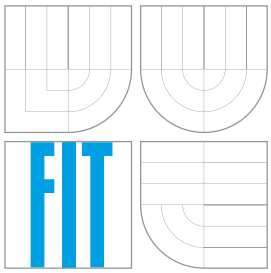
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

AUTOMATICKÉ ŘÍZENÍ VÝPOČTU

AUTOMATIC COMPUTATION CONTROL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN OPÁLKA

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. JIŘÍ KUNOVSKÝ, CSc.

BRNO 2014

Abstrakt

Práce se zabývá automatizací řízení numerických výpočtů. Nejprve je čtenář seznámen s numerickým řešením diferenciálních rovnic a paralelním, sériovým a sériově-paralelním numerickým integrátorem. Praktickým cílem práce je návrh řídících obvodů pro tři zmíněné varianty integrátorů. Součástí návrhu je i tvorba programového simulátoru řídícího obvodu sériově-paralelního integrátoru v pevné řádové čárce.

Abstract

This work deals with the automatic control of numerical calculations. The reader is acquainted with the numerical solution of differential equations and the parallel, serial, and series-parallel numerical integrator. The practical aim of this work is to design control circuits for the three mentioned variants of integrators. The design includes the development of a software simulator of the control circuit for the series-parallel integrator in a fixed point.

Klíčová slova

derivace, diferenciální rovnice, Taylorova ředa, mikroprocesor, registr, numerická integrace, pevná řádévá čárka, pohyblivá řádová čárka, integrátor, řadič, boothův algoritmus

Keywords

derivation, diferential equnation, Taylor series, mikroprocessor, register, numeric integration, fixed point, floating point, integrator, controller, Booth algorithm

Citace

Jan Opálka: Automatické řízení výpočtu, bakalářská práce, Brno, FIT VUT v Brně, 2014

Automatické řízení výpočtu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Doc. Ing. Jiřího Kunovského CSc. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Opálka
20. května 2014

Poděkování

Za odborné vedení mé bakalářské práce bych chtěl poděkovat svému školiteli Doc. Ing. Jiřímu Kunovskému, CSc., který mi svojí podporou, cennými radami, věcnými připomínkami, náměty a komentáři pomáhal směřovat práci ke zdárnému cíli. Mé poděkování patří také rodině, kamarádovi a konzultantovi Petrovi a všem ostatním, kteří mě po dobu studia jakkoliv podporovali a bez nichž by tato práce nemohla vzniknout.

Obsah

Kapitola 1

Úvod

Numerická integrace se využívá při numerickém řešení diferenciálních rovnic. Pro výpočet numerické integrace se používají integrátory – výpočetní jednotky vykonávající základní matematické operace. Algoritmus integrátoru může být zpracováván automaticky jeho řídícím systémem – řadičem, který synchronizuje jednotlivé operace, stanovuje jejich návaznost a řeší případné konflikty. Cílem této práce je navrhnout tři varianty řadičů k již vyvinutým numerický integrátorům pro řešení obyčejných diferenciálních rovnic (ODR).

Nejprve se seznámíme s problematikou numerické integrace (kapitola 2). Shrneme si nejčastěji používané numerické metody (2.1.3 – 2.1.4) a uvedeme numerickou metodu (2.2) založenou na Taylorově řadě, která je používaná v integrátorech určených pro řešení ODR. Popíšeme si obecné numerické integrátory (kapitola 3). Následně si rozebereme strukturu a funkci sériově–sériového integrátoru (3.3), sériově–paralelního integrátoru (3.1) a paralelně–paralelního integrátoru (3.2), popsaných v disertaci [16], pro které budeme navrhovat řídící systémy (řadiče). Pro zjednodušení a zefektivnění operace násobení používají tyto integrátory Boothův algoritmus (3.4). V práci se zabýváme moderním konceptem řadiče (kapitola 4), rozebíráme jeho logickou stavbu a blíže se zabýváme některými významnými bloky jeho struktury (4.1).

Při návrhu řadičů se zaměříme na variantu pro sériově–paralelní integrátor. Od této varianty řadiče se odvodí varianty pro PPI a SSI. Analyzujeme nezbytné řídící signály sériově–paralelního integrátoru (4.3). Pomocí výsledku této analýzy vytvoříme návrh tvaru mikroinstrukcí (4.2), ze kterých sestavíme mikroprogram pro automatické řízení výpočtu. K představení funkce řadiče sériově–paralelního integrátoru je vytvořen programový simulátor (5), který byl navržen názorně a jednoduše tak, aby mohl sloužit při výuce numerické integrace v předmětech ITO, IPR a VNV na FIT VUT v Brně.

Kapitola 2

Numerická integrace

Tato kapitola uvádí základní poznatky o numerické integraci a jejích metodách, které mohou být realizovány elementárním mikroprocesorem.

Numerická integrace se využívá tam, kde analytické řešení není možné. Výsledky získané metodami numerické integrace jsou pouze přibližné. Míra přesnosti výsledku je ovlivnitelná parametry metody. Rozlišujeme dvě hlavní skupiny numerických metod – jednokrokové a vícekrokové.

Ve velkém množství publikovaných numerických metod je zpozorovatelná odlišnost ve dvou kritériích – *přesnost* a *rychlost*. Podle potřeby je vždy jedno potlačeno na úkor druhého.

2.1 Numerické metody pro řešení diferenciálních rovnic

Obecný výpočet diferenciální rovnice za pomoci *jednokrokových metod* se skládá z iterací. Jednokrokové metody využívají hodnotu nalezenou v kroku t_i pro výpočet bodu t_{i+1}. Při rozběhu berou metody jako první hodnotu zadanou počáteční podmínku. Tyto metody jsou vhodné pro řešení obyčejných diferenciálních rovnic prvního řádu a soustav těchto rovnic. Rovnice vyšších řádů a parciální diferenciální rovnice pro numerickou integraci musí být převedeny na soustavu obyčejných diferenciálních rovnic prvního řádu.

K tomuto účelu slouží postupy jako: *metoda snižování řádu derivace* [14] nebo *metoda postupné integrace*. Pro metodu snižování řádu derivace, která je jednodušší, je důležité, aby na pravé straně diferenciální rovnice nebyly derivace vstupu. Metoda postupné integrace je vhodná pro rovnice s derivacemi vstupů na pravé straně.

Pro *vícekrokové metody* platí, že k výpočtu aktuálního řešení používají k předcházejících výsledků. Mezi vícekrokové metody patří např. Adams–Bashforth nebo metody typu prediktor–korektor. U většiny těchto metod je problémem takzvané nastartování metody, kdy je vyžadováno pro výpočet následujícího členu k předcházejících výsledků. Pro dosažení potřebného počtu předchozích výsledků se využije některá z metod jednokrokových.

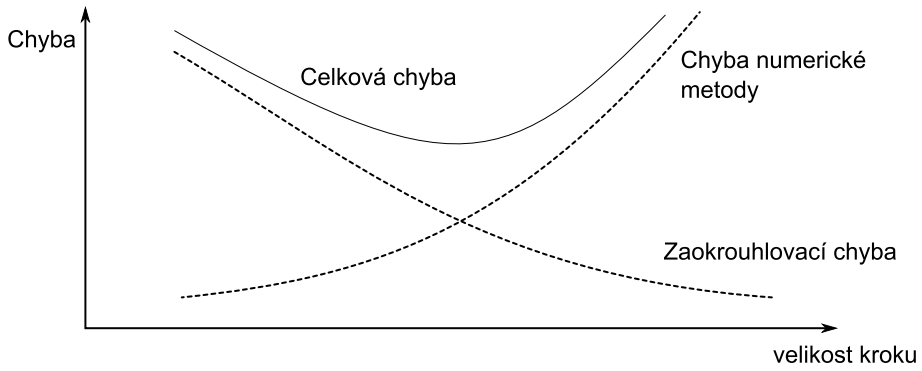
Ke speciálním metodám numerické integrace se řadí metody pro rovnice druhého řádu, metody založené na užití vyšších derivací a další. Existují i tzv. okrajové úlohy, které vznikají z některých úloh matematické fyziky a jsou tvořeny diferenciálními rovnicemi druhého řadu. Tyto metody nebudou v této práci dále rozebírány.

Nyní si popíšeme nejpoužívanější metody řešení diferenciálních rovnic (viz [3], [4], [6]).

2.1.1 Vlastnosti integračních metod

Nejpodstatnější vlastností numerických integračních metod je přesnost jejich výpočtů, neboli velikost chyby, kterou je výsledek ovlivněn. Velmi důležité vlastnosti jsou i stabilita numerického řešení, konvergence řešení ke správnému výsledku a rychlost metody. Nezanedbatelná je i jednoduchost programové implementace numerických integračních metod a také podmíněný počet bodů pro začátek řešení (více v [1], [13], [19]).

Chyba numerické metody se odvíjí od několika faktorů. Prvním z nich je lokální chyba, ke které dochází při každém kroku výpočtu. Lokální chyba je tvořena dvěma částmi – zaokrouhlovací chybou (odvíjí se od přesnosti hardwarových komponent počítače) a chybou numerické metody (určena použitým řádem Taylorovy řady – čím vyšší počet členů Taylorovy řady použijeme, tím vyšší přesnosti výsledku dostaneme). Velikost lokální chyby je možné ovlivnit velikostí kroku výpočtu numerické integrace. Další faktor působící na velikost celkové chyby je akumulace chyby, kdy dochází ke sčítání chyb jednotlivých kroků.



Obrázek 2.1: Celková chyba numerického výpočtu a její složky [19]

Stabilita numerické metody vyjadřuje vztah mezi chováním metody a délkou integračního kroku. Při špatné volbě kroku může dojít ke skokové změně chování metody, dochází k znestabilnění řešení a dostáváme zcela chybné výsledky.

2.1.2 Taylorova řada

Za základní jednokrokovou integrační metodu je považována Taylorova řada. Taylorova řada funkce y pro bod $i+1$ vypadá následovně:

$$y_{i+1} = y_i + hy_i' + \frac{h^2}{2!}y_i'' + \frac{h^3}{3!}y_i''' + \cdots + \frac{h^n}{n!}y_i^{(n)} \tag{2.1}$$

Písmenem h označujeme integrační krok. Touto metodou je možné určit požadovanou přesnost výpočtu. Výpočet konkrétní hodnoty y_{i+1} je iteračně prováděn do doby, než je dosažena požadovaná přesnost. Ta je vyhodnocena z relativního rozdílu dvou následujících členů Taylorovy řady. Nevýhodou použití Taylorovy řady je nutnost vyčíslování vyšších derivací.

Od Taylorovy řady se odvozují některé další integrační metody, přičemž se vychází z možnosti vynechat některé členy řady pro přibližné vyjádření výsledku.

2.1.3 Eulerova metoda

Nejjednodušší jednokroková metoda pro numerické řešení obyčejných diferenciálních rovnic s počátečními podmínkami je Eulerova metoda. Vychází z Taylorovy řady. Využívá pouze její první dva členy.

$$y_{i+1} = y_i + hy'_i \tag{2.2}$$

V případě vhodně zvoleného integračního kroku h lze při použití této metody dostat relativně přesné výsledky. Čím menší je integrační krok, tím přesnější je výsledek. Hlavní předností této metody je vynechání výpočtu vyšších derivací Taylorovy řady – jedná se o metodu 1. řádu.

2.1.4 Runge Kutta 4. řádu

V případě požadavku vyšší přesnosti, než jakou poskytuje Eulerova metoda, můžeme využít metody Runge–Kutta. Tyto metody patří do jednokrokových metod. Pro výpočet y_{i+1} je nutné použít pomocné mezivýpočty. Výsledný přírůstek najdeme jako váhový průměr vypočtených hodnot a jejich počet nám udává řád metody. Nevýhodou použití je pracnější postup a pomalejší získání výsledku než u Eulerovy metody. Tudíž se zde projeví zmíněná aplikace protichůdnosti, rychlosti a přesnosti u těchto metod. Ve srovnání s metodami prediktor–korektor mají metody Runge–Kutta malou rychlost. Další nevýhodou metod Runge–Kutta je obtížné určení chyby výpočtu. Jejich velkým přínosem je možnost využít jejich potenciál pro získání počátečních hodnot pro vícekrokové metody.

