relatívne krátkom čase.

Pri porovnaní výsledkov výpočtov, s pripojením kapacitora C_L a bez neho, sa grafy závislosti výstupného napätia na pohľad neodlišujú. Pri náhľade na grafy vývoja eletrického prúdu v obvode tiež nebadáme rozdiely (obrázok 4.8). Voľba nízkej hodnoty parazitnej kapacity kapacitora C_L zabezpečuje, že nedochádza k výraznému ovplyvneniu správania sa obvodu, čo je žiadúce.



Obr. 4.7: Overenie funkčnosti návrhu CMOS invertoru v programe FOS, vrátane kapacitora C_L (vpravo). Pri logickej 1 na vstupe výstupné napätie dosahovalo hodnoty veľmi blízke nule, čo odpovedá logickej 0. Naopak, pri vstupnej hodnote 0 napätie stúpne na $3.3V$, čo považujeme za logickú 1.



Obr. 4.8: Porovnanie závislostí vývoja elektrického prúdu na čase (výstup v programe FOS), pretekajúceho navrhnutým obvodom CMOS invertoru bez (vľavo) a tiež s prítomnosťou parazitnej kapacity.

Pri zostrojení rovníc pre CMOS invertor po pripojení kapacitora C_L som využila Ohmov zákon (2.11), pre vyjadrenie prúdu i, pretekajúceho obvodom (obrázok 4.9):

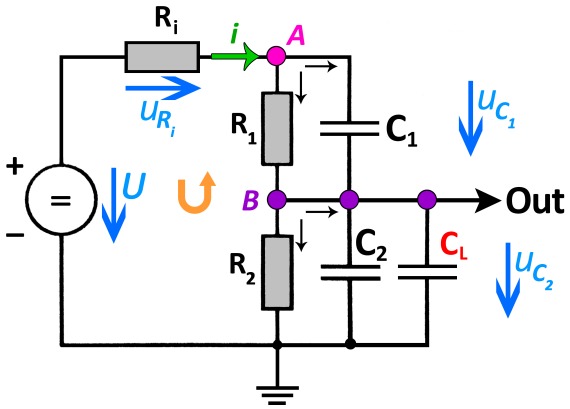
$$i = \frac{u_{R_i}}{R_i} \tag{4.3}$$

Ďalej podľa II. Kirchhoffovho zákona (2.28) platí:

$$U - u_{R_i} - u_{C_1} - u_{C_2} = 0 \tag{4.4}$$

Z tejto rovnice (4.4) som si vyjadrila u_{R_i} a dosadila do rovnice (4.3), čím som dostala finálnu rovnicu pre prúd i (4.5), ktorú som následne použila v zdrojovom kóde pre program FOS:

$$\begin{aligned} u_{R_i} &= U - u_{C_1} - u_{C_2} \\ i &= \frac{U - u_{C_1} - u_{C_2}}{R_i} \end{aligned} \tag{4.5}$$



Obr. 4.9: Detailný popis schémy CMOS invertoru s parazitnou kapacitou C_L, spolu s vyznačením jednotlivých napätí na komponentoch obvodu, smerov prúdu a vyznačenie uzlov A, B.

Pre zmenu napätí na kapacitoroch si môžeme vyjadriť nasledovné rovnice:

$$\begin{aligned} u'_{C_1} &= \frac{1}{C_1}.i_{C_1} \\ u'_{C_2} &= \frac{1}{C_2}.i_{C_2} \\ u'_{C_L} &= \frac{1}{C_L}.i_{C_L} \end{aligned} \tag{4.6}$$

Pre tieto rovnice je ešte potrebné si vyjadriť prúdy $i_{C_1}, i_{C_2}, i_{C_L}$. Na to poslúži I. Kirchhoffov zákon (2.12), aplikovaný na uzol A v obvode (obrázok 4.9):

$$\begin{aligned} i - i_{C_1} - i_{R_1} &= 0 \\ i_{C_1} &= i - i_{R_1} = \\ &= i - \frac{u_{R_1}}{R_1} \end{aligned} \tag{4.7}$$

Pre vyjadrenie i_{R_1} v poslednom výraze (4.7) bol opäť použitý Ohmov zákon, a keďže pre dané paralelné zapojenie rezistora R_1 a kapacitora C_1 platí, že $u_{R_1} = u_{C_1}$ (4.8),

$$\begin{aligned} i_{C_1} &= i - \frac{u_{R_1}}{R_1} = \\ &= i - \frac{u_{C_1}}{R_1} \end{aligned} \tag{4.8}$$

tak rovnica pre zmenu napätia na u_{C_1} je nasledovná:

$$u'_{C_1} = \frac{1}{C_1}.(i - \frac{u_{C_1}}{R_1}) \tag{4.9}$$

Pre vyjadrenie zvyšných prúdov i_{C_2}, i_{C_L} som v obvode na obrázku 4.9 aplikovala I. Kirchhoffov zákon (2.12) na uzol B, kde som za i_{C_1} dosadila jeden z výrazov (4.7) a na i_{R_2}

som aplikovala Ohmov zákon:

$$\begin{aligned} 0 &= i_{R_1} + i_{C_1} - i_{R_2} - i_{C_2} - i_{C_L} = \\ &= i_{R_1} + (i - i_{R_1}) - i_{R_2} - i_{C_2} - i_{C_L} = \\ &= i - i_{R_2} - i_{C_2} - i_{C_L} = \\ &= i - \frac{u_{R_2}}{R_2} - i_{C_2} - i_{C_L} \end{aligned} \tag{4.10}$$

Z rovnice (4.10) som si následne vyjadrila prúd pretekajúci kapacitorom C_2, teda i_{C_2} a tiež využila, že pre dané paralelné zapojenie rezistora R_2 a kapacitora C_2 platí, že $u_{R_2} = u_{C_2}$:

$$\begin{aligned} i_{C_2} &= i - \frac{u_{R_2}}{R_2} - i_{C_L} = \\ &= i - \frac{u_{C_2}}{R_2} - i_{C_L} \end{aligned} \tag{4.11}$$

Potom rovnica pre zmenu napätia na u_{C_2}, po dosadení (4.11) do príslušnej rovnice z (4.6), je nasledovná:

$$\begin{aligned} u'_{C_2} &= \frac{1}{C_2}.i_{C_2} = \\ &= \frac{1}{C_2}.(i - \frac{u_{C_2}}{R_2} - i_{C_L}) \end{aligned} \tag{4.12}$$

Rovnica 4.12 však nie je postačujúca, pretože v nej figuruje i_{C_L}, ktorého vyjadrenie zatiaľ nepoznáme. Z obrázka 4.9 je však známe, že $u_{C_2} = u_{C_L} = u_{out}$, a teda bude platiť, že $u'_{C_2} = u'_{C_L}$, a spolu s príslušnou diferenciálnou rovnicou z (4.6) dostaneme:

$$\begin{aligned} u'_{C_2} &= u'_{C_L} \\ \frac{1}{C_2}.(i - \frac{u_{C_2}}{R_2} - i_{C_L}) &= \frac{1}{C_L}.i_{C_L} \end{aligned} \tag{4.13}$$

Výraz (4.13) následne upravujeme a vyjadríme si i_{C_L}:

$$\begin{aligned} \frac{i - \frac{u_{C_2}}{R_2} - i_{C_L}}{C_2} &= \frac{i_{C_L}}{C_L} \\ C_L.(i - \frac{u_{C_2}}{R_2} - i_{C_L}) &= C_2.i_{C_L} \\ C_L.i - C_L.\frac{u_{C_2}}{R_2} - C_L.i_{C_L} &= C_2.i_{C_L} \\ C_L.i - C_L.\frac{u_{C_2}}{R_2} &= C_2.i_{C_L} + C_L.i_{C_L} \\ C_L.i - C_L.\frac{u_{C_2}}{R_2} &= (C_2 + C_L).i_{C_L} \\ i_{C_L} &= \frac{C_L.i - C_L.\frac{u_{C_2}}{R_2}}{C_2 + C_L} = \\ &= \frac{C_L}{C_2 + C_L}.i - \frac{C_L}{C_2 + C_L}.\frac{u_{C_2}}{R_2} \end{aligned} \tag{4.14}$$

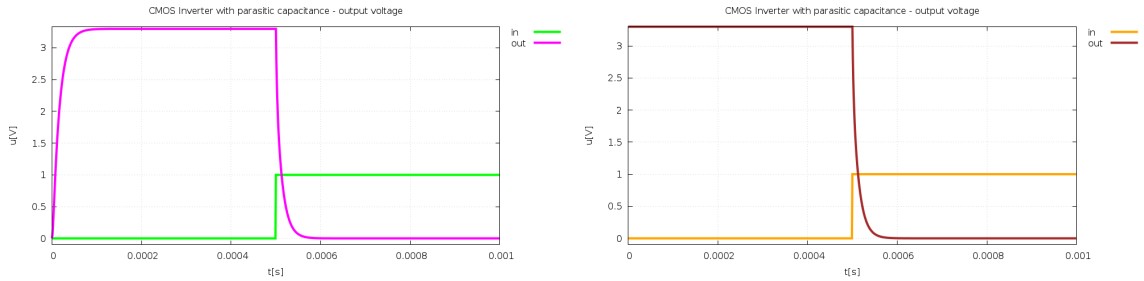
Po dosadení výsledného vyjadrenia i_{C_L} z (4.14) do neúplného vyjadrenia u'_{C_2} (4.12) dostávame výslednú rovnicu pre zmenu napätia na C_2, ktorá je zároveň rovnicou pre zmenu

napätia na C_L:

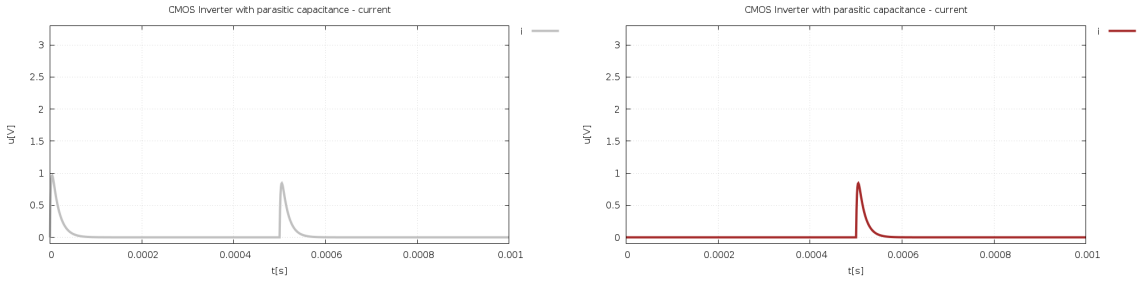
$$\begin{aligned}
u'_{C_2} &= \frac{1}{C_2}.(i - \frac{u_{C_2}}{R_2} - i_{C_L}) = \\
&= \frac{1}{C_2}.(i - \frac{u_{C_2}}{R_2} - \frac{C_L}{C_2 + C_L}.i + \frac{C_L}{C_2 + C_L}.\frac{u_{C_2}}{R_2}) = \\
&= \frac{1}{C_2}.((1 - \frac{C_L}{C_2 + C_L}.i + (\frac{C_L}{C_2 + C_L} - 1).\frac{u_{C_2}}{R_2}) = \\
&= \frac{1}{C_2}.(\frac{C_2 + C_L - C_L}{C_2 + C_L}.i + \frac{C_L - C_2 - C_L}{C_2 + C_L}.\frac{u_{C_2}}{R_2}) = \\
&= \frac{1}{C_2}.(\frac{C_2}{C_2 + C_L}.i - \frac{C_2}{C_2 + C_L}.\frac{u_{C_2}}{R_2}) = \\
&= \frac{1}{C_2 + C_L}.i - \frac{1}{C_2 + C_L}.\frac{u_{C_2}}{R_2} = \\
&= \frac{1}{C_2 + C_L}.(i - \frac{u_{C_2}}{R_2}) \\
u'_{C_L} &= u'_{C_2}
\end{aligned} \tag{4.15}$$

Model CMOS invertoru s uvedenými rovnicami a počiatočnými podmienkami je funkčný i v prípade prepnutia logickej hodnoty z hodnoty z 0 na 1, a to pri tých istých počiatočných podmienkach (4.1), bez kapacitora C_L i s ním. Pri výmene počiatočných podmienok (teda na podmienky (4.16)) získame však odlišné riešenie, znižujúce množstvo prúdu i (obrázok 4.11), pretekajúceho obvodom. Rozdiel je i v grafoch výsledného napätia (obrázok 4.10). Súčet hodnôt počiatočných podmienok by mal byť rovný hodnote napätia zdroja U.

$$\begin{aligned}
u_{C_1}(0) &= 0 \\
u_{C_2}(0) &= 3.3
\end{aligned} \tag{4.16}$$



Obr. 4.10: Závisloť výstupného napätia CMOS invertoru v programe FOS, vrátane C_L s parazitnou kapacitou, so vstupnými hodnotami 0 a 1, vľavo s počiatočnými podmienkami (4.1), ktoré po ich vzájomnej zámene majú za následok čiastočne odlišný výstup (vpravo).