Obecný tvar jednokrokových metod Runge–Kutta je:

$$y_{i+1} = y_i + h(\omega_1 k_1 + \cdots + \omega_s k_s) \tag{2.3}$$

$$k_1 = f(t_i, y_i)$$

$$k_1 = f(t_i + \alpha_l h, y_i + h\sum_{j=1}^{l-1} \beta_{lj} k_j), \qquad i = 2, \cdots, s$$

- konstanty

  $$\omega_n,\ \alpha_n \, a \, \beta_{lj}$$

  jsou volené podle řádu metody tak, aby získané řešení souhlasilo s Taylorovou řadou v bodě t_{i+1}.

Nejčastěji používanou metodou Runge–Kutta je Runge–Kutta 4. řádu. Obecný tvar používaného vzorce je následující:

$$y_{i+1} = y_i + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k4) \tag{2.4}$$

$$k_1 = f(t_i, y_i)$$

$$k_2 = f(t_{i+1} + \frac{1}{2}h, y_i + \frac{1}{2}hk_1)$$

$$k_3 = f(t_{i+1} + \frac{1}{2}h, y_i + \frac{1}{2}hk_2)$$

$$k_4 = f(t_{i+1} + h, y_i + hk_3)$$

Specifičtější informace o numerické integraci lze nalézt v publikacích zaměřených na numerické řešení diferenciálních rovnic (viz [1], [13]).

2.2 Metoda Taylorovy řady

2.2.1 Jednoduchá diferenciální rovnice

V této podkapitole je využita aplikace metody Taylorovy řady 2.1 na jednoduchou obyčejnou diferenciální rovnici prvního řádu.

$$y' = y_i \qquad y(0) = y_0 \tag{2.5}$$

Rozvinutím Taylorovy řady pro libovolné y_{i+1} získáme:

$$y_{i+1} = y_i + hy_i' + \frac{h^2}{2!}y_i'' + \frac{h^3}{3!}y_i''' + \cdots + \frac{h^p}{p!}y_i^{(p)} \tag{2.6}$$

Dle formule 2.5 ale platí, že $y' = y$ a proto:

$$y = y' = y^{(2)} = y^{(3)} = y^{(4)} = \cdots = y^{(p)} \tag{2.7}$$

Je tedy možné psát přímo:

$$y_{i+1} = y_i + hy_i + \frac{h^2}{2!}y_i + \frac{h^3}{3!}y_i + \cdots + \frac{h^p}{p!}y_i \tag{2.8}$$

Řešení je možné přepsat tímto způsobem:

$$y_{i+1} = y_i + DY1_i + DY2_i + DY3_i + DY4_i + DY5_i + \cdots + DYp_i \tag{2.9}$$

kde význam jednotlivých členů je:

$$DY1_i = hy_i \tag{2.10}$$

$$DY2_i = \frac{h}{2}DY1_i \tag{2.11}$$

$$DY3_i = \frac{h}{3}DY2_i \tag{2.12}$$

$$DY4_i = \frac{h}{4}DY3_i \tag{2.13}$$

$$DY5_i = \frac{h}{5}DY4_i \tag{2.14}$$

$$DYp_i = \frac{h}{p}DY(p-1)_i \tag{2.15}$$

Rovnice je základním předpisem pro vyřešení problému integrace pomocí Taylorovy řady (pro zadanou homogenní diferenciální rovnici). Tato rovnice přináší první možný koncept pro návrh elementárního integračního procesoru (řešeno v diplomové práci [15]). Výsledkem je výpočetní blok, který je schopen zpracovávat nejjednodušší matematické operace (sčítání, odčítání a násobení). Pro tento blok bude v následujících kapitolách řešena automatizace řízení výpočtu.

2.2.2 Soustava diferenciálních rovnic

Následujicí systém soustavy rovnic 2.16 - 2.17

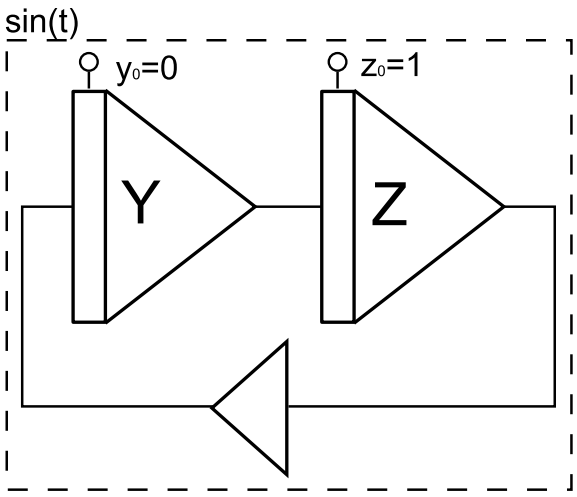
$$y^{'} = z \qquad y(0) = 0 \tag{2.16}$$

$$z^{'} = -y \qquad z(0) = 1 \tag{2.17}$$

popisuje dekomponovanou rovnici 2.18.

$$y^{'} = sint(t) \qquad y(0) = y_0 \tag{2.18}$$

Tuto soustavu rovnic je možné překreslit na invertor a dva integrátory. Invertor je blok, který převrací polaritu vstupního signálu. Integrátor provádí integraci vstupu. Soustava rovnic 2.16 – 2.17, která je zobrazena na následujícím obrázku, popisuje goniometrickou funkci sin.



Obrázek 2.2: Řešená soustava rovnic

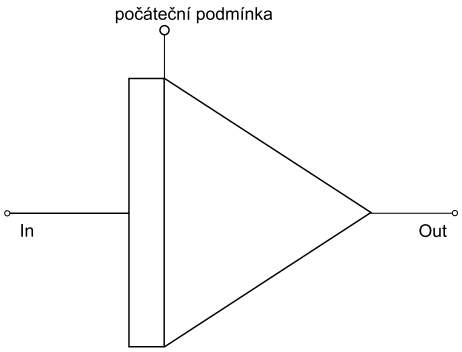
Kapitola 3

Numerický integrátor a jeho varianty

Numerický integrátor je logický obvod, který provádí integraci v číslicovém prostředí. Jako vstup mu slouží hodnoty vstupní proměnné a počáteční podmínky.

Numerické integrátory rozdělujeme na invertující a neinvertující podle toho, zda provádí inverzi znaménka výsledku. Pokud je potřeba dodatečně změnit znaménko výsledku, použije se invertor.

Na následujícím obrázku je blokové schéma numerického integrátoru, které se velmi často používá při grafickém návrhu pro blokové řešení deferencíálních rovnic.



Obrázek 3.1: Blokové schéma integrátoru

Integrátory jsou vytvořeny z výpočetních bloků, které provádějí základní matematické operace. Tyto bloky odpovídají svojí funkcí sčítačkám, odčítačkám, násobičkám nebo děličkám. Dalšími součástmi integrátoru jsou podpůrné obvody, jako například multiplexor pro přepínání kontextu, registry pro uchování dat, obvod řízené negace a blokovací obvod.

Výpočetním základem integrátoru (tedy mikroprocesoru) je aritmeticko–logická jednotka (ALU). ALU je zodpovědná za provedení numerické integrace. Pro výpočet rozsáhlých soustav diferenciálních rovnic 1. řádu je možné propojit více aritmeticko–logických jednotek. Koncepce ALU je zaměřena na repertoár základních matematických instrukcí, které z ALU vytvoří specializovanou výpočetní jednotku (viz [16]).

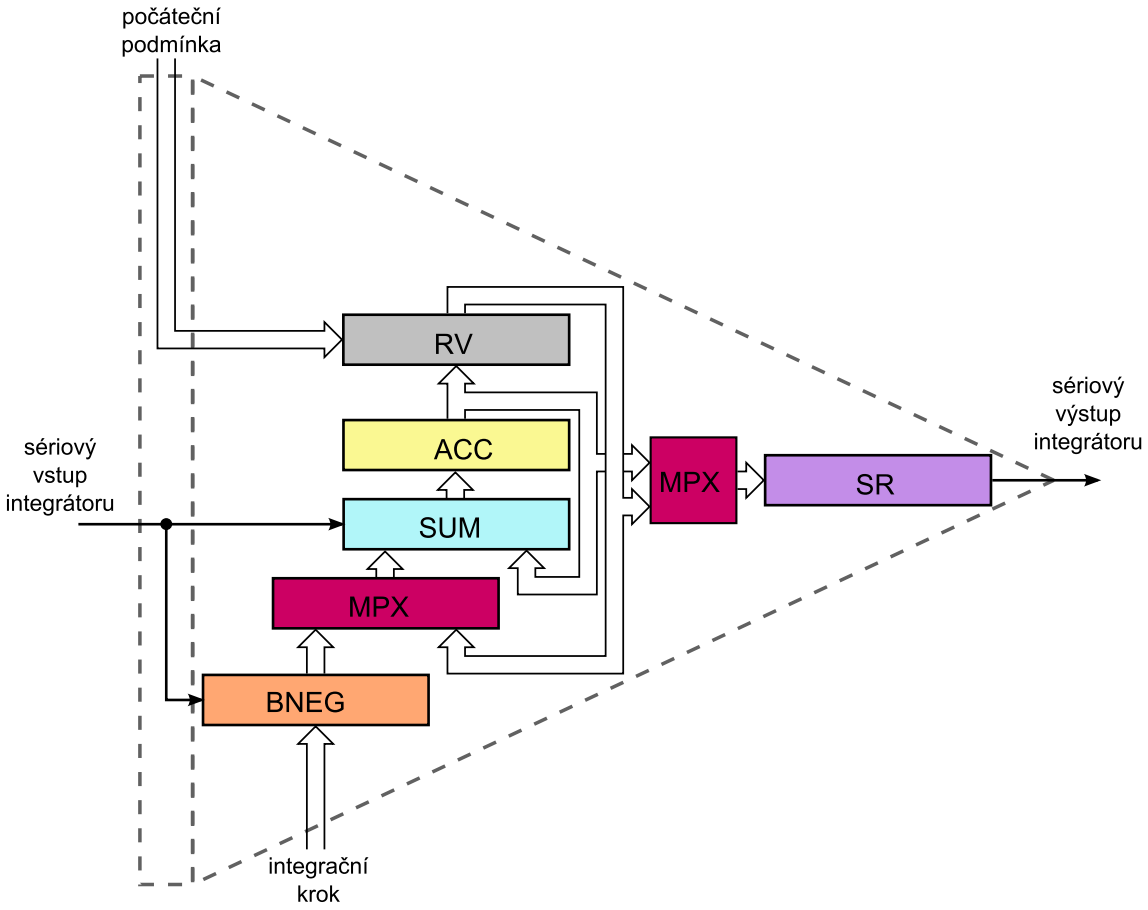
V předchozí kapitole 2 je předvedeno řešení soustavy homogenních lineárních diferenciálních rovnic s konstantními koeficienty. Výpočet těchto diferenciálních rovnic pomocí Taylorovy řady vyžaduje dvě základní matematické operace: násobení pro získávání jednotlivých členů Taylorovy řady (výpočet DY_{pi}) a sčítání jednotlivých členů dohromady (iterativní přičítání nových členů ke stávající hodnotě mezivýsledku). Obě operace mohou být prováděny sériově nebo paralelně. Podle způsobu provádění těchto operací jsou v disertační práci [16] odvozeny názvy navržených integrátorů:

- paralelně–paralelní integrátory (paralelní komunikace a paralelní výpočet, zkratka PPI)
- sériově–paralelní integrátory (sériová komunikace a paralelní výpočet, zkratka SPI)
- sériově–sériové integrátory (sériová komunikace a sériový výpočet, zkratka SSI)

Nyní budou tyto tři integrátory popsány detailně. Obrázky a popisy funkcí byly převzaty z disertace [16]

3.1 Sériově–paralelní integrátor

Tato varianta provádí sekvenční násobení s aplikací Boothova algoritmu. Sčítání je ponecháno v paralelní podobě.



Obrázek 3.2: Blokové schéma sériového–paralelního integrátoru

RV	registr výsledku
MPX	multiplexor
SUM	paralelní sčítačka
ACC	akumulátor
SR	posuvný registr
BNEG	obvod řízené negace (pro Boothův algoritmus)

Princip funkce sério–paralelního integrátoru (SPI) je možné popsat následovně: Na počátku je v registru výsledku (RV) a posuvném registru (SR) uložena počáteční hodnota y_0. Vstup integračního kroku má hodnotu h, multiplexor (MPX) je přepnut na cestu od bloku řízené negace (BNEG). Průběh výpočtu y_{i+1} má tuto sekvenci: Dojde k vynulovaní akumulátoru (ACC) a obvodu malé nuly (MN). Na vstupu SPI je připraven nejméně významný bit $f(y_i)$. Tento bit určuje, zda dojde k odečtení nebo přičtení registru násobence (RN) k obsahu akumulátoru ACC (odečítání se provádí náhradou za dvojkový doplněk) nebo zda bude ignorován (bude přičtena nula). Výsledek se ze sčítačky přenese do akumulátoru. Následuje aritmetický posuv akumulátoru a logický posuv posuvného registru

doprava. Tato sekvence je opakována integrátorem do té doby, dokud není přijat a zpracován poslední bit z $f(y_i)$.

Zmíněným postupem dosáhneme vynásobení kroku h a hodnoty vstupní funkce $f(y_i)$. Výsledek získaný tímto násobením je uložen do posuvného registru SR. Multiplexor musí být přepnut na vstup z registru výsledku RV. Tím je umožněno sečtení obsahu registru RV a obsahu ACC. Výsledek součtu je uložen do registrů RV a SR. Kompletní proces se opakuje do té doby, dokud není dosaženo požadované přesnosti nebo maximálního počtu iterací. Popsaným ukončením procesu získáme y_{i+1}

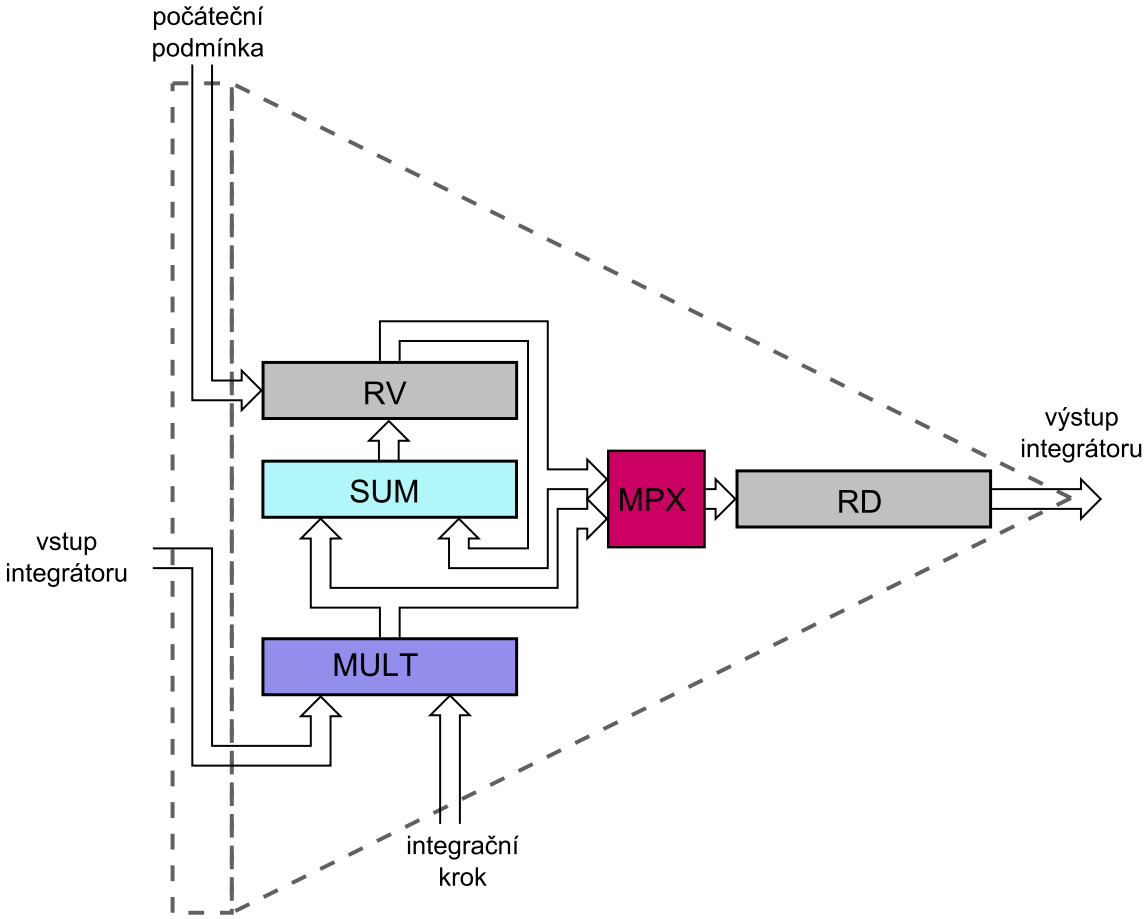
Výhodou sériové komunikace je nízký počet nezbytných vývodů pro rozhraní. Je-li vyžadována přesnější hodnota výsledku, je možné zvětšit počet bitů zobrazení při zachování stávajícího zapojení. Využívá se širších registrů, sčítačky a multiplexoru. Dochází ke zvětšení počtu cyklů nutných pro kompletní výpočet v závislosti na počtu bitů násobence. Výrazná nevýhoda tohoto přístupu oproti PPI je zpomalení násobení, neboť je vykonáváno v n krocích. Čas výpočtu je možné popsat tímto vzorcem uvedeným v [16]:

$$\begin{aligned} t_{SPI} &= \tau_{mult} + \tau_{sum} + \tau_{delay} \\ t_{SPI} &= n * \tau_{sum} + \tau_{sum} + \tau_{delay} \end{aligned} \tag{3.1}$$

Z tohoto vzorce plyne, že celkový čas výpočtu SPI je dán součtem času násobení τ_{mult} (což je n kroků sčítání, kde n je počet bitů násobence), době sečtení výsledku násobení k předchozímu celkovému výsledku na sčítačce τ_{sum} a zohledněním zpoždění signálů mezi vodičích τ_{delay}. SPI by bylo možné rychlostně optimalizovat například překódováním radix 4 či 8 (viz [16]), ale za cenu zesložitění zapojení.

3.2 Paralelně–paralelní integrátor

U této varianty probíhá násobení i sčítání paralelně a to v paralelních výpočetních členech (sčítačka a násobička). Následuje blokové schéma popisující zmíněnou variantu. Čárkovaná čára ohraničuje integrátor.



Obrázek 3.3: Blokové schéma paralelně–paralelního integrátoru

RV	registr výsledku
RD	registr výstupu
MPX	multiplexor
SUM	paralelní sčítačka
MULT	paralelní násobička

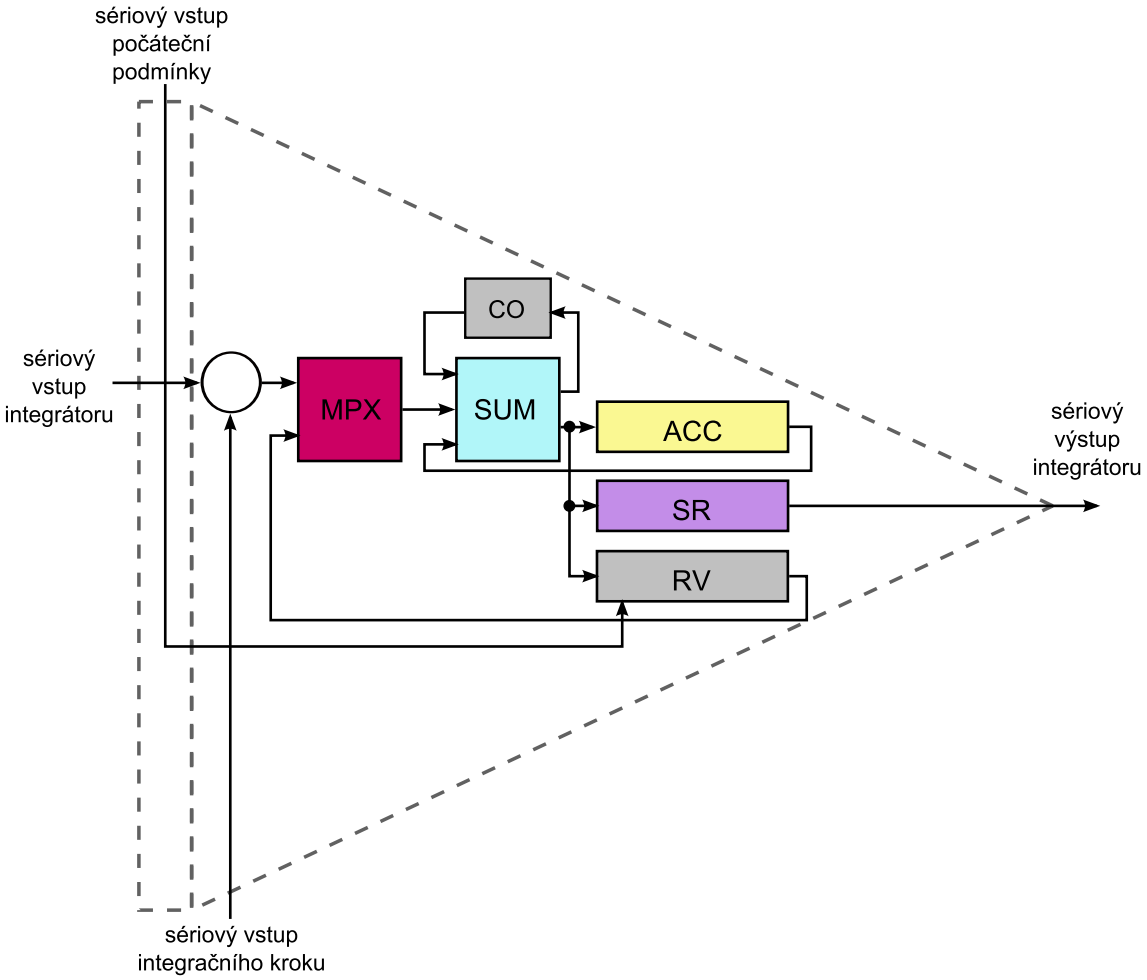
Výpočet se spustí po načtení vstupu y_i do registrů výstupu (RD) a registru výsledku (RV). Na vstup integrátoru se přivede hodnota $f(y_i)$. Integrační krok je nastaven na hodnotu h. Vzniklý součin (na paralelní násobičce MULT) z těchto vstupů se přepíše do RD a současně se obsah RV navýší o tuto hodnotu. Nyní RD obsahuje hodnotu DY_1. Registr výsledku (RV) reprezentuje mezisoučet $y_i + DY_1$. V dalším taktu se na vstupu objeví f(DY_1) a integrační krok se zmenší na hodnotu $h/2$. Jejich zpracováním na násobičce se získá člen DY_2, který je opět uložen do RD a také přičten k RV. Popisovaný cyklus je opakován tak dlouho, dokud se nedosáhne požadované přesnosti nebo maximálního počtu iterací, případně koncového času určeného pro výpočet. Po ukončení získáváme y_{i+1}. Tento typ integrátoru je nejrychlejší ze zmíněných variant, čas výpočtu jednoho členu Taylorovy řady je určen tímto vzorcem [16]:

$$t_{PPI} = \tau_{mult} + \tau_{sum} + \tau_{delay} \tag{3.2}$$

Skládá se tedy ze součtu času potřebného pro násobení τ_{mult} a doby spotřebované pro sčítání τ_{sec}. V případě, že se bude vyskytovat zpoždění signálů v propojovací síti integrátoru, tak se přičte i čas τ_{delay} reflektující tuto skutečnost. Rychlost tohoto zapojení je vykoupena složitostí zapojení, které je nejvíce ovlivněno kombinační násobičkou. Z toho plyne i větší prostorová náročnost této implementace. Mezi další nepříznivá kritéria lze zařadit počet vstupů a výstupu, který je přímo úměrný šířce použité paralelní datové sběrnice.

3.3 Sériově–sériový integrátor

Poslední zmíněná varianta vychází z výše zmíněného principu sériově-paralelního integrátoru. Rozdílem je kompletně sekvenční zpracování jak násobení, tak i sčítání bitů.