Obr. 4.11: Závisloť prúdu i CMOS invertoru v programe FOS, vrátane kapacitora C_L, so vstupnými hodnotami 0 a 1, vľavo s počiatočnými podmienkami, uvedenými v (4.1), ktoré po ich vzájomnej zámene majú za následok pokles prúdu na začiatku intervalu (vpravo) v porovnaní pred zmenou počiatočných podmienok.

Výsledné rovnice CMOS invertoru s použitím parazitnej kapacity, z ktorých vychádza i zdrojový kód programu FOS, sú (4.5), (4.9) a (4.15), teda:

$$\begin{aligned} i &= \frac{U - u_{C_1} - u_{C_2}}{R_i} \\ u'_{C_1} &= \frac{1}{C_1}.(i - \frac{u_{C_1}}{R_1}), \qquad u_{C_1}(0) = 3.3 \\ u'_{C_2} &= \frac{1}{C_2 + C_L}.(i - \frac{u_{C_2}}{R_2}), \quad u_{C_2}(0) = 0 \\ u_{C_L} &= u_{C_2} \end{aligned} \tag{4.17}$$

Simulácie CMOS obvodov som implementovala i v nástroji MATLAB, kde sa pre všetky možné hodnoty predpočítali výstupy, ktoré bolo možné následne okamžite použiť.

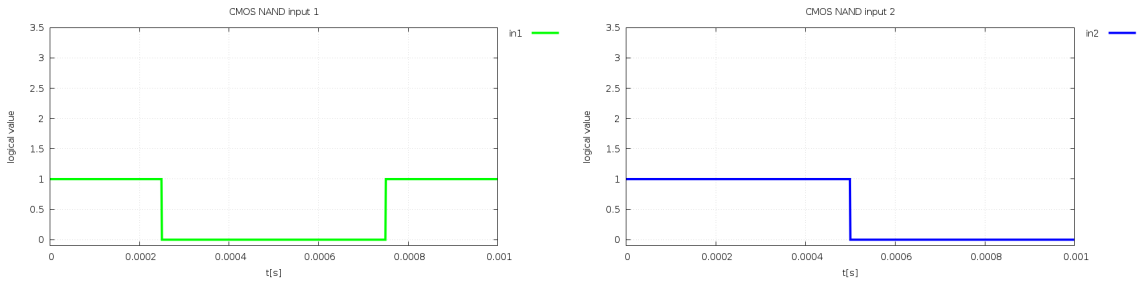
Pre jednoduchosť som zvolila riešenie CMOS invertoru s použitím konštantného zdroja napätia. Použitie striedavého zdroja napätia je možné, vyprodukuje však ďaleko viac rovníc, potrebných pre čo najpresnejšiu reprezentáciu navrhovaného modelu (implementovaním tvoriacich diferenciálnych rovníc (3.2)), no znamená ďalšie priblíženie sa skutočnosti.

Obdobným spôsobom som simulovala správanie sa navrhnutých obvodov CMOS NAND (obrázok 4.5) a CMOS NOR (obrázok 4.6). Rovnice modelu logického obvodu CMOS NAND 4.18 a CMOS NOR 4.19, s použitím párov rezistorov a kapacitorov (pre diferenciálne rovnice i s počiatočnými podmienkami; bez C_L), majú nasledovný tvar [40]:

$$\begin{aligned} i &= \frac{1}{R_i}.(U - u_{C_{12}} - u_{C_3} - u_{C_4}) \\ u'_{C_{12}} &= \frac{1}{C_{12}}.(i - \frac{R_{\bar{A}} + R_{\bar{B}}}{R_{\bar{A}}.R_{\bar{B}}}.u_{C_{12}}), \qquad u_{C_{12}}(0) = 3.3 \\ u'_{C_3} &= \frac{1}{C_3}.(i - \frac{1}{R_A}.u_{C_3}), \qquad u_{C_3}(0) = 0 \\ u'_{C_4} &= \frac{1}{C_4}.(i - \frac{1}{R_B}.u_{C_4}), \qquad u_{C_4}(0) = 0 \end{aligned} \tag{4.18}$$

$$
\begin{aligned}
i &= \frac{1}{R_i}.(U - u_{C_1} - u_{C_2} - u_{C_{34}}) \\
u'_{C_1} &= \frac{1}{C_1}.(i - \frac{1}{R_{\bar{A}}}.u_{C_1}), & u_{C_1}(0) &= 1.65 \\
u'_{C_2} &= \frac{1}{C_2}.(i - \frac{1}{R_{\bar{B}}}.u_{C_2}), & u_{C_2}(0) &= 1.65 \\
u'_{C_{34}} &= \frac{1}{C_{34}}.(i - \frac{R_A + R_B}{R_A.R_B}.u_{C_{34}}), & u_{C_{34}}(0) &= 0
\end{aligned}
\tag{4.19}
$$

Spolu s parazitnými kapacitami však pribudne ďalšia diferenciálna rovnica, pre každý z navrhovaných CMOS obvodov (okrem NOT). Existuje však možnosť zjednodušenia obvodu a teda nenavýšenia počtu jeho rovníc.



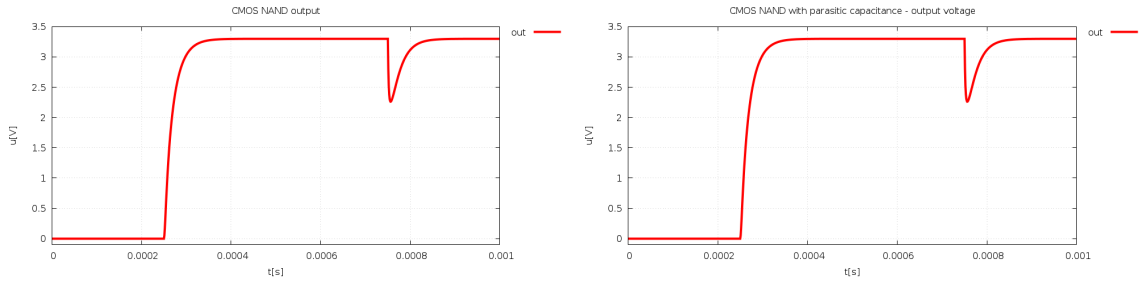
Obr. 4.12: Vstupné logické hodnoty obvodov CMOS NAND a CMOS NOR, ovplyvňujúce hodnoty rezistorov a tým výstupy týchto logických obvodov (program FOS).

Adekvátne rovniciam sa zmenil i zdrojový text CMOS NAND (a obdobne i CMOS NOR) pre program FOS. Hodnoty jednotlivých parametrov (4.2) pritom ostali zachované. Výstupné napätie pre CMOS NAND je súčtom napätí na kapacitoroch u_{C_3}, u_{C_4} i pri prítomnosti parazitnej kapacity C_L:

```
in1 = 1, 0, 0, 1;
in2 = 1, 1, 0, 0;

R1 = in1*Rclosed+(1-in1)*Ropen;
R2 = in2*Rclosed+(1-in2)*Ropen;
R3 = in1*Ropen+(1-in1)*Rclosed;
R4 = in2*Ropen+(1-in2)*Rclosed;

i   = (U - uc12 - uc3 - uc4)/Ri;
icl = CL*(C3 + C4)*i - CL*C4/C3*uc3/R3 - CL*C3/C4*uc4/R4;

uc12' = 1/C12*(i - uc12/R1 - uc12/R2) &uC120;
uc3'  = 1/C3 *(i - uc3/R3  - icl)     &uC30;
uc4'  = 1/C4 *(i - uc4/R4  - icl)     &uC40;

out = uc3 + uc4;
```

Z grafického porovnania je možné konštatovať, že rozdiel medzi výstupným napätím CMOS NAND s parazitnou kapacitou a bez nej i v tomto prípadeje nie je badateľný (grafy 4.13). Tomu odpovedajú i hodnoty prúdu pretekajúceho týmto logickým obvodom (obrázok

4.14), v čase od 0 do 0.001s.



Obr. 4.13: Overenie funkčnosti návrhu CMOS NAND, s pridanou parazitnou kapacitou i bez nej (vľavo). Pri vstupných logických hodnotách 1,1 výstupné napätie dosahuje nulové hodnoty (logická 0). V ostatných prípadoch vstupných hodnôt (obrázok 4.12) napätie stúpne na 3.3V, čo odpovedá logickej 1.

Do modelu CMOS NOR som implementovala zjednodušenie (podobne ako u CMOS invertoru), pri ktorom je výstupné napätie rovné u_C5 (obrázok 4.15). Zjednodušenie spočíva v úprave časti obvodu. Využila som paralelné zapojenie rezistorov R_A a R_B a kapacitorov $C34$ a CL.

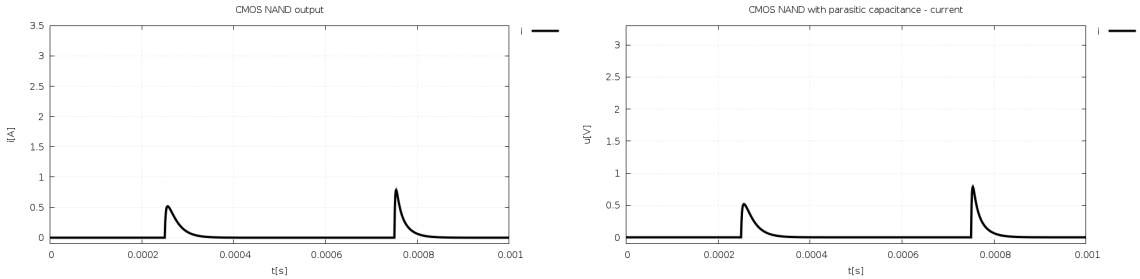
Zdrojový kód pre systém FOS, reprezentujúci model CMOS NOR po zjednodušení, je nasledovný:

```
uC50 = 0;
C5 = C34 + CL;
R5 = (R3 * R4)/(R3 + R4);

i = 1/Ri*(U - uc1 - uc2 - uc5);

uc1' = 1/C1 * (i - uc1/R1) &uC10;
uc2' = 1/C2 * (i - uc2/R2) &uC20;
uc5' = 1/C5 * (i - uc5/R5) &uC50;

out = uc5;
```

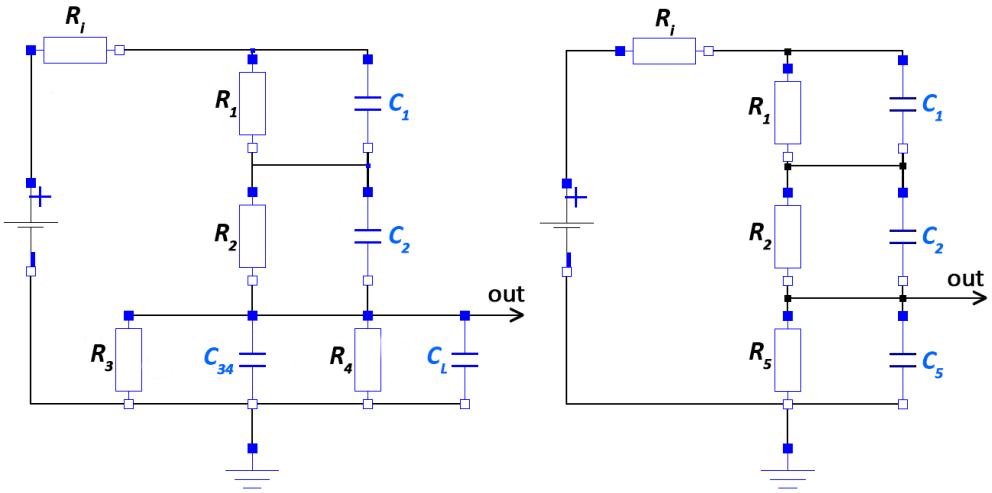
Model CMOS NOR som riešila i bez jeho zjednodušenia a bez parazitnej kapacity. Výstup takéhoto obvodu (pre porovnanie s použitím parazitnej kapacity) v programe FOS je na obrázku 4.16.

Funkčnosť návrhu CMOS NOR, s pridanou parazitnou kapacitou (obrázok 4.15), som tentoraz skúmala pri rozdelení časového intervalu na tri časti, pre lepšie znázornenie prechodných dejov obvodu, spojenými s prepínaním hodnôt rezistorov. Pri vstupných logických hodnotách 0,0 výstupné napätie dosahuje hodnotu 3.3V, čo odpovedá logickej 1 (obrázok 4.17 vpravo).

Pri rozdelení časového intervalu $\langle 0, 0.001 \rangle$ na tri časti, boli rovnice obvodu CMOS NOR prepočítané pre každú časť osobitne, a teda i hodnoty napätí na kapacitoroch sa na začiatku každého intervalu nastavili podľa hodnôt počiatočných podmienok. Pri overovaní funkčnosti obvodu pre celý časový interval, s počiatočnými podmienkami definovanými len pre začiatok celého intervalu, však objavíme nedostatočnosť, a to pre 2. polovicu celkového intervalu. Riešenie obvodu pomocou nástroja MATLAB však tento nedostatok odstraňuje.

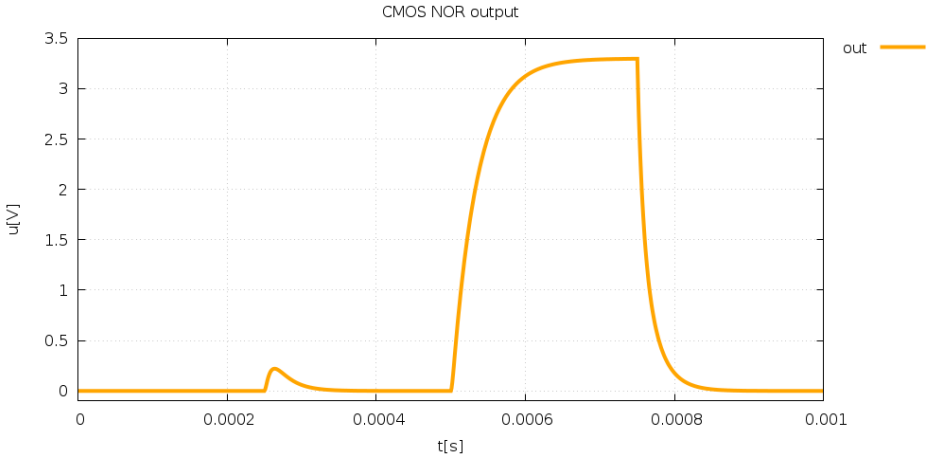


Obr. 4.14: Porovnanie závislostí vývoja elektrického prúdu na čase (program FOS), pretekajúceho navrhnutým obvodom CMOS NAND bez a s prítomnosťou parazitnej kapacity (vpravo). Pri porovnaní je možné pozorovať, že pripojenie C_L do obvodu zásadne neovplyvňuje vývoj prúdu v čase, taktiež i preto, že bola zvolená veľmi malá hodnota kapacity C_L.

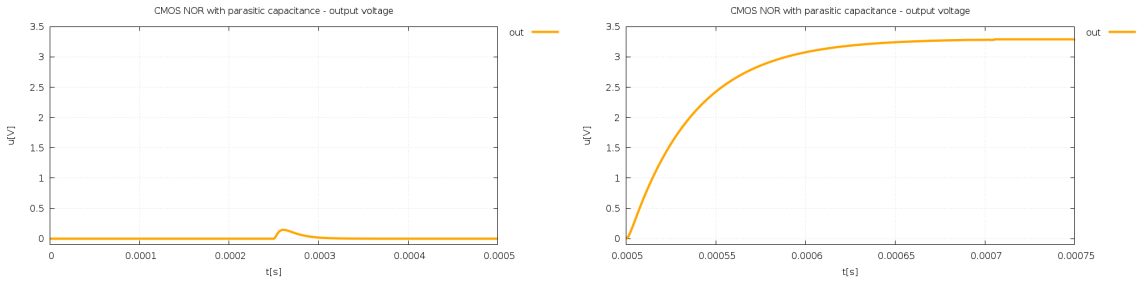


Obr. 4.15: Zjednodušenie modelu CMOS NOR po pridaní kapacitora C_L s parazitnou kapacitou, pre účely zmenšenia počtu diferenciálnych rovníc.

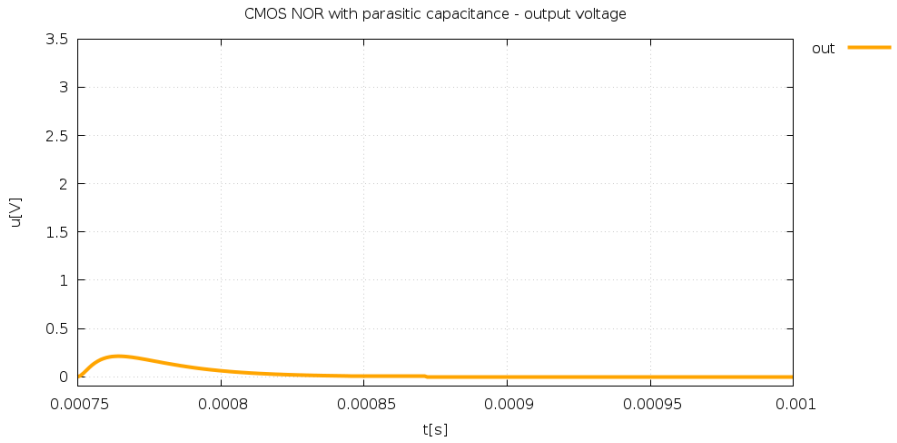
Z uvedených výsledkov vyplýva, že navrhnuté modely CMOS invertoru, CMOS NAND a CMOS NOR bez parazitnej kapacity i s ňou sú funkčné, ich výstupy odpovedajú hodnotám pravdivostných tabuliek pre jednotlivé logické operácie.



Obr. 4.16: Výstup modelu obvodu odpovedajúcemu logickej operácii NOR (napätie) v programe FOS, bez parazitnej kapacity.



Obr. 4.17: Výstup modelu obvodu odpovedajúcemu logickej operácii NOR (napätie) v programe FOS, spolu s parazitnou kapacitou.



Obr. 4.18: Podrobnejší náhľad na časť výstupu (napätie) modelu obvodu odpovedajúcemu logickej operácii NOR v programe FOS, s parazitnou kapacitou.

Kapitola 5

Implementácia simulácie CMOS obvodov

Implementáciu simulácie CMOS obvodov som realizovala v prostredí MATLAB, ktoré je jedným z najrozšírenejších v mnohých odvetviach či na školách. Možnosti implementácie simulačných výpočtov sú však široké.

Simuláciu CMOS invertoru som uskutočnila vytvorením jednoduchého skriptu v jazyku MATLAB, ktorý disponuje rozsiahlou škálou nástrojov, a tým poskytuje používateľovi nemalé možnosti. Podobne som postupovala i pri obvodoch CMOS NAND a CMOS NOR, tentoraz bez parazitných kapacít, keďže rozdiely vo výsledkoch sú veľmi malé. V prípade použitia parazitných kapacít pridáme či upravíme jednu diferenciálnu rovnicu a počiatočnú podmienku alebo postačí malá úprava matice A.

Pre tzv. ODE solvre dostupné v tomto nástroji je pri výpočte diferenciálnych rovníc potrebné rovnice upraviť a nahradiť maticou, reprezentujúcou daný systém diferenciálnych rovníc. Je možné však rovnice zadávať priamo a využiť dostupné nástroje generovania potrebných matíc. V tejto kapitole objasním postup riešenia simulácie CMOS obvodov v MATLAB-e, načrtnem viaceré možnosti, výsledky, ktoré spojím do jediného skriptu umožňujúceho riešiť ktorýkoľvek zo spomenutých CMOS obvodov či ich i spájať, čo vedie na simuláciu rozsiahlych CMOS VLSI obvodov.

V MATLAB-e bude riešená počiatočná úloha v tvare

$$y' = A.y + b, y(0) = y_0, \tag{5.1}$$

kde A predstavuje maticu koeficientov a_{11} až a_{mn}, pričom veľkosť matice je m x n, zvislý vektor b je vektor pravých strán a ďalšie zvislé vektory y a y' tvoria vektory neznámych funkcií a ich derivácií.

Po inicializácii premenných a počiatočných podmienok potrebných pre simuláciu CMOS invertoru v MATLAB-e som následne definovala simulačné parametre, ktoré sú neoddeliteľnou súčasťou každej simulácie. Časový interval som rozdelila na dve polovice, pričom každá je venovaná inej vstupnej logickej hodnote, podobne ako v programe FOS (kapitola 4):

```
% simulation parameters
dt = 0.00001; % integration step size
t1 = 0.0005;
t2 = 0.001;
tspan1 = [0, t1];  % time of simulation
tspan2 = [t1, t2];
```

Ďalej som si pripravila matice potrebné pri výpočtoch diferenciálnych rovníc pre invertor pri oboch vstupných hodnotách. Pre logickú hodnotu 1 je matica $A1$ definovaná v jazyku MATLAB nasledovne:

```
% logical 1
RA = 1;
RA_ = 1e10;
A1 = [(-1/C1)*(1/Ri + 1/RA_), -1/(C1*Ri);
     -1/(C2*Ri), (-1/C2)*(1/Ri + 1/RA)];
```

Matica $A2$ reprezentuje hodnoty pre logickú 0.