Obrázek 3.4: Blokové schéma sériově–sériového integrátoru

RV	posuvný registr výsledku
CO	klopný obvod pro uchovávání přenosu
MPX	multiplexor
ACC	akumulátor
SUM	úplná jednobitová sčítačka
SR	výstupní posuvný registr

Na počátku výpočtu dojde k vynulování akumulátoru (ACC), do registru výsledku (RV) je vložena vstupní hodnota y_i. Následuje vynulování obvodu pro uchování přenosu (CO). Multiplexor (MPX) je přepnut na cestu z RV. Na vstup integrátoru je vsunut nejméně významový bit $f(y_i)$ a na sériovém vstupu integračního kroku se postupně objevují jednotlivé bity integračního kroku h, vzestupně od nejméně významového bitu. V závislosti na hodnotě vstupu může být tato posloupnost přičtena úplnou sériovou sčítačkou (SUM) k hodnotě akumulátoru. Dokončením vyhodnocování mezivýsledku se na vstupu objeví vý-

znamnější bit $f(y_i)$ a současně s touto změnou dochází k posunu výstupního posuvného registru (SR). Naznačený postup je opakován, než je na vstup integrátoru nasunut poslední a tedy nejvýznamnější bit $f(y_i)$. Jakmile je dokončeno násobení, je hodnota DY_p vložena z ACC do SR. Následuje sériové přičtení k hodnotě uchovávané v RV. Tato varianta má nejmenší nároky na složitost zapojení. To je ovšem vykoupeno exponenciálním nárůstem počtu vykonaných cyklů v porovnání s ostatními integrátory.

$$\begin{aligned} t_{SSI} &= \tau_{mult} + \tau_{sum} + \tau_{delay} \\ t_{SSI} &= n * n * \tau_{sum} + n * \tau_{sum} + \tau_{delay} \\ t_{SSI} &= n^2 * \tau_{sum} + n * \tau_{sum} + \tau_{delay} \end{aligned} \tag{3.3}$$

Hodnota τ_{sum} představuje v této verzi integrátoru čas výpočtu úplné jednobitové sčítačky, n je počet bitů registrů, na které jsou uložena data reprezentující násobence a násobitele.

3.4 Boothův algoritmus

Boothův algoritmus je využíván v sériové násobičce v sériově-sériovém a sériově paralelním integrátoru. Slouží pro násobení dvou operandů ve dvojkové soustavě. Je založen na operacích dílčích součtů nebo rozdílů a na operacích posuvu. Při aplikaci násobení základní verzí tohoto algoritmu dochází ke zpracování násobitele porovnáním logických hodnot dvou jeho sousedních bitů (zpracovávaného a předchozího bitu) podle tabulky 3.4 (podle [16]).

K násobení pomocí tohoto algoritmu je využívána kladná i záporná hodnota násobence. Podle aktuálně zpracovávaných bitů násobitele algoritmus rozhodne o hodnotě, která bude přičtena k mezivýsledku (nula, záporný nebo kladný násobenec).

V řídícím zapojení, které bude zpracovávat Boothův algoritmus, bude přítomen pomocný obvod, který obsahuje signály *NEG* a *Cin*. Pomocí těchto signálů může být vytvořen záporný násobenec, což umožní odečtení násobence od mezivýsledku. Dále bude přítomen i signál *BL*, který umožní blokování přičtení násobence k mezivýsledku tím, že vynuluje násobenec, čímž dojde k přičtení nuly k mezivýsledku. K této blokaci přičtení násobence dochází, když hodnoty porovnávaných bitů násobitele jsou stejné.

bity násobitele		
bit_i	bit_{i-1} bit	akce při výpočtu
0	0	přičtení nul k mezivýsledku
0	1	přičtení násobence k mezivýsledku
1	0	odečtení násobence od mezivýsledku
1	1	přičtení nul k mezivýsledku

Tabulka 3.1: Boothův algoritmus s překódováním radix 2 [15]

Test dvou porovnávaných bitů násobitele se provádí za pomoci jednoho vodiče. Využívá se k tomu vstup integrátoru, na kterém se nachází aktuálně zpracovaný bit *Sin*. Předcházející bit je uložen v paměti v obvodu Malá nula (MN) - může jít o klopný obvod typu D.

Jakmile je dokončena akce vyplývající z kombinace porovnávaných bitů, a předtím, než se na vstupu objeví následující bit, je nutné provést uložení aktuálního bitu do obvodu MN, aby mohl sloužit při dalším porovnávání jako předchozí bit.

Existují i další varianty Boothova algoritmu, než kterou se zabýváme v této podkapitole. Jedná se o Boothův algoritmus v jiném překódování (např. s radixem 4, 8 nebo 16). Díky většímu překódování dochází k redukci počtu mezivýsledků při operaci násobení. Proto platí, že čím vyšší radix, tím je výpočet rychlejší. Informace o Boothově algoritmu a jeho variantách je možné nalézt v [21].

Boothův algoritmus násobení bude předveden na příkladu násobení dvou binárních čísel:

$$0101011001 * 0010000000$$

Tato čísla logicky odpovídají *počáteční podmínce* a *kroku* v simulátoru řadiče mikroprocesoru navrženém v této práci. Stručné vysvětlení funkce programu je možné nalézt v kapitole 5.

	0 1 0 1 0 1 1 0 0 1 ≡ 0 1 0 1 0 1 1 0 0 1	
$*$	0 0 1 0 0 0 0 0 0 0	
$+$	1 1 1 0 0 0 0 0 0 0	1 0 (mínus násobenec)
$\sum$	1 1 1 0 0 0 0 0 0 0	
$\rightarrow$	1 1 1 1 0 0 0 0 0 0	
$+$	0 0 1 0 0 0 0 0 0 0	0 1 (plus násobenec)
$\sum$	0 0 0 1 0 0 0 0 0 0	
$\rightarrow$	0 0 0 0 1 0 0 0 0 0	
$+$	0 0 0 0 0 0 0 0 0 0	0 0 (plus nula)
$\sum$	0 0 0 0 1 0 0 0 0 0	
$\rightarrow$	0 0 0 0 0 1 0 0 0 0	
$+$	1 1 1 0 0 0 0 0 0 0	1 0 (mínus násobenec)
$\sum$	1 1 1 0 0 1 0 0 0 0	
$\rightarrow$	1 1 1 1 0 0 1 0 0 0	
$+$	0 0 0 0 0 0 0 0 0 0	1 1 (plus nula)
$\sum$	1 1 1 1 0 0 1 0 0 0	
$\rightarrow$	1 1 1 1 1 0 0 1 0 0	
$+$	0 0 1 0 0 0 0 0 0 0	0 1 (plus násobenec)
$\sum$	0 0 0 1 1 0 0 1 0 0	
$\rightarrow$	0 0 0 0 1 1 0 0 1 0	
$+$	1 1 1 0 0 0 0 0 0 0	1 0 (mínus násobenec)
$\sum$	1 1 1 0 1 1 0 0 1 0	
$\rightarrow$	1 1 1 1 0 1 1 0 0 1	
$+$	0 0 1 0 0 0 0 0 0 0	0 1 (plus násobenec)
$\sum$	0 0 0 1 0 1 1 0 0 1	
$\rightarrow$	0 0 0 0 1 0 1 1 0 0	
$+$	1 1 1 0 0 0 0 0 0 0	1 0 (mínus násobenec)
$\sum$	1 1 1 0 1 0 1 1 0 0	
$\rightarrow$	1 1 1 1 0 1 0 1 1 0	
$+$	0 0 1 0 0 0 0 0 0 0	0 1 (plus násobenec)
$\sum$	0 0 0 1 0 1 0 1 1 0	výsledek

Kapitola 4

Řadič a návrh pro zmíněné varianty integrátorů

V následující kapitole jsou čerpány informace z knihy o logických obvodech [9]. Vnitřní obvody mikroprocesoru, a tedy v našem případě numerického integrátoru, řídí řadič. Je zodpovědný za nastavení cest, zápisy do registrů, čtení z registrů a organizaci posuvů registrů, řízení výpočetních operací ALU, dekódování následujících instrukcí, uchovávání kontextu prováděného mikroprogramu, kontrolu příznakových bitů a asynchronní nulování daných obvodů – například nulovaní akumulátoru.

Existují dva hlavní přístupy ke zhotovování sekvenčního řadiče. Prvním je čistě hardwarové zapojení, které ale v této práci nebude dále rozebíráno. Pozornost bude věnována druhému přístupu, a tím je mikroprogramový koncept. Stejně jako hardwarový řadič je i mikroprogramový řadič složitým sekvenčním obvodem.

Mikroprogramový řadič je vždy řešen konečným automatem jednotlivých instrukcí a je řídícím mozkem celého mikroprocesoru. V jeho řídící paměti jsou uloženy jednotlivé instrukce pro vykonávání vyžadované škály operací v dané ALU. Jednotlivé instrukce a celé mikroprogramy jsou uloženy nejčastěji v paměti typu ROM. Jeden mikroprogramový řadič může v reálném čase vykonávat jeden mikroprogram.

Posloupnosti řídících signálů, které generuje řadič, jsou vyvolány pomocí instrukčního dekodéru (ID). Ten rozlišuje typ instrukce na základě obsahu registru instrukce (IR), který obsahuje aktuálně zpracovávanou instrukci. Po vyhodnocení je nastavena počáteční adresa mikroprogramu. Mikroprogramová koncepce umožňuje snadnou a rychlou modifikaci instrukčního souboru bez dalších zásahů do architektury procesoru. Ovšem výhodou pevného hardwarového sekvenčního konceptu mikroprocesoru je rychlejší činnost, což lze zužitkovat při náročných či zdlouhavých výpočtech.

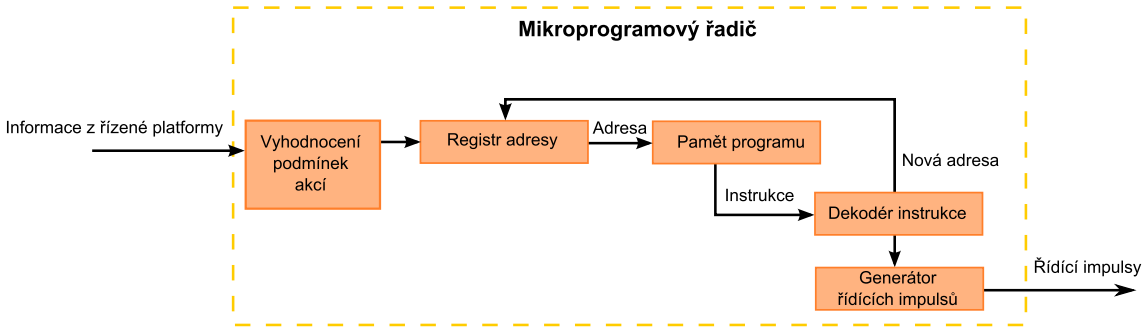
Jako ukazatel na další instrukce slouží řadiči registr zvaný programový registr (PC). Ten je postupně inkrementován a jeho využití spočívá v tom, že umožňuje přečtení všech částí instrukce a na konci každého čtení instrukce informuje řadič o adrese následujícího postupu v mikroprogramu. Tímto způsobem jsou zpracovávány všechny sekvenční instrukce.

Výjimku tvoří instrukce skoku a volání podprogramů. Jejich adresa se dočasně uloží do pomocného registru a následně přesune do PC. Další specialitou jsou podmíněné instrukce. Jsou vykonány pouze v případě, je-li splněna určitá podmínka. K ověřování splnění podmínky slouží stavy jednotlivých bitů stavového registru. Tyto tzv. příznakové bity jsou zavedeny do řadiče, kde jsou využívány pro modifikace toku mikroprogramu. Použití příznakového bitu je dáno tvarem operačního kódu mikroinstrukce. Takovýto postup může

sloužit k implementaci větvení, kdy při splnění nulového výsledku operace dojde k provedení první alternativy toku mikroprogramu. Pokud podmínka splněna není, využívá se druhé alternativy (viz [5],[11] a [20]).

4.1 Struktura řadiče

Na obrázku 4.1 je zjednodušená logická struktura mikroprogramového řadiče (navržená podle schématu na obrázku 25. [9], s. 378).