Tvar matíc je možné získať z rovníc popisujúcich obvod CMOS invertoru (4.1) tak, že rovnicu pre vyjadrenie prúdu i dosadíme do ostatných rovníc, teda do výrazov pre u'_{C_1} a u'_{C_2}:

$$\begin{aligned} u'_{C_1} &= \frac{1}{C_1}.(\frac{1}{R_i}.(U - u_{C_1} - u_{C_2}) - \frac{1}{R_{\bar{A}}}.u_{C_1}) \\ u'_{C_2} &= \frac{1}{C_2}.(\frac{1}{R_i}.(U - u_{C_1} - u_{C_2}) - \frac{1}{R_A}.u_{C_2}) \end{aligned} \tag{5.2}$$

Úpravou rovníc (5.2) dostaneme potrebné koeficienty matice A. Úpravu ilustrujú výrazy (5.3) a (5.4). Prvý riadok A je venovaný koeficientom z rovnice pre u'_{C_1} a druhý riadok pre u'_{C_2}. Prvý stĺpec patrí koeficientom vyskytujúcich sa pri premennej u_{C_1} a druhý pri u_{C_2}.

$$\begin{aligned} u'_{C_1} &= \frac{1}{C_1}.(\frac{U}{R_i} - \frac{1}{R_i}.u_{C_1} - \frac{1}{R_i}.u_{C_2} - \frac{1}{R_{\bar{A}}}.u_{C_1}) = \\ &= \frac{U}{C_1.R_i} - \frac{1}{C_1.R_i}.u_{C_1} - \frac{1}{C_1.R_i}.u_{C_2} - \frac{1}{C_1.R_{\bar{A}}}.u_{C_1} = \\ &= \frac{U}{C_1.R_i} + (-\frac{1}{C_1.R_i} - \frac{1}{C_1.R_{\bar{A}}}).u_{C_1} + (\frac{-1}{C_1.R_i}).u_{C_2} = \\ &= \frac{U}{C_1.R_i} + (\frac{-1}{C_1}.(\frac{1}{R_i} + \frac{1}{R_{\bar{A}}})).u_{C_1} + (\frac{-1}{C_1.R_i}).u_{C_2} \end{aligned} \tag{5.3}$$

$$\begin{aligned} u'_{C_2} &= \frac{1}{C_2}.(\frac{U}{R_i} - \frac{1}{R_i}.u_{C_1} - \frac{1}{R_i}.u_{C_2} - \frac{1}{R_A}.u_{C_2}) = \\ &= \frac{U}{C_2.R_i} - \frac{1}{C_2.R_i}.u_{C_1} - \frac{1}{C_2.R_i}.u_{C_2} - \frac{1}{C_2.R_A}.u_{C_2} = \\ &= \frac{U}{C_2.R_i} + (\frac{-1}{C_2.R_i}).u_{C_1} + (-\frac{1}{C_2.R_i} - \frac{1}{C_2.R_A}).u_{C_2} = \\ &= \frac{U}{C_2.R_i} + (\frac{-1}{C_2.R_i}).u_{C_1} + (\frac{-1}{C_2}.(\frac{1}{R_i} + \frac{1}{R_A})).u_{C_2} \end{aligned} \tag{5.4}$$

Teda matica A má tvar:

$$A = \begin{bmatrix} \frac{-1}{C_1}.(\frac{1}{R_i} + \frac{1}{R_{\bar{A}}}) & \frac{-1}{C_1.R_i} \\ \frac{-1}{C_2.R_i} & \frac{-1}{C_2}.(\frac{1}{R_i} + \frac{1}{R_A}) \end{bmatrix}$$

Vektory y a y' tvoria premenné reprezentujúce napätie a jeho zmenu na kapacitoroch u_{C_1} a u_{C_2}. Nie je nutné ich pre MATLAB explicitne definovať, no pre úplnosť uvádzam ich tvar:

$$y' = \begin{pmatrix} u'_{C_1} \\ u'_{C_2} \end{pmatrix}, y = \begin{pmatrix} u_{C_1} \\ u_{C_2} \end{pmatrix}$$

Vektor b naplníme zvyšnými hodnotami vyplývajúcimi z úprav rovníc (5.3) a (5.4):

$$b = \begin{pmatrix} \frac{U}{C_1.R_i} \\ \frac{U}{C_2.R_i} \end{pmatrix}$$

Vektor počiatočných podmienok je taktiež potrebné definovať:

```
y0 = [y0_1; y0_2]; % vector of initial conditions
```

Týmto máme všetko potrebné pre zahájenie výpočtov pomocou matlabovského ODE solvra. Zvolený bol *ode23* solver ako najbežnejší solver, vhodný pre väčšinu problémov reprezentovaných diferenciálnymi rovnicami [29]. Vhodný pre stabilné systémy, nevykazujúce tuhosť. Nasledujúcim kódom získame riešenie sústavy diferenciálnych rovníc:

```
myodefun1 = @(t,y) A1*y+b;
myodefun2 = @(t,y) A2*y+b;

[T_ode23, Y_ode23] = ode23(myodefun1, tspan1, y0);     % logical 1
[T_ode23_2, Y_ode23_2] = ode23(myodefun2, tspan2, y0); % logical 0
```

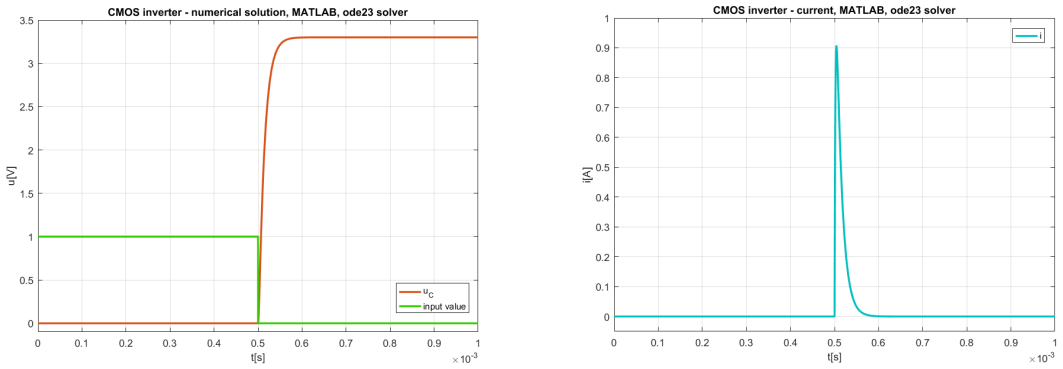
Vypočítané hodnoty riešenia u_{C_1} a u_{C_2} boli uložené v premennej Y_ode23 (pre vstupnú logickú hodnotu 1) a Y_ode23_2 (pre vstupnú hodnotu 0). K nim prislúchajúce hodnoty nezávislej premennej - času t sú v T_ode23 a T_ode23_2. Tieto hodnoty bolo potrebné ďalej spracovať. Výstupom CMOS invertoru je napätie u_{C_2}, ktoré som pre jednoduché grafické vykreslenie pre oba logické vstupy uložila do premennej $uc2$. Obdobne som spracovala výsledné napätie u_{C_1} (hodnoty v premennej $uc1$), pre potreby výpočtu prúdu i (premenná *cur*), pretekajúceho obvodom:

```
% for CMOS inverter out is uc2
lle = length(Y_ode23);
out = Y_ode23(1:lle);           % uc1 solution for input = 1
out_ode = Y_ode23((lle+1):end); % uc2 solution
logical_in = ones(1, lle);

lle = length(Y_ode23_2);
out_2 = Y_ode23_2(1:lle);           % uc1 solution  for input = 0
out_ode_2 = Y_ode23_2((lle+1):end); % uc2 solution
logical_in_2 = zeros(1, lle);

% join two vectors
t = [T_ode23; T_ode23_2];
in = [logical_in, logical_in_2];
uc1 = [out, out_2];         % uc1 solution, for getting current
uc2 = [out_ode, out_ode_2]; % uc2, the final solution

% getting current
cur = [];
c = 1/Ri;
for i = 1:length(uc1),
    cur(i) = c*(U - uc1(i) - uc2(i));
end
```

O vykreslenie výsledku (výstupné napätie *out*) do grafu sa stará nasledujúci kód:

```
figure
plot(t, uc2, t, in)
```

```
title('CMOS inverter - numerical solution, MATLAB, ode23 solver')
xlabel('t[s]')
ylabel('u[V]')
legend('u_C', 'input value');
```



Obr. 5.1: Výstup modelu CMOS obvodu odpovedajúcemu logickej operácii NOT (invertor) v programe MATLAB, bez parazitnej kapacity. Vpravo závislosť prúdu od času. Pre výpočty bol zvolený matlabovský *ode23* solver.

Podobne je možné zobraziť hodnoty prúdu i, pre celý časový interval $\langle 0, t2 \rangle$ (obrázok 5.1 vpravo).

Získaný výstup CMOS invertoru je veľmi podobný výstupu získanému v programe FOS (obrázok 4.7). V MATLAB-e je možné overiť výsledky aj pre ďalšie solvre.

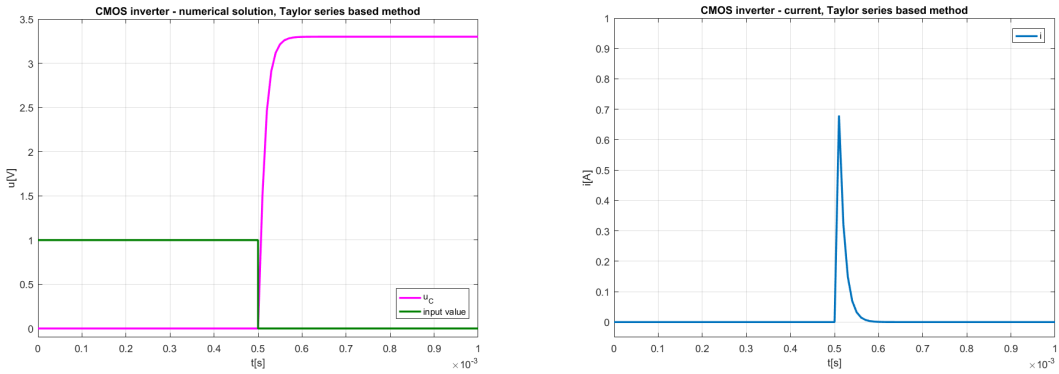
Pre porovnanie uvádzam výpočet riešenia návrhu CMOS invertoru upomocou explicitnej Taylorovej metódy (1. riadok je pre logickú hodnotu 1 a 2. pre 0):

```
[T_Taylor,  Y_Taylor,  ORD,DY] = explicitTaylorLinear(dt,tspan1,y0,A1,b,eps);
[T_Taylor_2,Y_Taylor_2,ORD,DY] = explicitTaylorLinear(dt,tspan2,y0,A2,b,eps);
```

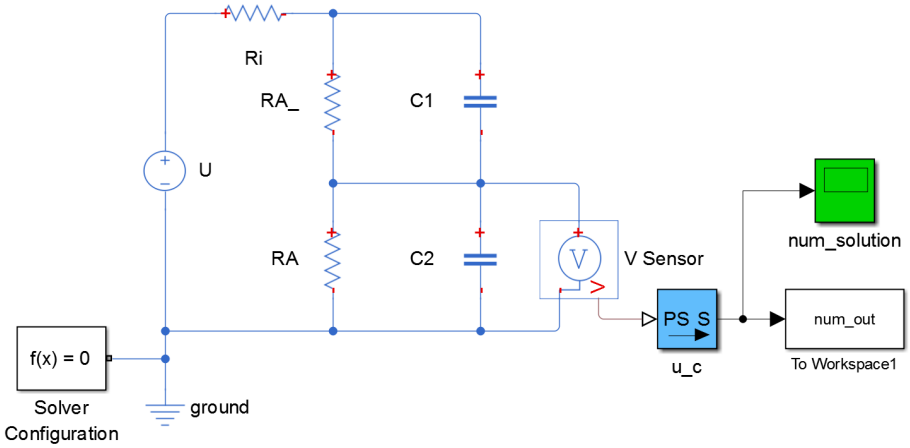
Pomocou `explicitTaylorLinear` voláme metódu pre výpočet riešenia diferenciálnych rovníc popisujúcich CMOS invertor. Táto metóda je implementovaná v osobitnom súbore *explicitTaylorLinear.m* [51]. Grafický výstup pre výsledné napätie u_{out} a prúd i je na obrázku 5.2.

Pri porovnaní výstupov CMOS invertoru v MATLAB-e pomocou ode23 solvra a explicitnej Taylorovej metódy grafické výstupy napätí na obrázkoch 5.1 a 5.2 na pohľad nevykazujú odlišnosti. Rozdiel je však v hodnotách prúdu i, ktorý pri použití ode23 solvra dosahuje pri prepnutí vstupnej logickej hodnoty z 1 na 0 hodnoty až $0.9A$, pričom maximálna hodnota prúdu vypočítaného pomocou explicitnej Taylorovej metódy je nižšia, teda menšia než $0.7A$. Tento rozdiel je zaujímavý a súvisí s presnosťou vykonaných výpočtov.

Model CMOS invertoru je možné simulovať v rámci systému MATLAB i prostredníctvom Simulink-u. Schéma modelu je na obrázku 5.3. I pomocou Simulink-u vieme získať výstupy, ktoré sú použiteľné a odpovedajúce správnym pravdivostným hodnotám príslušnej logickej operácie (NOT). Simulácia obvodov v MATLAB-e mimo prostredia Simulink je však používateľsky prívetivejšia, umožňuje nám jednoduchším spôsobom modifikovať obvod a hodnoty veličín, presnosť výpočtu a ďalšie parametre simulácie, to však za predpokladu, že použité rovnice sú pre daný obvod odvodené správne.



Obr. 5.2: Výstup modelu CMOS obvodu odpovedajúcemu logickej operácii NOT (invertor) v programe MATLAB, bez parazitnej kapacity. Vpravo závislosť prúdu od času. Pre výpočty bola zvolená explicitná Taylorova metóda.

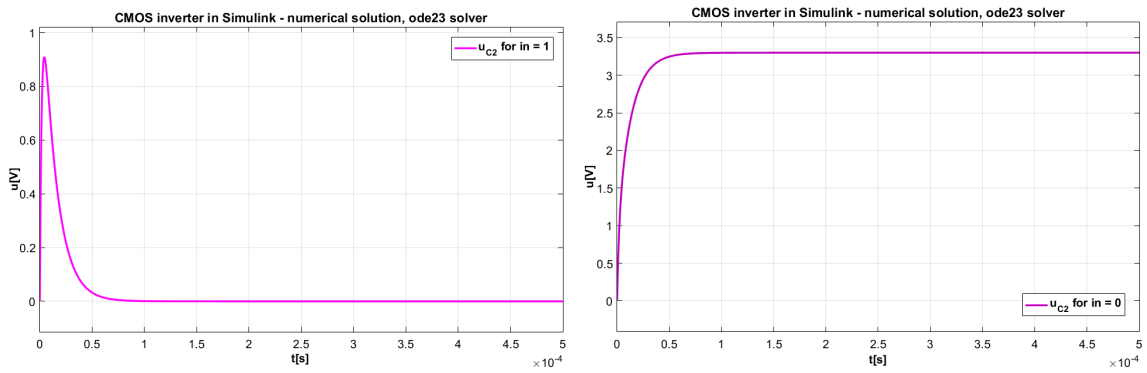


Obr. 5.3: Schéma zapojenia návrhu modelu CMOS invertoru pomocou dvojíc rezistor-kapacitor v simulačnom prostredí Simulink.

Pre návrhy logických obvodov CMOS NAND a CMOS NOR som vytvorila jednotný skript *nand_nor.m*, ktorý pomocou jednej premennej (*cmos*) rieši prepínanie sa medzi obvodom NAND a NOR, pričom jeho riešenie je podobné ako u CMOS invertoru. Zadefinovaním množstva konštánt pre často používané vzťahy je možné urýchliť výpočet tým, že si množstvo hodnôt predvypočítame a následne už len využívame v ďalších výpočtoch, nie je potrebné mnohé čiastkové výpočty vykonávať znova. Príkladom je nasledovný zdrojový kód:

```
if cmos == 1,
    c1  = -1/C12/Ri; % constants
    c2  = -1/C3/Ri;
    c3  = -1/C4/Ri;
...
    a = (RA_ + RB_)/C12/RA_/RB_;
    b = 1/C3/RA;
    c = 1/C4/RB;
```

```
...
A1 = [c1 - a, c1,     c1;
      c2,     c2 - b, c2;
      c3,     c3,     c3 - c];
```



Obr. 5.4: Výstup modelu CMOS obvodu odpovedajúcemu logickej operácii NOT (invertor) v programe Simulink, bez parazitnej kapacity, vľavo pre vstupnú logickú hodnotu 1 a vpravo pre 0.

Môj skript však poskytuje mnoho priestoru pre ďalšiu optimalizáciu. Optimalizácia takýchto výpočtov môže byť námetom pre ďalší výskum.

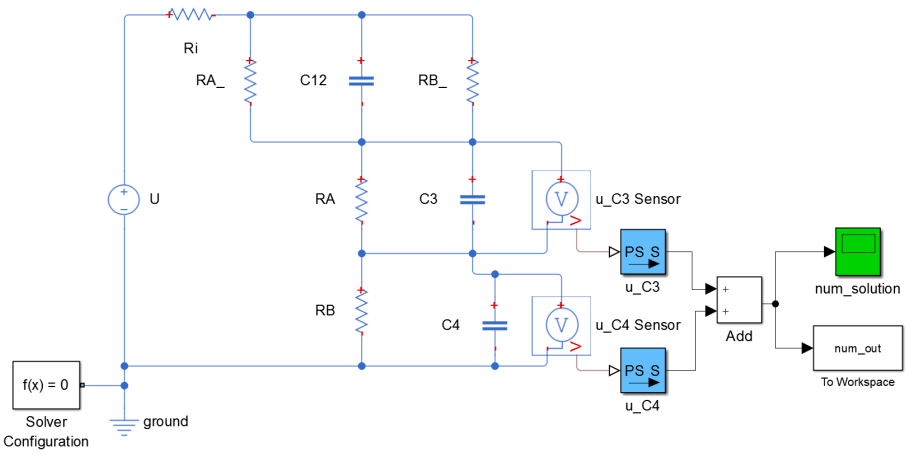
Časový interval $\langle 0, tmax \rangle$ som pre CMOS NAND a CMOS NOR (podobne ako vo FOS) rozdelila na štyri podintervaly, pričom každý z nich náleží iným logickým vstupným hodnotám. Tvar matíc pre jednotlivé časové intervaly a vstupné hodnoty som si odvodila a predvypočítala. Tieto matice je možné použiť opakovane, i pre viaceré numerické metódy (napríklad pre rôzne matlabovské solvre pre riešenie diferenciálnych rovníc). Výsledky pre jednotlivé časové podintervaly som následne spojila, pre názornejší grafický výstup. Z numerických metód boli zvolené opäť ode23 solver (obrázok 5.6 a 5.8) a explicitná Taylorova metóda (obrázok 5.7 a 5.9).

Výstupy CMOS NAND a CMOS NOR obvodov v MATLAB-e sú funkčné a odpovedajúce pre dané logické vstupy. Opäť sa však hodnoty prúdu počas prepnutia vstupných hodnôt pri použití explicitnej Taylorovej metódy odlišovali od hodnôt získaných pomocou ode23 solvra. Pre overenie som vytvorila schémy modelov i v Simulink-u a spustila simulácie pre rozličné vstupné hodnoty, ktorých správanie sa opäť vykazovalo priaznivé výsledky: hodnoty výstupných napätí opäť odpovedali očakávaným výstupným hodnotám. Obrázok schémy modelu CMOS NAND 5.5 predstavuje ukážku funkčného zapojenia prvkov obvodu, realizujúcich operáciu negácie logického súčinu. Podobne som v Simulink-u vytovrila i schému CMOS NOR a model odsimulovala.

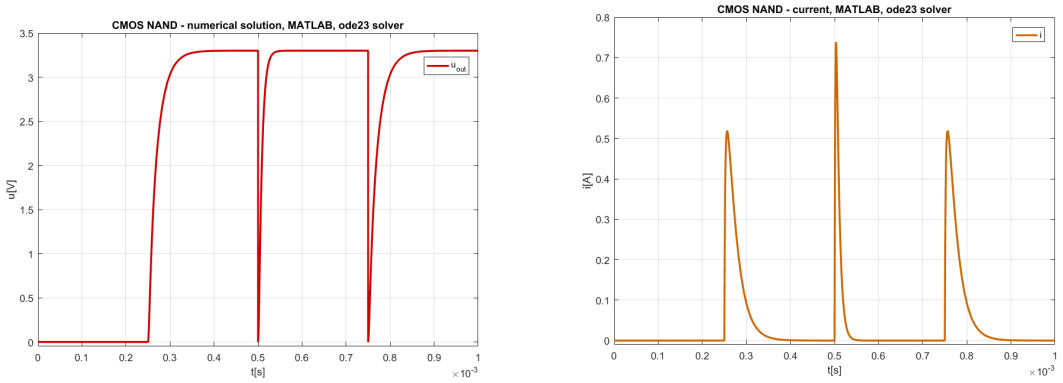
V snahe o porovnanie analytického a numerického riešenia, podobne ako u jednoduchého RC obvodu vo svetových simulačných štandardoch, však narazíme na to, že odvodiť analytické riešenie pre CMOS NAND alebo NOR nie je jednoduché. Taylorova metóda sa však považuje za veľmi presnú metódu [41]. Spôsob výpočtu ju však predurčuje pre optimalizáciu rýchlosti výpočtov.

V doterajších príkladoch som uviedla možnosti výpočtu s vopred známym tvarom matice A, pre riešenie systému diferenciálnych rovníc (5.1). Pre zložitejšie systémy zložené z tisícok logických obvodov je však nemožné ručne počítať tvar matíc či vektora b. Riešením sú

existujúce nástroje, schopné previesť systém rovníc do maticového tvaru. V MATLAB-e túto úlohu rieši metóda `equationsToMatrix` [27].



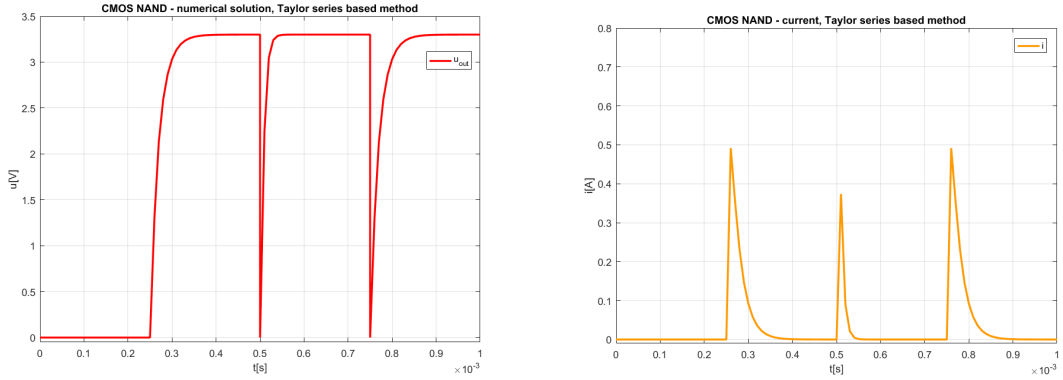
Obr. 5.5: Schéma zapojenia modelu logického obvodu CMOS NAND pomocou dvojíc rezistor-kapacitor, v simulačnom prostredí Simulink.



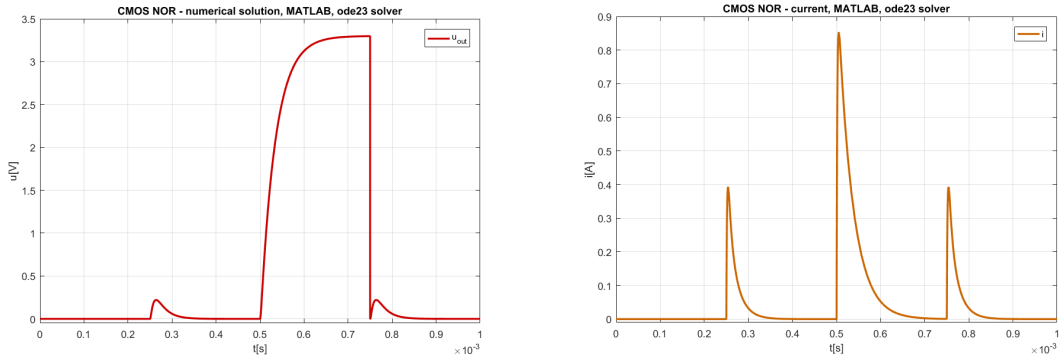
Obr. 5.6: Výstup modelu CMOS obvodu odpovedajúcemu logickej operácii NAND v programe MATLAB, bez parazitnej kapacity. Vpravo závislosť prúdu od času. Pre výpočty bol zvolený ode23 solver.