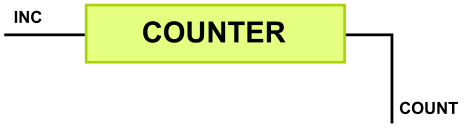
Obrázek 4.1: Logické schéma řadiče

Informace přijímané z řízené platformy (ALU) jsou zpracovávány v bloku *Vyhodnocování podmínek akcí*. Tento blok testuje splnění podmínek instrukcí (podmínky skoku, čítání a zastavení řadiče).

Registr adresy obsahuje adresu mikroinstrukce, která bude zpracována v následujícím taktu hodin řadiče. Registr adresy je také znám pod pojmem čítač instrukcí. Aktuálně zpracovávaná mikroinstrukce se vyhledá v *paměti mikroprogramu*. *Dekodér instrukce* zajistí rozdělení přijaté mikroinstrukce a interpretuje ji podle pro něj známého formátu.

Zpětnou vazbou vkládá dekodér instrukce novou adresu (následující mikroinstrukci) do registru adresy, kde po ověření vnějších podmínek bude zvolen další postup mikroprogramu. Dekodér instrukce předá vyextrahované pokyny z mikroinstrukce *generátoru řídících impulsů*, který má za úkol provést příslušné změny řídících signálů.

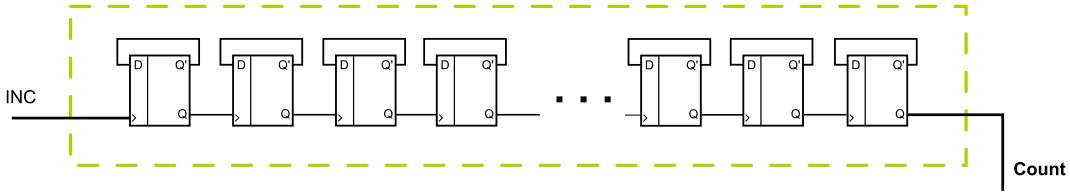
Každý řadič obsahuje několik čítačů, které mu pomáhají při vyhodnocování různých podmínek. Tyto čítače jsou sestaveny z logických bloků.



Obrázek 4.2: Čítač řadiče

Vyobrazený čítač 4.2 je řízen jediným signálem *INC*. K přičtení jedničky k současnému stavu čítače dochází v koicidenci s impulsy na vodiči *INC*. Aktuální stav čítače je možné odečíst na vodiči *COUNT*.

V našem případě bylo navrženo použití čítačů tvořených z klopných obvodů typu D. Jedná se o paměťový člen odvozený z klopného obvodu RS. Ke změně stavu paměťového členu typu D dochází s impulsem na hodinovém vstupu CLK. Další podrobnosti o problematice klopných obvodů a specifika zmíněných členů je možné nalézt v [9].



Obrázek 4.3: Vnitřní struktura čítače

V námi použitém zapojení čítače jsou klopné obvody (KO) vzájemně propojeny vždy výstupem Q předcházejícího KO se vstupem D následujícího KO a výstup Q' předešlého KO s hodinovým vstupem následujícího KO. Počet n klopných obvodů omezuje velikost čítače na hodnotu 2^n (více viz [9], [10]).

4.2 Instrukční sada a návrh formátu instrukce

Instrukční sada je množina instrukcí, kterou používá řadič pro řízení ALU. Existují dva architektonické koncepty instrukčních sad: CISC (Complex Instruction Set Computer) a RISC (Reduced Instructuion Set Computer). CISC obsahuje optimalizované instrukce pro složité výpočetní operace. Architektura RISC naopak poskytuje malý počet relativně jednoduchých instrukcí, což umožňuje rychlejší vykonávání instrukcí oproti architektuře CISC [22]. V našem návrhu používáme architekturu RISC.

Výstupem z paměti mikroprogramu v řadiči je mikroinstrukce. Interpretací této mikroinstrukce se vykoná jedna nebo několik akcí, které mohou byt provedeny současně nebo rozprostřeně na časové ose.

Bylo popsáno několik postupů pro návrh formátu mikroinstrukce. V jednom z historicky prvních formátů každý jednotlivý bit mikroinstrukce odpovídá právě jedné akci, respektive jednomu řídícímu signálu. Tento přístup umožňuje větší rychlost vykonávání mikroprogramu díky menší intenzitě práce s pamětí a díky tomu, že připouští změnu více signálů.

V dnešní době se dává více přednost blokovému rozdělení instrukčního slova. Tyto bloky (skupiny bitů) mohou mít různou délku. Pro blok o n bitech platí, že umožňuje zakódovat až 2^n akcí. Počet bloků určuje nejvyšší možný počet současně vykonávaných akcí [9].

V naší práci se zaměříme na řešení pomocí prvně zmíněného uspořádání, tedy každý bit mikroinstrukce odpovídá právě jednomu řídícímu signálu. Ze schématu 4.4 vyplývá, že pro řídící signály potřebujeme 8 bitů v mikroinstrukci. Každý ze dvou multiplexorů si pro své řízení vyžádá jeden bit. Další 3 bity slouží k nulování registrů (RV, MN) a akumulátoru. Pro řízení aritmetického posuvu akumulátoru a logického posuvu posuvného registru budou použity další 2 bity. Poslední je modifikační bit pro registr výsledku, který přepíná mezi režimem zápisu a čtení.

K výše zmíněným řídícím bitům je nutné ještě připojit pole s adresou následující instrukce. Velikost tohoto pole je dána velikostí paměti mikroprogramu. Pro aplikaci výpočtu $y' = y$ postačí adresové pole o velikosti čtyř bitů.

Pro realizaci skokových instrukcí se na začátek každé instrukce přidá modifikační bit, který oznamuje, zda-li se jedná o instrukci sekvenčního toku mikroprogramu nebo instrukci skoku (podle [9]).

4.3 Analýza nezbytných řídících signálů pro SPI

K řízení funkčních bloků jsou potřeba jednotlivé signály a jejich podrobná analýza. Zaměříme se nejdříve na vyčlenění množiny signálů pro řízení SPI, která částečně vyplývá z přítomných funkčních bloků v zapojení. Od této varianty se potom odvodí zbylé dva řadiče pro varianty PPI a SSI. Pro označení jednotlivých řídících signálů budou požívány často užívané anglické zkratky.

4.3.1 Řízení pro obvod negace

Přes tento obvod vstupuje do Sériově-paralelního integrátoru integrační krok (tedy násobenec). Ze schématu zapojení SPI 3.2 vyplývá, že tento blok poskytne negaci násobence v závislosti na hodnotě sériového vstupu. Vzhledem k použití Boothova algoritmu násobení je potřeba zajistit ještě blokování informace z bloku negace. Proto prvním řídícím signálem ($BLOK$) bude signál pro vytváření nuly a blokování informace z bloku negace. Tento signál se ovšem negeneruje v řadiči, ale na logických členech v obvodu tzv. Malé nuly 4.3.3.

4.3.2 Signály pro multiplexory

Multiplexor přepínající mezi registrem výsledku a blokem negace vyžaduje nastavení pro řízení přepnutí výstupu ($SEL1$). Jeho dva vstupy je možné rozeznat na jednom bitu. Proto tento řídící signál bude jednobitový. Podobná situace je u multiplexoru, který umožnuje výběr dat z registru výsledku nebo akumulátoru. Takže bude mít také jednobitový řídící signál ($SEL2$).

4.3.3 Obvod Malé nuly a posuvný registr

Malá nula je vlastně klopný obvod typu D zvětšující kapacitu posuvného registru o jeden bit (nejméně významný). Více informací o této problematice je možné nalézt v [7].

Pro posuvný registr bude potřeba zajistit řídící signály, které budou přepínat mezi režimem čtení a zápisu (R/W_{SR}). Ze způsobu zpracování výpočtu pomocí Taylorovy řady dále vyplývá požadavek na asynchronní nulování (CLR_{SR}). Všechny tyto řídící signály je nutné rozvést i k obvodu Malé nuly. Jako poslední je zde přítomen bitový příznak reprezentující pokyn k provedení logického posuvu (LSH).

4.3.4 Registr výsledku

Pro registr výsledku bude taktéž přítomný řídící signál pro výběr zápisu nebo čtení (R/W_{RV}). Dále bude přítomné asynchronní nulování pro vynulování tohoto registru (CLR_{RV}).

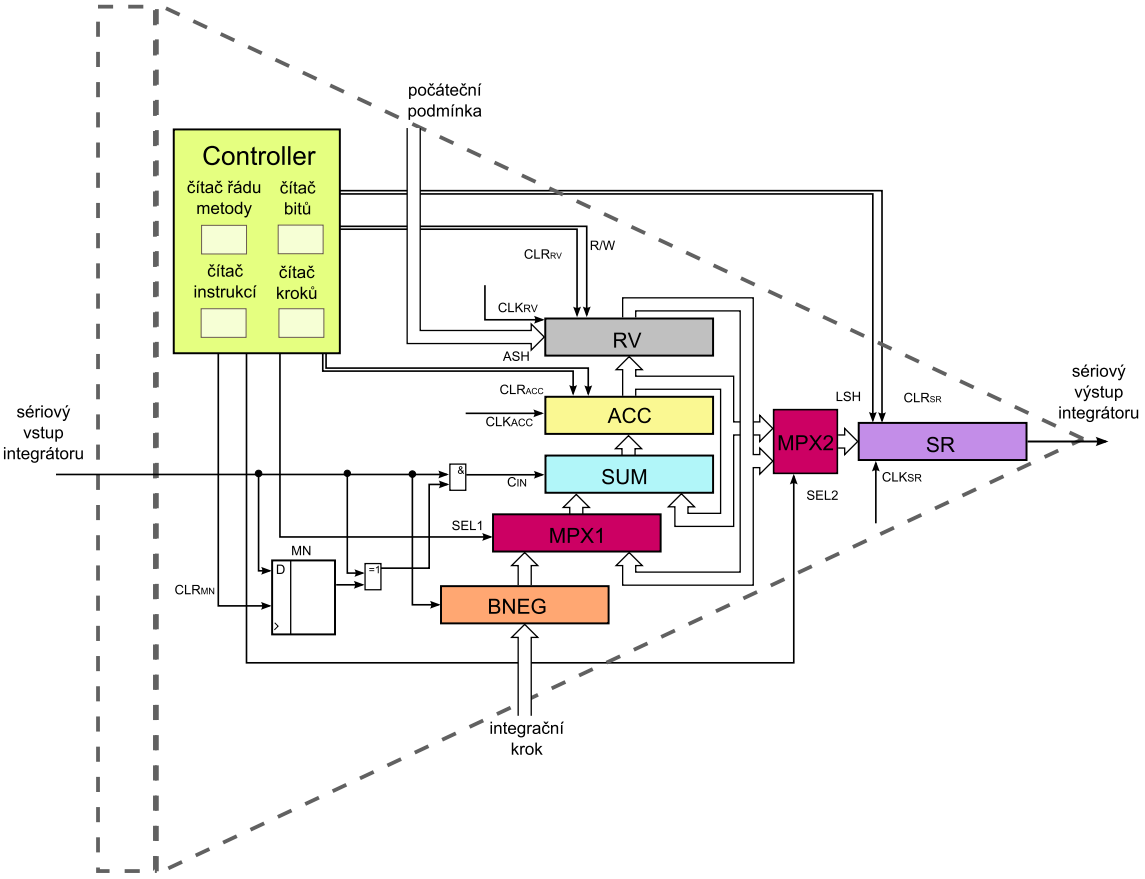
4.3.5 Řídící signály akumulátoru a paralelní sčítačky

Pro paralelní sčítačku je podstatný jeden vedlejší vstup. Jedná se o přenos z nižšího řádu (C_{in}). Vyhodnocuje se logickou operací *and* mezi předcházejícím a aktuálním vstupem.

Pro akumulátor musí existovat alespoň dva řídící jednobitové signály. První z nich bude sloužit pro indikaci požadavku na provedení aritmetického posuvu (ASH). Druhý signál zajišťuje asynchronní nulování akumulátoru (CLR_{ACC}).

4.4 Řadič pro SPI

Obrázek 4.4 zobrazuje vytvořený řadič s řídícími propojeními. Jako základ obrázku byl použit návrh integrátoru v disertaci [16]. Ke schématu integrátoru byl přidán paměťový blok Malá nula (MN), řídící logické obvody a samotný řadič. Schéma respektuje rozbor potřebných signálů 4.3.