Skript *eqToAṁ* je názornou ukážkou, ako je z diferenciálnych rovníc možné získať maticu A, ktorú je následne možné využiť pri riešení daného elektrického obvodu pomocou príslušného solvru. Pri použití metódy `equationsToMatrix` nie je nutné uvádzať vektor b v hranatých zátvorkách ľavej strany rovnice.

```
syms uc1 uc2 % specify variables

% define equation for current
myI = @(uc1, uc2) (U - uc1 - uc2)/Ri;

% define differential equations
```



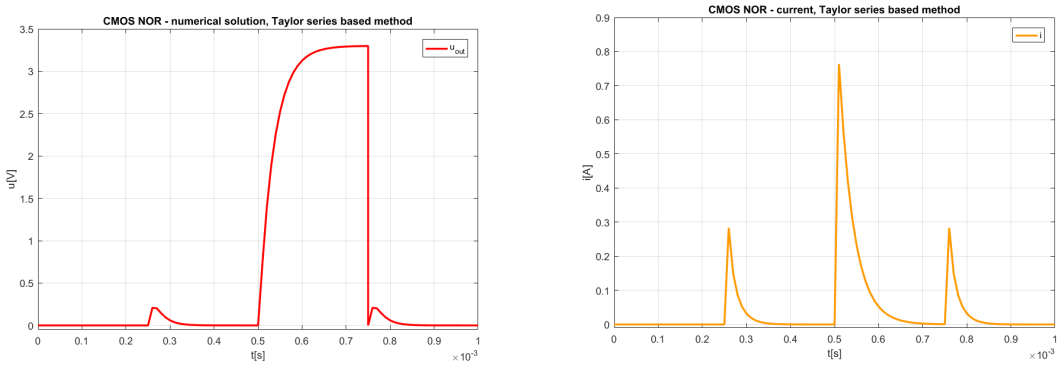
Obr. 5.7: Výstup modelu CMOS obvodu odpovedajúcemu logickej operácii NAND v programe MATLAB, bez parazitnej kapacity, použitím explicitnej Taylorovej metódy. Vpravo závislosť prúdu od času.



Obr. 5.8: Výstup modelu CMOS obvodu odpovedajúcemu logickej operácii NOR v programe MATLAB, bez parazitnej kapacity, vypočítaný pomocou ode23 solvru, dostupného v MATLAB-e. Vpravo závislosť prúdu od času.

```
myEquations = [uc1' == 1/C1*(myI - uc1/RA_);
               uc2' == 1/C2*(myI - uc2/RA)];

% get the matrix A and vector b
[A, b] = equationsToMatrix(myEquations);

% another solution, without definig equation for current (already substituted)
[B] = equationsToMatrix([uc1' == 1/C1*(1/Ri*(U-uc1-uc2)-1/RA_*uc1),
                         uc2' == 1/C2*(1/Ri*(U-uc1-uc2)-1/RA*uc2)]);

% make the matrix usable for ode solvers in MATLAB
A = double(A)*(-1);
```

Vyššie uvedený princíp pre získanie systému diferenciálnych rovníc v potrebnom tvare som využila pri riešení ďalších príkladov obvodov, pozostávajúcich z viacerých logických hradiel (CMOS invertoru, CMOS NAND alebo CMOS NOR), taktiež pri riešení obvodov pre viaceré vstupné hodnoty, z ktorých pozostávajú zložité CMOS VLSI obvody. Prístupov



Obr. 5.9: Výstup modelu CMOS obvodu odpovedajúcemu logickej operácii NOR v programe MATLAB, bez parazitnej kapacity. Vpravo závislosť prúdu od času. Pre výpočty bola zvolená explicitná Taylorova metóda.

existuje viacero, taktiež i možností, ako spájať uvedené obvody a následne ich simulovať. Predvypočítanie výsledkov pre všetky možné kombinácie logických vstupov ma význam v tom prípade, ak parametre prvkov pre každý z logických hradiel pre tú istú operáciu, sú zhodné (teda napríklad ak odpor kapacitora R_i jedného obvodu CMOS NAND bude rovnakej veľkosti ako odpor R_i iného CMOS NAND v danom systéme) a tiež neposkytuje mnoho priestoru pre demonštráciu paralelizácie výpočtov, pri ktorej je hlavnou úlohou zvládnuť veľké množstvo diferenciálnych rovníc.

Simuláciu CMOS invertoru, CMOS NAND a CMOS NOR som implementovala v skriptoch *inverter_EQtoA.m* a *nand_nor_EQtoA.m*, ktoré simulujú ich správanie sa a prepínanie medzi logickými hodnotami podobne ako v nástroji FOS, vykresľujú získané výsledky do grafov a pri vytváraní matice A využívajú dostupné nástroje MATLAB-u, teda nie je potrebné explicitne vytvárať tieto matice.

Vhodnejšími pre ďalšie experimenty či spájanie týchto CMOS obvodov do veľkých CMOS VLSI obvodov sú funkcie *inverter.m*, *nand.m* a *nor.m*, ktoré som vytvorila pre tieto účely. Tieto skripty využívajú matlabovský ode23 solver, ich vstupnými premennými sú vstupné logické hodnoty a hodnoty veličín prvkov, zapojených v danom obvode. Skripty spustíme v prostredí MATLAB jednoduchým spôsobom. Užívateľsky jednoduché použitie bolo tiež jedným z faktorov pri mojej realizácii implementácie simulácie týchto obvodov. Príklad použitia takéhoto skriptu je nasledovný:

```
inverter(1, 1, 5e-6, 3.3, 0.0005)
nand(1, 1, 1, 5e-6, 3.3, 0.0005)
nor(1, 0, 1, 5e-6, 3.3, 0.0005)

% logic OR
a = 1;
b = 0;
nor = nor(a, b, 1, 5e-6, 3.3, 0.0005);
or  = inverter(nor, 1, 5e-6, 3.3, 0.0005)

% logic OR, simplified expression
inverter(nor(1, 0, 1, 5e-6, 3.3, 0.0005), 1, 5e-6, 3.3, 0.0005)
```

Prvá (alebo prvé dve hodnoty u CMOS NAND a CMOS NOR) vstupná hodnota príslušnej funkcie (skriptu) predstavuje logickú hodnotu, na základe ktorej sa inicializujú hod-

noty rezistorov v obvode. Výstupom funkcie je jediná logická hodnota, odpovedajúca výstupnej hodnote pravdivostnej tabuľky (pre invertor 4.1, pre NAND a NOR 4.2).

Pre demonštráciu implementácie simulácie CMOS obvodov som zvolila hradlo XOR (tiež aj tzv. exkluzívne OR), ktoré sa využíva v CMOS VLSI obvodoch v hojnej miere. Logická operácia XOR je v najzákladnejšom tvare operácia realizovaná na dvoch operandoch (podobne ako NAND a NOR), pričom výsledkom tejto operácie je pravdivostná hodnota 1 vtedy, ak je pravdivostná hodnota 1 pre jeden zo vstupov, ale nie pre oba:

$$XOR(A, B) = A \oplus B = \bar{A}.B + A.\bar{B} \tag{5.5}$$

Pravdivostné hodnoty XOR uvádzam v tabuľke 5.1.

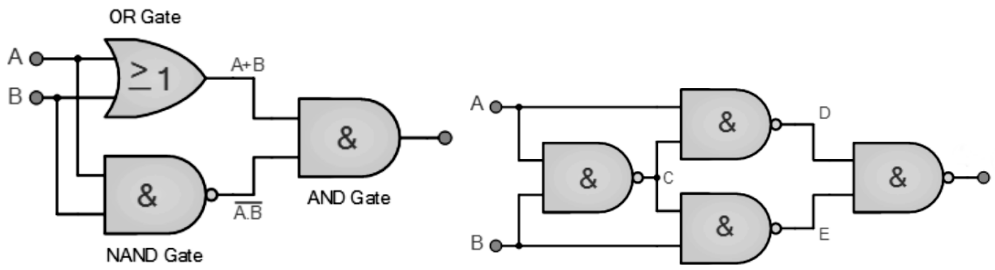
Tabuľka 5.1: Logické hodnoty odpovedajúce vstupnému V_{in} a výstupnému napätiu V_{out} pri využití CMOS XOR.

V_{in}		XOR
1	1	0
1	0	1
0	1	1
0	0	0

Operácia XOR môže byť realizovaná viacerými spôsobmi, v súvislosti s použitými elementárnejšími logickými obvodmi [20]:

- pomocou troch hradiel - AND, OR, NAND obvodov (obrázok 5.10 vľavo),
- pomocou štyroch hradiel NAND (obrázok 5.10 vpravo).

Po pripojení ďalšieho hradla NAND na výstup XOR získame logický obvod vykonávajúci logickú operáciu XNOR.



Obr. 5.10: Realizácia operácie XOR pomocou CMOS hradiel, vľavo pomocou troch hradiel AND OR, NAND, vpravo pomocou štyroch logických obvodov NAND [20].

Vo všeobecnosti sa pri konštruovaní mikroprocesorov uprednostňuje použitie hradiel NAND, z viacerých dôvodov a vlastností, vyplývajúcich zo zapojenia takýchto logických obvodov. Použitie iba jedného druhu logických obvodov (NAND) je výhodnejšie v porovnaní s realizáciou pomocou troch typov obvodov - AND, OR a NAND. Hradlá XOR sa využívajú hlavne pri konštrukcií sčítačiek a iných komponentov.

Simulovať CMOS XOR (pričom nás zatiaľ nezaujíma paralelizácia) pomocou vytvorenej funkcie *nand* môžeme viacerými spôsobmi. Jedným zo spôsobov je uskutočniť potrebné

výpočty postupne, teda definovať pomocné premenné pre výpočet výstupov hradiel NAND, ktoré sú vstupmi pre posledné zo štvorice hradiel, a to nasledovne:

```
in1 = 1;
in2 = 1;
a = nand(in1, in2, 1, 5e-6, 3.3, 0.0005);
b = nand(in1, a, 1, 5e-6, 3.3, 0.0005);
c = nand(a, in2, 1, 5e-6, 3.3, 0.0005);
out = nand(b, c, 1, 5e-6, 3.3, 0.0005)
```

K dispozícii máme i ďalšie možnosti, napríklad použiť skrátený zápis predchodzích výpočtov:

```
in1 = 1;
in2 = 1;
a = nand(in1, in2, 1, 5e-6, 3.3, 0.0005);
out = nand(nand(in1, a, 1, 5e-6, 3.3, 0.0005), nand(a, in2, 1, 5e-6, 3.3, 0.0005),
      1, 5e-6, 3.3, 0.0005)
```

Horeuvedený kód pre operáciu XOR je možné skrátiť na jeden riadok, pričom funkcia *nand* volá sama seba, pre získanie potrebných vstupných hodnôt:

```
in1 = 1;
in2 = 1;
out = nand(nand(in1, nand(in1, in2, 1, 5e-6, 3.3, 0.0005), 1, 5e-6, 3.3, 0.0005),
      nand(nand(in1, in2, 1, 5e-6, 3.3, 0.0005), in2, 1, 5e-6, 3.3, 0.0005), 1,
      5e-6, 3.3, 0.0005)
```

Pri poslednej možnosti výpočtu operácie XOR však funkciu *nand* voláme o jeden krát viac, než je potrebné v predchodzích situáciách. Zaujímavé je preskúmať tieto možnosti, v súvislosti s časom potrebným pre výpočet.

Meranie času výpočtu v prostredí MATLAB pokrýva viacero spôsobov, pričom každý z nich poskytuje iné výsledky. Pre experimenty v tejto práci som zvolila na meranie času doporučovanú funkciu *timeit*, ktorá meria výpočtový čas zvolených funkcií a [28]. Pomocou tejto funkcie som zmerala uskutočnenie operácie XOR, pre každý z horeuvedených spôsobov výpočtu. Táto funkcia uskutočňuje meranie výpočtového času danej funkcie mnohokrát za sebou a následne z nameraných hodnôt určuje priemernú hodnotu, ktorá predstavuje výstup funkcie *timeit*.