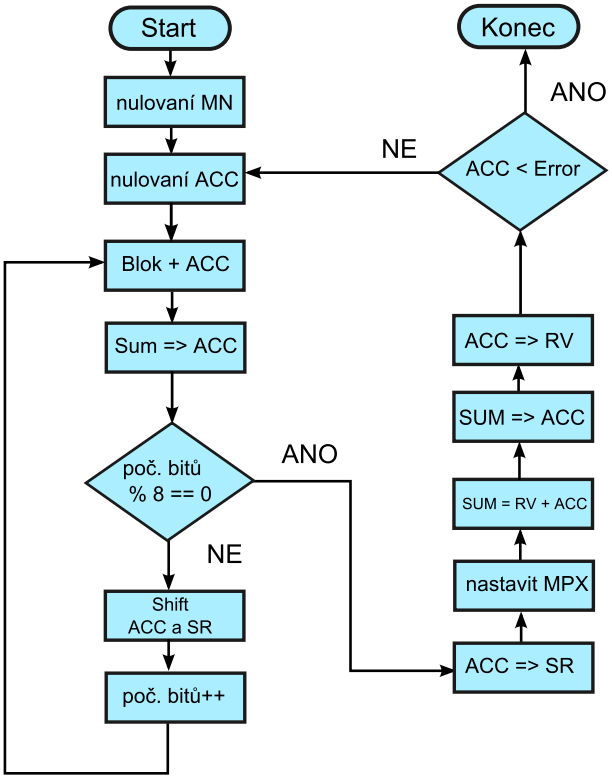
Obrázek 4.4: Blokové schéma sériově–paralelního integrátoru s řadičem

Do výchozího obrázku bylo přidáno deset řídících signálů a hodinové signály pro ACC, RV a SR. Hodinový signál pro SR je přiveden i na obvod MN. Všechny signály jsou označeny jednoduchou čarou s šipkou reprezentující tok řízení. Tyto řídící signály se dají rozdělit do několika podskupin - nulovací, výběrové, posunové, zapisovací a přenos z nižšího řádu.

K nulovacím signálům patří všechny signály, které slouží pro nulování: CLR_{ACC}, CLR_{SR}, CLR_{RV}, CLR_{MN}. Výběrové signály $SEL1$ a $SEL2$ slouží multiplexorům MPX1 a MPX2 pro zvolení správné vstupní sběrnice, na které jsou platná data. Mezi posunové signály patří signál ASH pro aritmetický posun v ACC a signál LSH pro logický posun v SR. Dále zapojení obsahuje signály $R/\overline{W}$ pro povolení zápisu do paměťových bloků. Přenos z nižšího řádu se vyhodnocuje z aktuální a předchozí hodnoty vstupu za pomocí paměťového bloku MN.

4.4.1 Vývojový diagram SPI

Vývojový diagram 4.5 popisuje tok mikroprogramu zpracovávaného v řadiči sério–paralelního integrátoru. Vidíme, že řadič provádí sčítání, porovnání dvou operandů a nulování paměťových bloků. Algoritmus použitý v řadiči pro SPI je podrobně popsán v sekci 4.4.2.



Obrázek 4.5: Vyvojový diagram pro řadič SPI

Uzel *Start* můžeme ztotožnit se začátkem simulace řízení SPI a uzel *Konec* s jejím ukončení. Z vývojového diagramu je patrné, že jsou při výpočtu využity i čítače radiče.

4.4.2 Mikroprogram pro řadič SPI

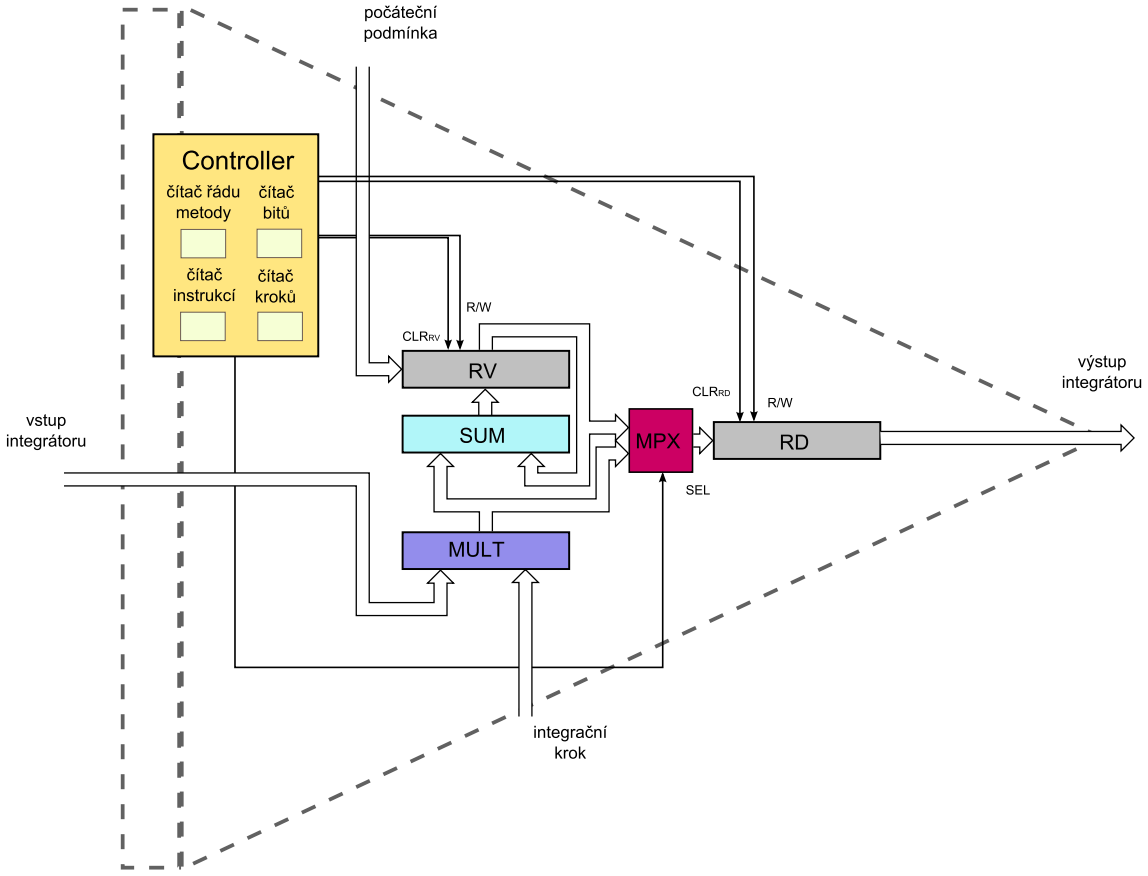
Do paměti mikroprogramu navrženého řadiče bude vložen algoritmus podobný algoritmu v tabulce 4.1. Při jeho inicializaci dojde k vynulování MN a ACC. Následuje sečtení hodnoty z MPX1 s hodnotou ACC. Výsledek se ze SUM přesune do ACC. Provede se test konce násobení pomocí hodnot čítačů *počet bitů* a *počet cyklů* a podle výsledku se buď pokračuje v násobení, nebo se provede uložení hodnoty Dy_i do registru SR. Provede se přepnutí MPX1 a hodnota Dy_i se sečte s hodnotou RV. Výsledek tohoto součtu se uloží zpět do RV.

CLR_{ACC}	CLR_{MN}	$R/\overline{W}_{ACC}$	$R/\overline{W}_{SR}$	$SEL1$	CLK_{ACC}	CLK_{SR}	CLK_{RV}	instrukce	popis funkce
0	1	0	0	1	X	X	X	0001	reset malé nuly
1	0	0	0	1	X	X	X	0010	reset ACC
0	0	0	0	0	X	X	X	0011	MPX + ACC → SUM
0	0	0	0	1	1	X	X	0100	SUM → ACC
if(počet cyklů % počet bitů == 0) 0110 else 0111									test konce násobení
0	0	1	1	1	1	1	X	0001	posuv ACC, SR
0	0	0	0	1	X	1	X	1000	ACC → SR
0	0	0	0	0	X	X	X	1001	přepnutí MPX
0	0	0	0	0	1	X	X	1010	RV + ACC → ACC
0	0	0	0	1	X	X	1	0000	ACC → RV

Tabulka 4.1: Algoritmus činnosti řadiče SPI [15]

4.5 Řadič pro PPI

Řadič vytvářený pro PPI je nejjednodušší z navrhovaných řadičů v této práci. Obsahuje nejmenší počet řídících signálů. Do množiny přidaným signálů k výchozímu zapojení patří výběrový signál *SEL* pro multiplexor, nulovací signály pro registry (registr výsledku a registr výstupu) a signály pro přepnutí mezi zápisem a čtením těchto dvou registrů.



Obrázek 4.6: Blokové schéma paralelně–paralelního integrátoru s řadičem

Předpokládá se přítomnost hodinových signálů pro paměťové bloky (RD a RV), ty ale

z hlediska uchování přehlednosti nebyly zakresleny do schématu 4.6. Zmíněné hodinové signály určují, kdy bude proveden zápis do paměťových bloků.

4.5.1 Mikroprogram pro řadič PPI

Algoritmus uložený v paměti mikroprogramu řadiče určeného pro PPI je znázorněn tabulkou 4.2. Nejdříve se provede nulování registrů RV a RD pomocí nulovacích signálů CLR_x, kde x je označení registru. Pak se nahraje *počáteční podmínka* y_i do registru RV. Z tohoto registru se přenese počáteční podmínka do registru RD. Následuje přepnutí MPX na datovou sběrnici z násobičky (MULT). Další instrukce provede přičtení hodnoty z násobičky k mezivýsledku v RV a vloží hodnotu násobení do RD. Pokračuje se testem splnění podmínek pro ukončení násobení. V případě, že má dojít k výpočtu y_{i+2}, vrací se mikroprogram na začátek, ovšem jako počáteční podmínku volí právě získanou y_{i+1}.

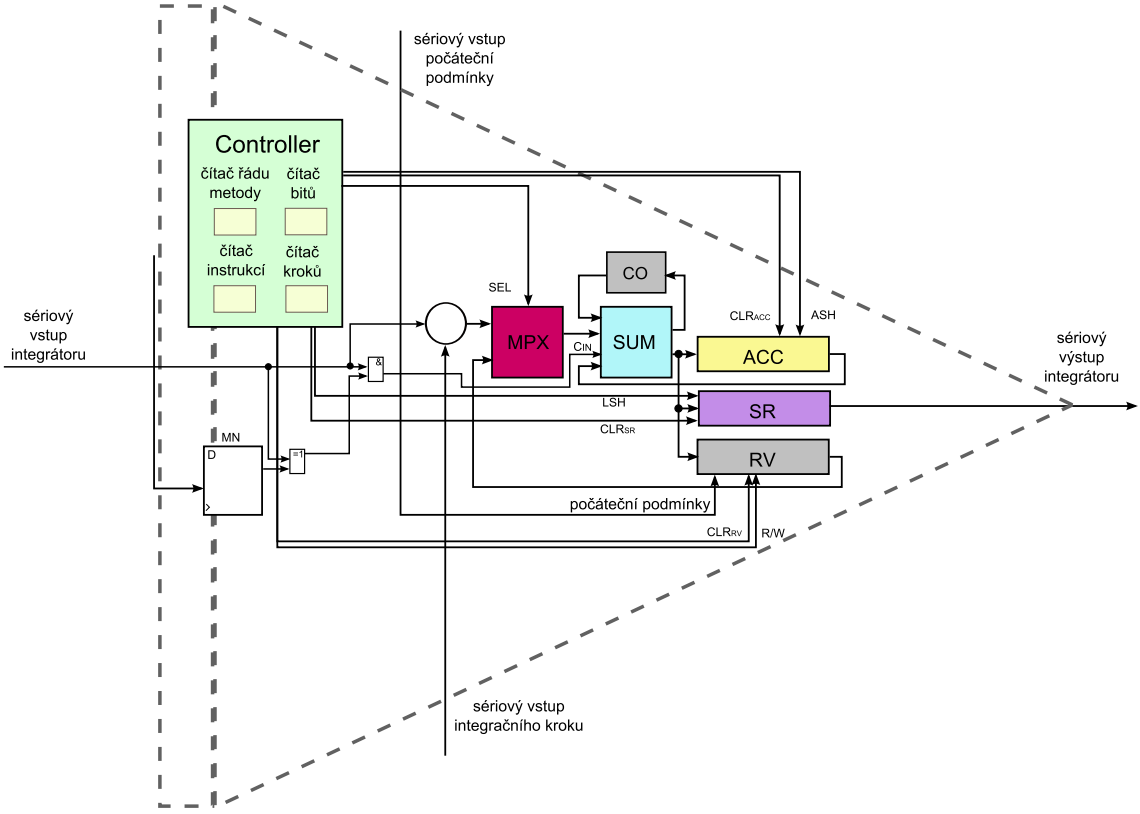
CLR_{RV}	CLR_{RD}	$R/\overline{W}_{RV}$	$R/\overline{W}_{RD}$	SEL	CLK_{RV}	CLK_{RD}	instrukce	popis funkce
1	1	0	0	1	X	X	0001	reset RV a RD
0	0	0	1	1	1	X	0010	y_i do RV
0	0	1	0	1	X	1	0011	RV do RD
0	0	0	0	0	X	X	0100	přepnutí MPX
0	0	0	0	0	X	X	0101	MUL do RD, RV += MUL
if(err < pož or iterace == max. počet iterací or t == t_{end}) 0111 else 0010								test konce násobení
0	0	1	1	1	X	X	1000	výsledek na výstup
if(chceme y_{i+2}) 0000								test konce výpočtu

Tabulka 4.2: Algoritmus činnosti řadiče PPI

4.6 Řadič pro SSI

Nejsložitější řídící zapojení obsahuje varianta řízení sériově–sériového integrátoru. Tento řadič obsahuje nejvíce řídících signálů. Vzhledem k tomu, že zpracování probíhá po bitech, je vysláno také nejvíce řídících impulsů.

I v tomto řešení se vychází z řadiče pro SPI. Řadič pro SSI obsahuje výběrový signál multiplexoru *SEL*, který umožňuje přepínat mezi dvěma vstupy MPX. Pro řízení akumulátoru využívá radič signál pro nulování CLR_{ACC} a aritmetický posuv ASH. Posuvný registr (SR) je taktéž ovládán dvěma signály: nulovacím CLR_{SR} a signálem LSH pro logický posuv. Pro registr výsledku (RV) ještě řadič disponuje signálem $R/\overline{W}$ pro určení režimu zápisu či čtení. Je zde přítomen i nulovací signál registru výsledku CLR_{RV}. Mimo blok řadiče je ještě přidán paměťový obvod malé nuly MN a řídící logický obvod pro výpočet přenosu z nižšího řádu.



Obrázek 4.7: Blokové schéma sériově–sériového integrátoru s řadičem

V zapojení 4.7 nejsou zobrazeny hodinové řídící vstupy u paměťových modulů, aby nedocházelo ke zbytečnému překryvu vodičů. Hodinové řídící signály jsou přivedeny ke všem paměťovým blokům – ACC, SR, RV a MN. Hodinový signál pro SR a MN je totožný.

4.6.1 Mikroprogram pro řadič SSI

Mikroprogram pro řízení SSI je zapsán v tabulce 4.3. Řadič nejdříve inicializuje nulováním ACC, RV a CO. Druhá instrukce vkládá do RV hodnotu počáteční podmínky y_i. Provede se přepnutí MPX na vodič vedený z RV. Bit se přičte k hodnotě akumulátoru. Následuje test zpracování všech bitů počáteční podmínky. Pokud ještě nebyly přičteny všechny bity,

provádí se posun ACC a návrat k přičítání. Tato fáze by se dala zoptimalizovat tím, že při hodnotě 0 na vstupu by se přeskakovalo k dalšímu bitu na vstupu. Pokud byly přičteny všechny bity, pokračuje mikroprogram posunem SR a dalším testem. Ten zkoumá, zda byly zpracovány všechny bity SR - tedy zjišťuje konec násobení. Pokud ne, vrací se zpět. V případě, že je násobení u konce, dochází k načtení obsahu ACC do SR. Poslední část výpočtu spočívá v přičtení členu Taylorovy řady k mezivýsledku v RV, což popisují tři poslední instrukce. V případě žádosti o pokračování výpočtu pro další člen dochází k další iteraci mikroprogramu. Ve druhé instrukci je načtena hodnota y_{i+1} do RV.

CLR_{ACC}	CLR_{RV}	CLR_{SR}	LSH	ASH	SEL	$R/\overline{W}_{ACC}$	$R/\overline{W}_{RV}$	$R/\overline{W}_{SR}$	instrukce	popis funkce
1	1	0	0	0	1	1	1	1	0001	reset ACC, RV, CO
0	0	0	0	0	1	0	1	1	0010	y_i do RV
0	0	0	0	0	0	1	1	1	0011	přepnutí MPX
0	0	0	0	0	1	1	0	0	0100	ACC = SUM + ACC
if(zpracovány všechny bity kroku) 0110 else 0101										test konce násobení bitu
0	0	0	0	1	1	1	1	1	0011	posuv ACC
0	0	0	1	0	1	1	1	1	0111	posuv SR → SR
if(zpracovány všechny bity SR) 1000 else 0011										test konce násobení
hodnota ACC uložena do SR									1001	Dy_p do SR
0	0	0	0	0	0	0	1	1	1010	ACC_i + RV
0	0	0	0	1	0	1	1	1	1011	posun ACC
if(nebyly sečteny všechny bity ACC s RV) 1001 else 0000										test konce sčítaní

Tabulka 4.3: Algoritmus činnosti řadiče SSI

Kapitola 5

Simulátor řízení sériově–paralelního integrátoru

Posledním a důležitým úkolem bakalářské práce bylo zhotovení simulátoru pro sériově–paralelní variantu integrátoru a jejího řadiče. Základním požadavkem na tento program bylo jednoduché a srozumitelné zobrazení všech probíhajících akcí tak, aby bylo možné simulátor využít ve výuce.

Simulátor výpočet skutečně provádí na základě uživatelsky zadaných parametrů. Nejedná se tedy o zobrazení předpřipravené posloupnosti akcí.

Vzhledem k tomu, že nebyl kladen žádný požadavek na konkrétní programovací jazyk, byl zvolen vyšší objektově orientovaný programovací jazyk C#. Tento jazyk spadá do podmnožiny jazyků podporované technologií .NET. C# je založen na jazycích C++ a Java. Vzhledem k tomu, že Java i C++ jsou potomky jazyka C, tak i C# částečně přebírá syntaxi jazyka C. C# i .NET jsou vyvinuty firmou Microsoft [23].

Platformu .NET je možno využít pro vývoj webových aplikací a služeb, pro tvorbu základních návrhových postupů a digramů, vytváření aplikací pro OS Windows (s grafickým uživatelským rozhraním nebo konzolových), realizaci speciálních softwarů pro přenosná zařízení. V dnešní době se objevují pokusy o emulaci platformy .NET i na jiných OS než je Windows.

Jazyk C# i platforma .NET se v současnosti stále vyvíjejí. Jednotlivé verze operačního systému Windows podporují různé verze platformy .NET. Ne vždy je však zajištěna vzájemná kompatibilita. Ke snazšímu vývoji aplikací na této platformě Microsoft vytvořil vývojové prostředí Visual Studio [18], které plně podporuje tvorbu softwarových produktů v programovacích jazycích platformy .NET. Podrobným průvodcem k použití platformy .NET je například [8].

5.1 Analýza a návrh simulátoru

Rozhodli jsme se, že grafickou koncepcí simulátoru navážeme na disertaci [16], protože to považujeme za výhodné pro využití simulátorů ve výuce, ale i pro případné zájemce o hlubší studium problematiky numerické integrace za pomocí Taylorovy řady.

Vzhledem k tomu, že soustavy obyčejných diferenciálních rovnic patří mezi formy popisu spojitých systémů, využívá se v simulátoru spojité simulace.

5.2 Spojitá simulace

Simulace, při které je využíván abstraktní model zakomponovaný v počítačovém programu, se snaží simulovat určitý reálný systém. Úroveň složitosti použitého abstraktního modelu se odvíjí od vlastností systému, které chceme sledovat a nebo jsou pro účel simulace jinak podstatné. Simulací získáváme nové poznatky a informace o chování zkoumaného systému prováděním experimentů s jeho modelem. Zobecnění informací získaných z analýzy poznatků může přinést nové znalosti o simulovaném systému.

Spojitá simulace se zaměřuje na simulaci spojitých systémů. Mezi aplikaci spojité simulace patří: modely řízení systémů v průmyslu, modely elektrických obvodů, modelování přírodních dějů, modely pro biologii a ekologii a také počítačové hry – simulátory dopravy, lodí, letectví.

Mezi formy popisu spojité simulace patří soustavy obyčejných diferenciálních rovnic, soustavy algebraických rovnic, bloková schémata a parciální diferenciální rovnice. Popis může být složen i z kombinací zmíněných forem. Složité spojité simulace vyžadují použití speciálních parciálních diferenciálních rovnic, které vyžadují pro správný běh výpočtu použití náročných matematických postupů [19].

5.3 Realizace simulátoru

Samotná aplikace simulátoru je postavena na WPF technologii z platformy .NET. Simulátor se skládá z logické výpočetní části a z grafického vyobrazení. Vzhledem k tomu, že se má jednat o názornou aplikaci a výukovou pomůcku, je hlavní důraz kladen na část grafického uživatelského rozhraní. Toto rozhraní je vytvořeno v jazyce XAML, který je určen pro efektivní popis a návrh grafického rozhraní aplikací s okny.

Při zobrazování jednotlivých akcí jsou využity animace, které tvoří *Storyboardy*. Jednotlivé události (eventy) reagují na dokončení těchto *Storyboardů* a provádějí změny ve vyobrazených hodnotách. Páteří použitého zobrazování je konečný automat, jež rozhoduje o výběru prováděné akce. Tento přístup následně poskytuje jednoduchou možnost integrace řídících prvků, které mohou být ovládány uživatelem.

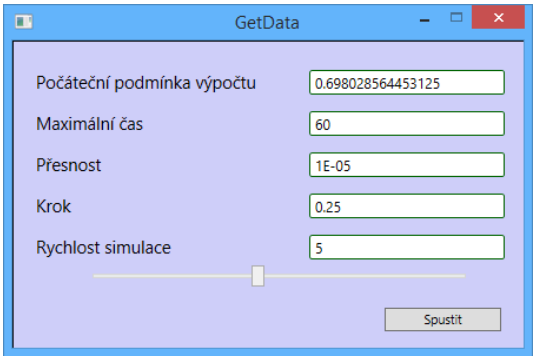
Vnitřní řízení výpočtu je ovlivňováno čtyřmi čítači. Jejich úlohou je počítat počty instrukcí, kroky výpočtu, počty bitů a řád metody. Hodnoty těchto čítačů napomáhají ve vyhodnocování podmínek a dovolují tak zjistit potřebu změny toku mikroprogramu.

Na pozadí za grafickou prezentací běží výpočetně přesná část aplikace. Provádí výpočet Taylorovou řadou na pokyn z řadiče a následně umožní zobrazení nového výsledku uživateli. Přesnost této části je omezena velikostí použitých datových typů, pomocí kterých jsou výpočetní argumenty uchovávány.

5.4 Nastavení počátečních hodnot

Jakmile je spuštěn program, dojde k automatickému nastavení defaultních hodnot ze souboru *App.Config*. Pro změnu těchto parametrů je nutné použít položku z menu *Nastavit parametry*. Objeví se okno, ve kterém se nastavují všechny počáteční podmínky výpočtu numerické integrace. Pro simulaci lze nastavit rychlost r a maximální čas trvání simulace t_{end}. K výpočtu diferenciální rovnice Taylorovou řadou je možné nastavit počáteční podmínku y_0, velikost kroku numerické integrace h a chybu výsledku $error$, kterou je ochoten uživatel akceptovat.

Rychlost a maximální čas simulace jsou akceptovány v celých číslech. Parametry počáteční podmínka, integrační krok a přesnost se zadávají v pevné řádové čárce.



Obrázek 5.1: Okno pro nastavení parametrů simulace

5.5 Základní popis a úvod pro práci s programem

Program poskytuje uživateli základní menu pro nastavení parametrů simulace a výpočtu. Volba *O programu* vypíše informace o aplikaci a jejím autorovi.