Meranie výpočtového času pre CMOS XOR (bez paralelizácie) som pre všetky tri možnosti zápisu realizovala nasledovným spôsobom:

```
in1 = 1;
in2 = 1;

% Option 1
a = nand(in1, in2, 1, 5e-6, 3.3, 0.0005);
b = nand(in1, a, 1, 5e-6, 3.3, 0.0005);
c = nand(a, in2, 1, 5e-6, 3.3, 0.0005);
k = @() nand(in1, in2, 1, 5e-6, 3.3, 0.0005);
l = @() nand(in1, a, 1, 5e-6, 3.3, 0.0005);
m = @() nand(a, in2, 1, 5e-6, 3.3, 0.0005);
out = @() nand(b, c, 1, 5e-6, 3.3, 0.0005);
t1 = timeit(k) + timeit(l) + timeit(m) + timeit(out)

% Option 2
```

```
a = nand(in1, in2, 1, 5e-6, 3.3, 0.0005);
aa = @() nand(in1, in2, 1, 5e-6, 3.3, 0.0005);
out = @() nand(nand(in1, a, 1, 5e-6, 3.3, 0.0005), nand(a, in2, 1, 5e-6, 3.3,
          0.0005), 1, 5e-6, 3.3, 0.0005);
t2 = timeit(aa) + timeit(out)

% Option 3
out = @() nand(nand(in1, nand(in1, in2, 1, 5e-6, 3.3, 0.0005), 1, 5e-6, 3.3,
          0.0005), nand(nand(in1, in2, 1, 5e-6, 3.3, 0.0005), in2, 1, 5e-6, 3.3,
          0.0005), 1, 5e-6, 3.3, 0.0005)
t3 = timeit(out)
```

Meranie pre každú z možností som uskutočnila 60-krát a určila priemer získaných výsledkov. Výsledné hodnoty vykazujú malý ale badateľný rozdiel, z ktorého vyplýva, akým spôsobom je vhodné pracovať s mnou vytvorenými funkciami *invertor, nand*, či *nor*. Najkratší priemerný čas vykazuje prvá možnosť. Realizácia výpočtov pomocou čiastkových výpočtov a ich následné použitie sa javí ako výhodnejšia alternatíva, v porovnaní s volaním funkcie v rámci inej funkcie, kedy je nutné volať tie isté funkcie s tými istými vstupmi viackrát. Otázkou je, aký vplyv má tento spôsob realizácie výpočtov na dostupnú pamäť. To môže byť predmetom ďalšieho skúmania. Výsledné priemerné hodnoty meraní CMOS XOR pomocou mnou implementovanej funkcie *nand* sú v tabuľke 5.2.

Tabuľka 5.2: Priemerné hodnoty meraní simulácie CMOS XOR pomocou vytvorenej matlabovskej funkcie *nand*, pri použití ode23 solvra pre výpočet diferenciálnych rovníc.

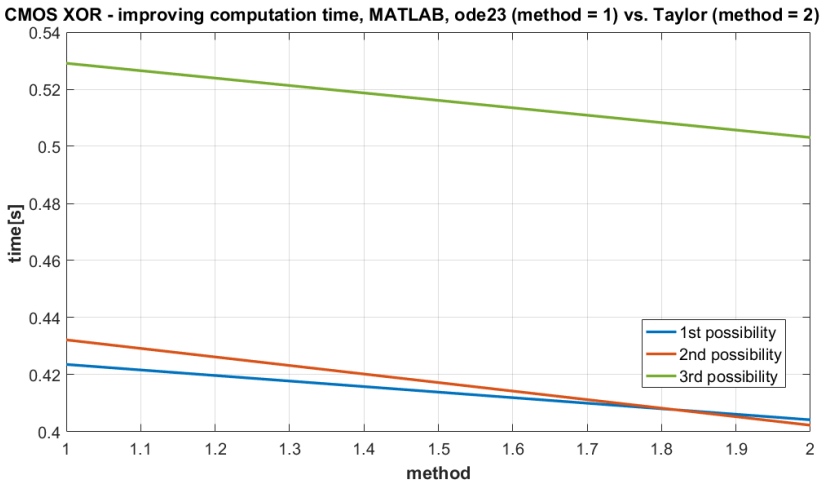
Možnosti zápisu	Výpočtový čas [s]
1.	0.4236
2.	0.4322
3.	0.5291

Funkcie *inverter, nand, nor* využívajú pre svoje numerické výpočty ode23 solver. Zaujímavé je uskutočnenie horeuvedených meraní použitím Taylorovej metódy, kde som presnosť výpočtu nastavila na hodnotu 10^{-10}. Tá je implementovaná v špeciálnych funkciách *inverter_t, nand_t, nor_t*, taktiež jednoducho spustiteľné v MATLAB-e. Výsledky merania výpočtového času pre simulácie CMOS XOR sú pre porovnanie v tabuľke 5.3.

Tabuľka 5.3: Priemerné hodnoty meraní simulácie CMOS XOR pomocou vytvorenej matlabovskej funkcie *nand*, pri použití Taylorovej metódy pri výpočtoch diferenciálnych rovníc.

Možnosti zápisu	Čas [s], eps $= 10^{-10}$	Čas [s], eps $= 10^{-5}$
1.	0.4042	0.3837
2.	0.4023	0.3893
3.	0.5031	0.4781

Zaujímavé je porovnať rýchlosť výpočtov i z hľadiska nastavenia presnosti výpočtov (premenná *eps*) u Taylorovej metódy prípadne iných parametrov. Pri nižšej presnosti predpokladáme kratší čas výpočtu. Pre uskutočnenie meraní času som opäť využila zapojenie



Obr. 5.11: Skrátenie času simulácie CMOS XOR v prostredí MATLAB po implementovaní explicitnej Taylorovej metódy pre uvedené možnosti zápisu pomocou funkcie *nand*.

štyroch hradiel CMOS NAND pre logickú operáciu XOR a otestovala som toto zapojenie obvodu pre vyššie spomenuté možnosti zápisu výpočtu. Porovnanie výsledkov meraní pre simuláciu CMOS XOR, pri presnosti 10^{-10} a 10^{-5}, je v tabuľke 5.3.

Z doterajších výsledkov je možné konštatovať, že simulácia obvodov pomocou metódy Taylorovho radu pre riešenie diferenciálnych rovníc má priaznivý vplyv na výpočtový čas, a to i bez paralelizácie výpočtov. Pri zapojení len štyroch logických obvodov CMOS NAND, metóda Taylorovho radu vykazuje malé zrýchlenie výpočtu. Z uvedených informácií sa dá usudzovať, že pri zapojení veľkého množstva logických obvodov tvoriacich CMOS VLSI obvody sa toto zrýchlenie stane ešte markantnejším.

Tabuľka 5.4: Celkové priemerné hodnoty meraní simulácie CMOS XOR pomocou vytvorenej matlabovskej funkcie *nand*, pri použití ode23 solvra, explicitnej Taylorovej metódy s presnosťou 10^{-10} a 10^{-5}.

Metóda výpočtu	Presnosť	Výpočtový čas [s]
ode23 solver	$AbsTol = 10^{-6}, RelTol = 10^{-3}$	0.4616
explicitná Taylorova metóda	$eps = 10^{-10}$	0.4365
explicitná Taylorova metóda	$eps = 10^{-5}$	0.4170

Keďže je XOR rozšíreným logickým obvodom pri stavbe CMOS VLSI obvodov, pre potreby jednoduchšej simulácie som vytvorila matlabovský skript *xor_.m*, ktorý nahrádza použitie viacerých funkcií *nand* a tým sa stáva z používateľskeho hľadiska príjemnejším. Vďaka absencii opakovaného volania jednej funkcie, inicializácie premenných a uskutočnenia niektorých výpočtov poklesne i výpočtový čas, čím je tento spôsob implementácie CMOS XOR vhodnejší. Prealokácia niektorých polí zabraňuje zmene veľkosti polí počas výpočtov, čím sa šetria výpočtové zdroje a urýchľuje čas simulácie. Pri počte štyroch vzájomne pospájaných hradiel NAND sa v konečnom dôsledku zmenší rozdiel v rýchlosti výpočtov pomocou ode23 solvra v porovnaní s použitím Taylorovej metódy.

Pre CMOS XOR boli vyskúšané viaceré solvre ako ode23, ode45 solver pre porovnanie s explicitnou Taylorovou metódou. Pre každý zo solvrov som vytvorila osobitný skript, ktorý je neskôr použitý na meranie výpočtového času (tabuľka 5.5). Skript *xor_ode45.m* je prispôsobený pre solvre ode23 aj ode45, posledná vstupná hodnota predstavuje prepínač pre voľbu konkrétneho solvra (0 pre ode23).

Tabuľka 5.5: priemerné hodnoty meraní simulácie CMOS XOR implementovanej v jedinej funkcii, pri použití ode23 a ode45 solvra a explicitnej Taylorovej metódy s presnosťou 10^{-10}.

Metóda výpočtu	Výpočtový čas [s]
ode23 solver	0.2051
ode45 solver	0.2166
explicitná Taylorova metóda	0.2029

Všetky merania som uskutočnila s rovnakými parametrami ako v nástroji FOS. Príklady použitia skriptov v MATLAB-e sú nasledovné:

```
xor_(0, 0, 1, 5e-6, 3.3, 0.0005)
xor_ode45(0, 0, 1, 5e-6, 3.3, 0.0005, 1)
```

Paralelizáciu výpočtov som uskutočnila pomocou jazyka *SPMD* (*single program multiple data*) [30]. Existuje viacero možností použitia SPMD, napríklad pomocou distribuovaných dátových typov. Distribuované polia sú prístupné vnútri každého SPMD bloku (tzv. worker) ale i mimo neho. To psokytuje široké možnosti použitia. SPMD však má i obmedzenia, ako napríklad nie je možné v rámci kódu pre paralelizáciu definovanie tzv. anonymnej funkcie ($(@(t,y)A*y+b)$). Preto je nutné všetky potrebné matice vytvoriť pred zahájením paralelizácie. Pre paralelizáciu výpočtov v MATLAB-e existuje i *parfor*.

Pri použití SPMD som opäť implementovala do skriptu *xor_t_spmd* explicitnú Taylorovu metódu. Pri porovnaní simulácie CMOS XOR bes paralelizácie a s ňou sa vyskytol malý rozdiel v hodnotách času trvania výpočtov. Tento rozdiel je priaznivo naklonený skôr k realizácii bez SPMD. Rozdiel vzniká v dôsledku času potrebného na nastavenie prostredia pre paralelizáciu a inicializáciu ďalších premenných. Vzhľadom k ostatným výsledkom meraní času však použitie Taylorovej metódy má nezanedbateľný význam.

Tabuľka 5.6: Priemerné hodnoty meraní simulácie CMOS XOR implementovanej v jedinej funkcii, pri použití explicitnej Taylorovej metódy s presnosťou 10^{-10} v prostredí MATLAB, pri paralelizácii výpočtov s SPMD a bez neho.

Funkcia	SPMD	Výpočtový čas [s]
xor_t	NIE	0.2029
xor_t_spmd	ÁNO	0.3342

Paralelizácia CMOS XOR implementovaná v skripte *xor_t_spmd* je riešená nasledovným spôsobom, v ktorom sa podľa hodnoty intexu daného workera (reprezentujúceho jedno jadro, výpočtovú jednotku) určia príslušné indexy pre prístup do matice reprezentujúcej diferenciálne rovnice pre dva logické obvody CMOS NAND a následne sa vykoná výpočet s adekvátnou numerickou metódou :

```
spmd (num_th) % using two workers, each for one NAND
    b = labindex * num_equations;
    a = b - num_equations + 1;

    A = A_spmd(a:b,:);

    bb = labindex * num_bb;
    aa = bb - num_bb + 1;

    b = b_spmd(aa:bb,:);

    % TAYLOR SERIES
    [T_Taylor, Y_Taylor, ORD, DY] = explicitTaylorLinear(dt, tspan, y0, A, b, eps);

    out_3 = Y_Taylor(:, 2); % uc3 solution
    out_4 = Y_Taylor(:, 3); % uc4 solution

    out_taylor = out_3(end) + out_4(end);
    if out_taylor > U/2
        myout = 1;
    else
        myout = 0;
    end
end % end of parallel computation
```

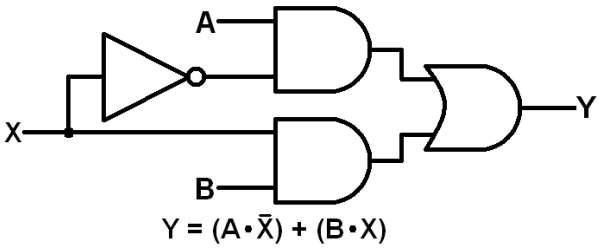
Po ukončení paralelizácie je potrebné pristúpiť k získaným výsledkom (a dokončiť simuláciu CMOS XOR):

```
% get results of parallel computations
out2 = myout{1};
out3 = myout{2};
```

Ďalším príkladom možnosti implementácie simulácie CMOS obvodov je možnosť simulácie základného 1-bit 2-vstup multiplexora (obrázok 5.12), pozostávajúceho z dvoch hradiel AND, jedného OR a NOT hradla, pričom AND i OR dokážeme zostrojiť pomocou NAND, NOR a invertoru [3]. Multiplexor (MUX) je vo všeobecnosti významným elektronickým prvkom, ktorý pracuje s viacerými signálmi. Okrem klasických (signálových) vstupov, ktoré spracuvávam v tejto práci v rámci CMOS obvodov, prijíma i adresové vstupy.

V súvislosti s CMOS VLSI obvodmi, existujú ďaleko zložitejšie logické jednotky, pracujúce i s väčším množstvom vstupov, než spomenutý multiplexor alebo CMOS XOR. Implementácia zložitejších obvodov len pomocou mnou navrhnutých CMOS modelov nie je jednoduchá a vyžaduje ďalšie úpravy i hlbšie technické znalosti z oblasti elektroniky. Z toho dôvodu som pre konkrétne príklady použitia zvolila jednoduchší CMOS XOR, ktorý predstavuje dôležitý základ pri stavbe náročnejších obvodov.

Všetky merania som uskutočnila na 64-bitovom operačnom systéme Windows 7 N, s procesorom Intel Core i5-5300U CPU, 2.3GHz.



Obr. 5.12: Schéma 1-bit 2-vstup multiplexora, zloženého z dvoch logických obvodov AND, jedného OR a jedného NOT hradla [3].

Kapitola 6

Záver a zhodnotenie výsledkov

Jedným z cieľov tejto práce bolo porovnať možnosti simulácie CMOS logických obvodov pomocou rozličných nástrojov. Pre porovnanie boli zvolené nástroje FOS a MATLAB, kde sme tentokrát nepoužili grafické prostredie Simulink, ale využili sme existujúce nástroje, napríklad funkcie pre generovanie matíc, čo je výborný prostriedok, uplatňujúci sa najmä pri paralelizácii výpočtov. Výpočet je možné uskutočniť tak, že po úprave sú na sebe mnohé výpočty nezávislé, čo smeruje k rozdeleniu výpočtov medzi dostupné jadrá počítača (CPU). Vďaka týmto nástrojom je možné implementovať numerické metódy výpočtov, ktoré sú vhodné pri paralelizácii výpočtov.

Využitie numerických výpočtov použitím metódy Taylorovho radu spolu s paralelizáciou výpočtov je vhodným pre simuláciu CMOS VLSI obvodov, ktoré sú zložené i z miliónov komponentov, a vďaka paralelizácie je možné čas trvania výpočtov skrátiť, čím sa simulácie takýchto obvodov stávajú jednoduchšími a dostupnejšími. Moderné systémy kladú stále väčšie nároky na výpočtové zdroje.

Literatúra

[1] *All About Circuits: SPICE Quirks.*
URL
https://www.allaboutcircuits.com/textbook/reference/chpt-7/spice-quirks/

[2] *FOS WEBGUI.*
URL http://www.fit.vutbr.cz/~ikocina/fos/index_en.php

[3] *LogicBlocks Experiment Guide: 2-to-1 Multiplexer.*
URL https://learn.sparkfun.com/tutorials/logicblocks-experiment-guide/7-2-to-1-multiplexer

[4] *NGSPICE Online.*
URL http://www.ngspice.com/

[5] *Simulink Reference: Clock.*
URL
http://www-rohan.sdsu.edu/doc/matlab/toolbox/simulink/slref/clock.html

[6] *SPICE3f5 Manual.*
URL http://www.ngspice.com/spice3f5_doc/index.php

[7] *Vysoce náročné výpočty.*
URL http://www.fit.vutbr.cz/~kunovsky/TKSL/index.html.cs

[8] *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling, TUTORIAL and RATIONALE.* Prosinec 1999.
URL https://www.modelica.org/documents/ModelicaRationale13norev.pdf

[9] *The CMOS Inverter.* 1999.
URL http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee141_f01/Notes/chapter5.pdf

[10] *eCircuit Center: About SPICE.* 2002.
URL http://www.ecircuitcenter.com/AboutSPICE.htm

[11] *TOPSPICE SIMULATION – STEP BY STEP.* 2002.
URL http://www.ecircuitcenter.com/RunSPICE.htm

[12] *Blogs: Smooth MATLAB Graphics.* Srpen 2008.
URL http://blogs.mathworks.com/pick/2008/08/29/smooth-matlab-graphics/

[13] *Laboratorní přípravek IPR.* 2010.
URL http://www.fit.vutbr.cz/~satek/FRVS/index.php?stranka=projekty

[14] *Differential Equations I.* Září 2011.
URL http://www.math.toronto.edu/selick/B44.pdf

[15] *About Pspice.* 2016.
URL http://www.pspice.com/about

[16] *Comparing Matlab, Mathematica and Maple numerical speed for matrix rank calculation.* Září 2016.
URL http://12000.org/my_notes/rankTest/test.htm

[17] *Getting started with Dymola.* Dassault Systèmes AB, Září 2016.
URL https://www.3ds.com/fileadmin/PRODUCTS/CATIA/DYMOLA/PDF/Getting_started_with_Dymola_02.pdf

[18] *CATIA Systems Engineering – Dymola.* 2017.
URL https://www.3ds.com/products-services/catia/products/dymola

[19] *Dymola: C Compiler.* 2017.
URL
https://www.3ds.com/products-services/catia/products/dymola/c-compiler/

[20] *Exclusive-OR Gate Tutorial.* 2017.
URL http://www.electronics-tutorials.ws/logic/logic_7.html

[21] *How Does Maple Compare?* 2017.
URL https://www.maplesoft.com/products/maple/compare/

[22] *L2-Norm.* 2017.
URL http://mathworld.wolfram.com/L2-Norm.html

[23] *Maple: Code Editor.* 2017.
URL https://www.maplesoft.com/products/maple/features/codeedit.aspx

[24] *Maple: CUDA support.* 2017.
URL https://www.maplesoft.com/products/maple/features/cuda.aspx

[25] *MapleSim: A Powerful Modelica Platform.* 2017.
URL http://www.maplesoft.com/products/maplesim/modelica.aspx

[26] *MapleSim: Multidomain Modeling with Built-in Components.* 2017.
URL https://www.maplesoft.com/products/maplesim/features/built_in_components.aspx

[27] *Matlab documentation: equationsToMatrix.* 2017.
URL https://www.mathworks.com/help/symbolic/equationstomatrix.html

[28] *Matlab documentation: Measure Performance of Your Program.* 2017.
URL https://www.mathworks.com/help/matlab/matlab_prog/measure-performance-of-your-program.html

[29] *Matlab documentation: ode23.* 2017.
URL https://www.mathworks.com/help/matlab/ref/ode23.html

[30] *Matlab documentation: spmd*. 2017.
URL https://www.mathworks.com/help/distcomp/spmd.html

[31] *MATLAB: Model-Based Design*. 2017.
URL https://www.mathworks.com/solutions/model-based-design.html

[32] *MATLAB: Object-Oriented Programming*. 2017.
URL
http://www.mathworks.com/help/matlab/object-oriented-programming.html

[33] *MATLAB: Simulink*. 2017.
URL https://www.mathworks.com/products/simulink.html

[34] *MATLAB: The Language of Technical Computing*. 2017.
URL https://www.mathworks.com/products/matlab.html

[35] *What is Maple: Product Features*. 2017.
URL https://www.maplesoft.com/products/Maple/features/

[36] *What is MathML 2.0?* 2017.
URL http://www.maplesoft.com/standards/MathML/info.html

[37] Doc. Ing. Jiří Kunovský, CSc.: *Materiály k predmetu Vysoce náročné výpočty*. FIT VUT, Brno, 2016.
URL https://www.fit.vutbr.cz/study/courses/VNV/private/prednasky/2016/vnv_2016_02_25.pdf

[38] Doc. Ing. Jiří Kunovský, CSc.: *Materiály k predmetu Vysoce náročné výpočty, metoda snižování řádu derivace*. FIT VUT, Brno, 2017.
URL https://www.fit.vutbr.cz/study/courses/VNV/private/prednasky/2017/vnv_2017_02_09.pdf

[39] Ernst Hairer and Gerhard Wanner and Syvert P. Nørsett: *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2008, ISBN 978-3-540-56670-0.
URL https://profs.info.uaic.ro/~fliacob/An2/2014-2015/Resurse_MCM_2015/Hairer,%20Norsett,%20Wanner_Solving%20Ordinary%20Differential%20Equations%20I.pdf

[40] Filip Kocina and Gabriela Nečasová and Petr Veigend and Jan Chaloupka and Václav Šátek and Jiří Kunovský: *Modelling VLSI Circuits Using Taylor Series*. 2016.

[41] Jiří Kunovský: *Modern Taylor Series Method*. Faculty of Electrical Engineering and Computer Science, Technical University of Brno, 1994, habilitation work.

[42] Kaung Myat Win and Sheila P. Werth: *RLC Circuits Tutorial - Transient Analysis*.
URL http://ece.wpi.edu/mathworks/ece2010_tutorial3.pdf

[43] Lukáš Kraicinger: *Simulátor logických obvodů*.
URL http://www.stud.fit.vutbr.cz/~xkraic00/

[44] Maplesoft, a division of Waterloo Maple Inc.: *MapleSim User's Guide*. 2014.
URL https://www.maplesoft.com/documentation_center/maplesim6/MapleSimUserGuide.pdf

[45] Mgr. Irena Růžičková and RNDr. Rudolf Hlavička, CSc.: *Skriptá k predmetu Numerické metody.*
URL http://physics.ujep.cz/~jskvor/NME/DalsiSkripta/Numerika.pdf

[46] Milan Kubíček and Miroslava Dubcová and Drahoslava Janovská: *Numerické metody a algoritmy.* 2005.

[47] Milan Murina: *Teorie obvodů.* 2002, ISBN 80-214-222081-2.

[48] Neil H. E. Weste and David Money Harris: *CMOS VLSI Design: A Circuits and Systems Perspective (4th Edition).* 2011.
URL http://ic.sjtu.edu.cn/ic/dic/wp-content/uploads/sites/10/2013/04/CMOS-VLSI-design.pdf

[49] Nenzi, P.; Vogt, H.: *Ngspice Users Manual, Version 26plus.* Únor 2014.
URL http://ngspice.sourceforge.net/docs/ngspice-manual.pdf

[50] Peter Fritzson and Olena Rogovchenko: *Introduction to Object-Oriented Modeling, Simulation and Control with Modelica.* Únor 2012.
URL https://openmodelica.org/images/docs/userdocs/modprod2012-tutorial1-Peter-Fritzson-ModelicaTutorial.pdf

[51] Šátek, V.; Kocina, F.; Kunovský, J.; aj.: Taylor Series Based Solution of Linear ODE Systems and MATLAB Solvers Comparison. In *MATHMOD VIENNA 2015 - 8th Vienna Conference on Mathematical Modelling*, ARGESIM REPORT No. 44, ARGE Simulation News, 2015, ISBN 978-3-901608-46-9, s. 693–694.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10795

[52] Steven Givant and Paul Halmos: *Introduction to Boolean Algebras.* 2009.

[53] Zdeněk Biolek: *Úvod do SPICE pomocí programu MicroCap.* 2009.

Prílohy

Príloha A

Obsah priloženého pamäťového média