Stěžejními ovládacími prvky jsou čtyři tlačítka (popis viz následující tabulka) a posuvník, který ovlivňuje rychlost animace.

Start	spouští automatickou simulaci
Pozastavit	umožňuje pozastavit aplikaci
Krok	umožní simulovat jeden krok výpočtu od aktuální pozice
Reset	nastaví simulátor do původního nastavení

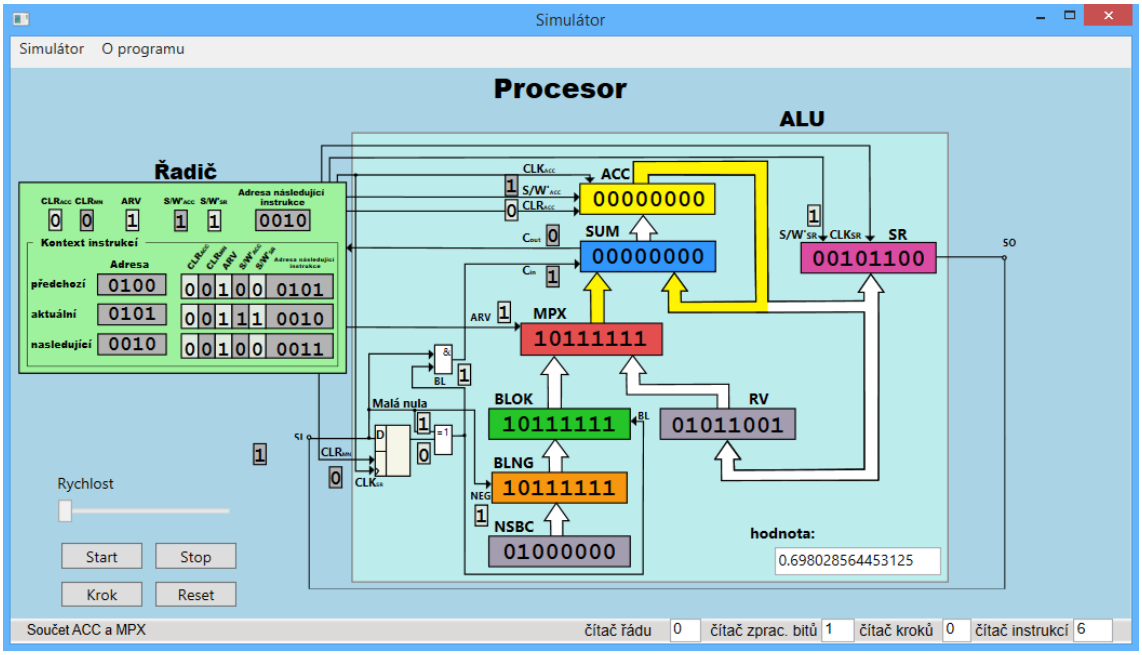
Po ručním nastavení počátečních podmínek dojde k automatickému spuštění simulace 5.2. V defaultním nastavením jsou tyto hodnoty:

$$r = 1 \qquad t_{end} = 60 \qquad y_0 = 0,698028564453125 \qquad h = 0,25 \qquad error = 0,00001$$

V levé části okna simulátoru se nachází zelený blok řadiče. Obsahuje informace získané z aktuálně zpracovávané instrukce a kontext instrukcí mikroprogramu. Mezi získané informace z instrukce patří hodnoty řídících signálů: CLR_{ACC}, CLR_{MN}, ARV, $S/\overline{W}_{ACC}$, $S/\overline{W}_{SR}$. Také z aktuálně zpracovávané instrukce řadič zjistí *adresu následující instrukce*. Kontext instrukcí se skládá ze tří instrukcí - předcházející, aktuálně zpracovávaná a následující instrukce. Vedle označení těchto instrukcí se nachází jejich adresa v paměti mikroprogramu. Tato paměť mikroprogramu je obsažena v řadiči. Následuje pole, které obsahuje kompletní data samotné instrukce, tak jak jsou uložena v paměti mikroprogramu. Tato data instrukce dostane řadič ke zpracování pomocí dekodéru instrukcí a ostatních komponent pro rozklad a analýzu instrukce, aby mohl instrukci provést.

Vlevo pod zobrazeným řadičem pro SPI se nachází stavová informace popisující aktuální operaci, kterou řadič provádí, a stav simulace výpočtu. V pravé části stavového řádku jsou vyobrazeny čítače řadiče, které slouží při řízení toku mikroprogramu.

Ve střední části okna simulátoru je zobrazena ALU. Jednotlivé bloky v ALU reprezentují reálné bloky a zobrazují jejich obsah na osmi bitech. Tento rozsah byl volen proto, aby



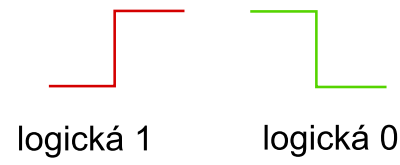
Obrázek 5.2: Zobrazení snímku z běžící simulace

bylo možné zobrazit problematiku, ale příliš nekomplikovat výpočet. Jednobitové vodiče jsou symbolizovány jednoduchou čárou zakončenou šipkou ve směru působení signálu. Ke každému jednobitovému vodiči je vytvořen informační box, který zobrazuje aktuální hodnotu na vodiči. Většina jednobitových vodičů slouží pro řízení synchronizace jednotlivých operací. Tuto synchronizaci provádí řadič pomocí instrukcí. Sběrnice mezi bloky ALU jsou znázorněny širším vodičem s bílou vyplní. Směr přenosu informace u sběrnice je také označen šipkou. Pokud je na sběrnici aktuálně přenášena informace, tak dojde při simulaci ke změně vnitřní výplně na žlutou. Všechny sběrnice v simulátoru jsou osmibitové.

5.6 Nepoužité koncepty

V průběhu tvorby této práce bylo navrženo a zrealizováno několik konceptů simulátoru, přičemž hlavním impulzem pro tvorbu dalších (lepších) variant bylo zvýšení úrovně srozumitelnosti.

Jedním z nepoužitých konceptů je například označení signálových stavů a jejich změn. Původně toto značení bylo řešeno jako časový průběh změny signálu (viz 5.3). Zelený průběh popisoval přechod do nuly a červený průběh oznamoval nastavení signálu na vysokou logickou úroveň. Toto řešení bylo graficky nepřehledné, a proto bylo nahrazeno boxy s číselným popisem aktuálního stavu daného signálu.



Obrázek 5.3: Původní označení signálů

Kapitola 6

Závěr

V této práci proběhlo seznámení s numerickou integrací a nejčastěji používanými numerickými metodami pro řešení ODR. Byly shrnuty jejich důležité vlastnosti. Blíže jsme se zaměřili na užití Taylorovy metody, která tvoří výpočetní základ pro dále zkoumané integrátory.

Teoretickým základem pro hlavní část práce bylo seznámení se s numerickými integrátory SPI, PPI a SSI v provedení pro pevnou řádovou čárku. Tyto integrátory je možné využít při řešení jednoduchých diferenciálních rovnic.

Hlavní částí této práce je analýza integrátorů a následný návrh řídících obvodů těchto integrátorů. Při tvorbě základní verze řadiče se vycházelo ze sériově-paralelního integrátoru. Z této varianty byly odvozeny zbylé dva řadiče. Řadič pro SPI nabízí střední cestu mezi rychlostí a prostorovou složitostí. Představili jsme si základní návrhy tvarů instrukcí a instrukčních sad všech tří variant řadičů. Ke každému řadiči jsme vytvořili jeden mikroprogram, který určuje tok řízení výpočtu.

Řídící obvody i algoritmy, navržené v této práci, mohou být dále optimalizovány. K prezentaci vzniklých řadičů byl vytvořen simulátor varianty řadiče SPI, který umožnuje ověřit funkčnost celého návrhu. Tato práce, společně se vzniklým simulátorem, který je uložen na přiloženém CD, bude využita k hardwarové realizaci. Předpokládá se provedení na hradlových polích. Vzhledem k tomu, že aplikace simulátoru byla navržena s ohledem na názornost a popis algoritmu, je možné zahrnout tyto materiály do výuky hardwarových předmětů.

Tato práce, společně se vzniklým simulátorem, který je uložen na přiloženém CD, bude využita k hardwarové realizaci. Předpokládá se provedení na hradlových polích. Vzhledem k tomu, že aplikace simulátoru byla navržena s ohledem na názornost a popis algoritmu, je možné zahrnout tyto materiály do výuky hardwarových předmětů.

Možností dalšího výzkumu v této oblasti je porovnání časové i prostorové efektivity navržených algoritmů řízení, optimalizace návrhů řadičů včetně vlastní HW implementace. Zajímavým tématem by také mohlo být přenesení návrhu do oblasti výpočtů s pohyblivou řádovou čárkou, s následným ověřením funkčnosti a případnou HW implementací.

Literatura

[1] A. Ralston: *Základy numerické matematiky.* Academia, 1973.

[2] B. Fajmon a I. Růžičková: *Matematika 3.* Skriptum FEKT VUT Brno, 2005.

[3] F. Jirásek a J. Benda: *Matematika: pro bakalářské studium.* Ekopress, první vydání, 2006, ISBN 80-86929-02-7.

[4] J. Brabec a J. Hrůza: *Matematická analýza II.* SNTL - Nakladatelství technické literatury, 1986.

[5] J. Bílek, M. Šnorek a J. Žáček: *Mikroprocesorová technika.* Praha: České vysoké učení technické v Praze, 1988.

[6] J. Diblík a J. Baštinec: *Matematika III.* VUT v Brně, první vydání, 1991, ISBN 80-214-0315-2.

[7] J. Kunovský: *Modern Taylor Series Method.* FEI-VUT Brno, 1994, 115 s., Habilitation work.

[8] J. Liberty & A. Horovitz: *Programming .NET 3.5.* O'Reilly Media, 2008, ISBN 978-0-52756-3.

[9] J. M. Bernard, J. Hugon & R. Le Corver: *Od logických obvodů k mikroprocesorům.* Praha: SNTL - Nakladatelství technické literatury, 1988.

[10] J. Petřík a K. Rauner: *Elektronika (digitální část).* Plzeň: Vydavatelství Západočeské univerzity v Plzni, 2001, ISBN 80-7082-776-9.

[11] J. Pinker: *Mikroprocesory a mikropočítače: obecné principy konstrukce současných mikroprocesorů a mikropočítačů.* Praha: BEN-technická literatura, 2004, ISBN 80-7300-110-1.

[12] J. Podlešák: *Přístrojové aplikace mikroprocesorů.* Vydavatelství ČVUT, 1999, ISBN 80-01-02004-5.

[13] L. Čermák a R. Hlavička: *Numerické metody.* akademické nakladatelství Cerm, 2005, ISBN 80-214-3071-0.

[14] M. Bobek, J. Haška, I. Serba a M. Lukeš: *Analogové počítače.* SNTL - Nakladatelství technické literatury, 1982.

[15] M. Kraus: *Elementární procesor specializovaného paralelního systému.* diplomová práce, FIT VUT v Brně, Brno, 2006.

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

[17] M. Čambor: *Paralelní řešení parciálních diferenciálních rovnic.* diplomová práce, FIT VUT v Brně, Brno, 2011.

[18] Microsoft: Visual Studio [online]. [cit. 12-4-2014].
URL `<https://www.microsoft.com/cze/msdn/vstudio/>`

[19] P. Peringer: Studijní opora předmětu IMS. Technická zpráva, 2007.

[20] S. Zmrzlý: *Mikroprocesorová technika.* Brno: Vysoké učení technické v Brně, první vydání, 1996, ISBN 80-214-0799-9.

[21] V. Drábek: *Výstavba počítačů.* Vysoké učení technické v Brně, nakladatelství PC-DIR, 1995, ISBN 80-214-0691-7.

[22] V. Dvořák a V. Drábek: *Architektura procesorů.* nakladatelství VUTIUM, první vydání, 1999, ISBN 80-214-1458-8.

[23] Web: Microsoft Developer Network [online].
URL `<http://msdn.microsoft.com>`

Dodatek A

Obsah CD

Přiložené CD obsahuje:

dokumentaci - zdrojové soubory této práce v LaTeXu

simulátor - zdrojové soubory pro aplikaci simulátoru řízení řadiče Sério-paralelního integrátoru